Optimizing Millions of Hyperparameters by Implicit Differentiation

Jonathan Lorraine   Paul Vicol   David Duvenaud

University of Toronto, Vector Institute
{lorraine, pvicol, duvenaud}@cs.toronto.edu

Abstract

We propose an algorithm for inexpensive gradient-based hyperparameter optimization that combines the implicit function theorem (IFT) with efficient inverse Hessian approximations. We present results on the relationship between the IFT and differentiating through optimization, motivating our algorithm. We use the proposed approach to train modern network architectures with millions of weights and *millions of hyperparameters*. We learn a data-augmentation network—where every weight is a hyperparameter tuned for validation performance—that outputs augmented training examples; we learn a distilled dataset where every feature in each data point is a hyperparameter; and we tune millions of regularization hyperparameters. Jointly tuning weights and hyperparameters with our approach is only a few times more costly in memory and compute than standard training.

1 Introduction

Neural networks (NNs) generalization to unseen data crucially depends on hyperparameter choice. Hyperparameter optimization (HO) has a rich history [Schmidhuber, 1987, Bengio, 2000], and achieved recent success in scaling due to gradient-based optimizers [Domke, 2012, Maclaurin et al., 2015, Franceschi et al., 2017, 2018, Shaban et al., 2019, Finn et al., 2017, Rajeswaran et al., 2019, Liu et al., 2018, Grefenstette et al., 2019, Mehra and Hamm, 2019]. There are dozens of regularization techniques to combine in deep learning, and each may have multiple hyperparameters Kukačka et al. [2017]. If we can scale HO to have as many—or more—hyperparameters as parameters, there are various exciting regularization strategies to investigate. For example, we could learn a distilled dataset with a hyperparameter for every feature of each input Maclaurin et al. [2015], Wang et al. [2018], weights on each loss term Ren et al. [2018], Kim and Choi [2018], Zhang et al. [2019], or augmentation on each input Cubuk et al. [2018], Xie et al. [2019].

When the hyperparameters are low-dimensional— e.g., 1-5 dimensions—simple methods, like random search, work; however, these break down for medium-dimensional HO—e.g., 5-100 dimensions. We may use more scalable algorithms like Bayesian Optimization [Močkus, 1975, Snoek et al., 2012, Kandasamy et al., 2019], but this often breaks down for high-dimensional HO—e.g., >100 dimensions. We can solve high-dimensional HO problems locally with gradient-based optimizers, but this is difficult because we must differentiate through the optimized weights as a function of the hyperparameters. Formally, we must approximate the Jacobian of the best-response function of the parameters to the hyperparameters.

We leverage the Implicit Function Theorem (IFT) to compute the optimized validation loss gradient with respect to the hyperparameters—hereafter denoted the *hypergradient*. The IFT requires inverting the training Hessian with respect to the NN weights, which is infeasible for modern, deep networks. Thus, we propose an approximate inverse, motivated by a link to unrolled differentiation Domke [2012] that scales to Hessians of large NNs, is more stable than conjugate gradient Liao et al. [2018], Shaban et al. [2019], and only requires a constant amount of memory.

Finally, when fitting many parameters, the amount of data can limit generalization. There are *ad hoc* rules for partitioning data into training and validation sets— e.g., using 10% for validation. Often, practitioners re-train their models from scratch on the combined training and validation partitions with optimized hyperparameters, which can provide marginal test-time performance increases. We verify empirically that standard partitioning and re-training procedures perform well when fitting few hyperparameters, but break down when fitting many. When fitting many hyperparameters, we need a large validation partition, which makes re-training our model with optimized hyperparameters vital for strong test performance.

Contributions

- We motivate and generalize existing inverse Hessian approximation algorithms by connecting them to iterative optimization algorithms.

- We scale IFT-based hyperparameter optimization to modern, large neural architectures, including AlexNet and LSTM-based language models.
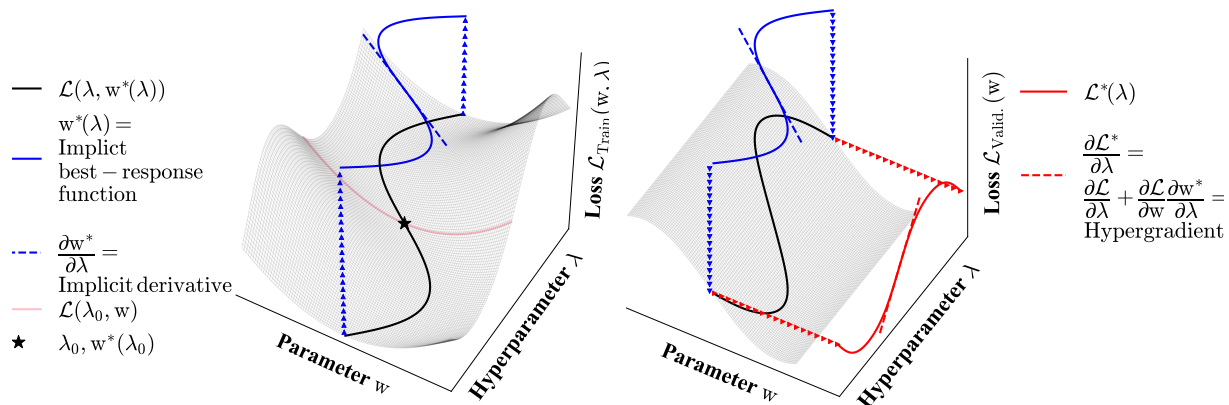
Figure 1: Overview of gradient-based hyperparameter optimization (HO). *Left:* a training loss manifold; *Right:* a validation loss manifold. The implicit function $\mathbf{w}^*(\boldsymbol{\lambda})$ is the best-response of the weights to the hyperparameters and shown in blue projected onto the $(\boldsymbol{\lambda}, \mathbf{w})$-plane. We get our desired objective function $\mathcal{L}_V^*(\boldsymbol{\lambda})$ when the best-response gives the network weights in the validation loss, shown projected on the hyperparameter axis in red. The validation loss does not depend directly on the hyperparameters, as is typical in hyperparameter optimization. Instead, the hyperparameters only affect the validation loss by changing the weights' response. We show the best-response Jacobian in blue, and the hypergradient in red.

- We demonstrate several uses for fitting hyperparameters almost as easily as weights, including per-parameter regularization, data distillation, and learned-from-scratch data augmentation methods.

- We explore how training-validation splits should change when tuning many hyperparameters.

2 Overview of Proposed Algorithm

There are four essential components to understanding our proposed algorithm. Further background is provided in Appendix A, and notation is shown in Table 5.

1. HO is nested optimization: Let $\mathcal{L}_T$ and $\mathcal{L}_V$ denote the training and validation losses, $\mathbf{w}$ the NN weights, and $\boldsymbol{\lambda}$ the hyperparameters. We aim to find optimal hyperparameters $\boldsymbol{\lambda}^*$ such that the NN minimizes the validation loss after training:

$$\boldsymbol{\lambda}^* := \arg\min_{\boldsymbol{\lambda}} \mathcal{L}_V^*(\boldsymbol{\lambda}) \text{ where} \tag{1}$$

$$\mathcal{L}_V^*(\boldsymbol{\lambda}) := \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})) \text{ and } \mathbf{w}^*(\boldsymbol{\lambda}) := \arg\min_{\mathbf{w}} \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w}) \tag{2}$$

Our implicit function is $\mathbf{w}^*(\boldsymbol{\lambda})$, which is the *best-response* of the weights to the hyperparameters. We assume unique solutions to arg min for simplicity.

2. Hypergradients have two terms: For gradient-based HO we want the hypergradient $\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}$, which decomposes into:

$$\underbrace{\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \left(\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} + \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}} \right)\bigg|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})} =$$

$$\underbrace{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}}}_{\text{hyperparam direct grad.}} + \overbrace{\underbrace{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}^*(\boldsymbol{\lambda})}}_{\text{parameter direct grad.}} \times \underbrace{\frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{best-response Jacobian}}}^{\text{hyperparam indirect grad.}} \tag{3}$$

The direct gradient is easy to compute, but the indirect gradient is difficult to compute because we must account for how the optimal weights change with respect to the hyperparameters (i.e., $\frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}$). In HO the direct gradient is often identically 0, necessitating an approximation of the indirect gradient to make any progress (visualized in Fig. 1).

3. We can estimate the implicit best-response with the IFT: We approximate the best-response Jacobian—how the optimal weights change with respect to the hyperparameters—using the IFT (Thm. 1). We present the complete statement in Appendix C, but highlight the key assumptions and results here.

Theorem 1 (Cauchy, Implicit Function Theorem). *If for some $(\boldsymbol{\lambda}', \mathbf{w}')$, $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}', \mathbf{w}'} = 0$ and regularity conditions are satisfied, then surrounding $(\boldsymbol{\lambda}', \mathbf{w}')$ there is a function $\mathbf{w}^*(\boldsymbol{\lambda})$ s.t. $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})} = 0$ and we have:*

$$\frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}'} = - \left[\underbrace{\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}}}_{\text{training Hessian}} \right]^{-1} \times \underbrace{\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}}_{\text{training mixed partials}} \bigg|_{\boldsymbol{\lambda}', \mathbf{w}^*(\boldsymbol{\lambda}')} \text{(IFT)}$$

The condition $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}', \mathbf{w}'} = 0$ is equivalent to $\boldsymbol{\lambda}', \mathbf{w}'$ being a fixed point of the training gradient field. Since $\mathbf{w}^*(\boldsymbol{\lambda}')$ is a fixed point of the training gradient field, we can leverage the IFT to evaluate the best-response Jacobian locally. We only have access to an approximation of the true best-response—denoted $\widehat{\mathbf{w}^*}$—which we can find with gradient descent.

4. Tractable inverse Hessian approximations: To exactly invert a general $m \times m$ Hessian, we often require $\mathcal{O}(m^3)$ operations, which is intractable for the matrix in Eq. IFT in modern NNs. We can efficiently
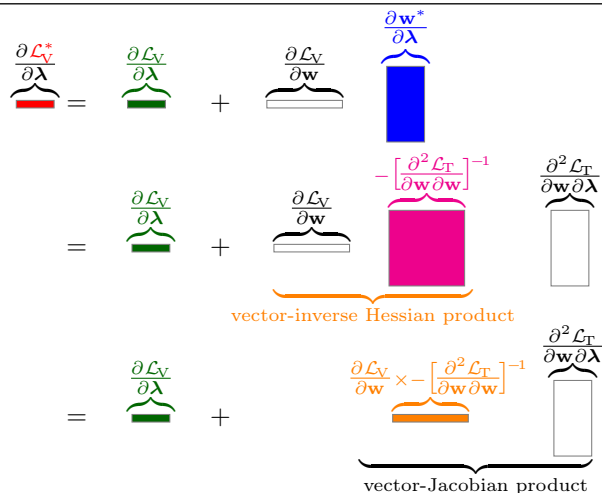
Figure 2: Hypergradient computation. The entire computation can be performed efficiently using vector-Jacobian products, provided a cheap approximation to the inverse-Hessian-vector product is available.

approximate the inverse with the Neumann series:

$$\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1} = \lim_{i \to \infty} \sum_{j=0}^{i} \left[I - \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}}\right]^j \quad (4)$$

In Section 4 we show that unrolling differentiation for i steps around locally optimal weights $\mathbf{w}^*$ is equivalent to approximating the inverse with the first i terms in the Neumann series. We then show how to do this approximation without instantiating any matrices by using efficient vector-Hessian products [Pearlmutter, 1994].

Algorithm 1 Gradient-based HO for $\boldsymbol{\lambda}^*, \mathbf{w}^*(\boldsymbol{\lambda}^*)$

1: Initialize hyperparameters $\boldsymbol{\lambda}'$ and weights $\mathbf{w}'$
2: **while** not converged **do**
3:      **for** $k = 1 \ldots N$ **do**
4:          $\mathbf{w}' -= \alpha \cdot \frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}', \mathbf{w}'}$
5:      $\boldsymbol{\lambda}' -= \texttt{hypergradient}(\mathcal{L}_V, \mathcal{L}_T, \boldsymbol{\lambda}', \mathbf{w}')$
6: **return** $\boldsymbol{\lambda}', \mathbf{w}'$ $\triangleright \boldsymbol{\lambda}^*, \mathbf{w}^*(\boldsymbol{\lambda}^*)$ from **Eq.1**

Algorithm 2 $\texttt{hypergradient}(\mathcal{L}_V, \mathcal{L}_T, \boldsymbol{\lambda}', \mathbf{w}')$

1: $\mathbf{v}_1 = \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}', \mathbf{w}'}$
2: $\mathbf{v}_2 = \texttt{approxInverseHVP}(\mathbf{v}_1, \frac{\partial \mathcal{L}_T}{\partial \mathbf{w}})$
3: $\mathbf{v}_3 = \texttt{grad}(\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}, \boldsymbol{\lambda}, \texttt{grad\_outputs} = \mathbf{v}_2)$
4: **return** $\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}}|_{\boldsymbol{\lambda}', \mathbf{w}'} - \mathbf{v}_3$ $\triangleright$ Return to **Alg. 1**

2.1 Proposed Algorithms

We outline our method in Algs. 1, 2, and 3, where α denotes the learning rate. Alg. 3 is also shown in Liao et al. [2018], and a special case of algorithms from Christianson [1998]. We visualize the hypergradient computation in Fig. 2.

Algorithm 3 $\texttt{approxInverseHVP}(\mathbf{v}, \mathbf{f})$: Neumann approximation of inverse-Hessian-vector product $\mathbf{v}[\frac{\partial \mathbf{f}}{\partial \mathbf{w}}]^{-1}$

1: Initialize sum $\mathbf{p} = \mathbf{v}$
2: **for** $j = 1 \ldots i$ **do**
3:      $\mathbf{v} -= \alpha \cdot \texttt{grad}(\mathbf{f}, \mathbf{w}, \texttt{grad\_outputs} = \mathbf{v})$
4:      $\mathbf{p} += \mathbf{v}$
5: **return** $\alpha \mathbf{p}$ $\triangleright$ Return to **Alg. 2**.

---

3 Related Work

Implicit Function Theorem. The IFT has been used for optimization in nested optimization problems [Ochs et al., 2015, Anonymous, 2019, Lee et al., 2019], backpropagating through arbitrarily long RNNs [Liao et al., 2018], k-fold cross-validation [Beirami et al., 2017], and influence functions [Koh and Liang, 2017]. Early work applied the IFT to regularization by explicitly computing the Hessian (or Gauss-Newton) inverse [Larsen et al., 1996, Bengio, 2000]. In Luketina et al. [2016], the identity matrix approximates the IFT's inverse Hessian. HOAG [Pedregosa, 2016] uses conjugate gradient (CG) to invert the Hessian approximately and provides convergence results given tolerances on the optimal parameter and inverse. In iMAML [Rajeswaran et al., 2019], a center to the weights is fit to perform on multiple tasks, where we fit to perform on the validation loss. In DEQ [Bai et al., 2019], implicit differentiation is used to add differentiable fixed-point methods into NN architectures. We use a Neumann approximation for the inverse-Hessian, instead of CG [Pedregosa, 2016, Rajeswaran et al., 2019] or the identity.

Approximate inversion algorithms. CG is difficult to scale to modern, deep NNs. We use a Neumann inverse approximation, which is a more stable alternative to CG in NNs [Liao et al., 2018, Shaban et al., 2019] and useful in stochastic settings [Agarwal et al., 2017]. The stability is motivated by connections between the Neumann series and unrolled differentiation [Shaban et al., 2019]. Alternatively, we could use prior knowledge about the NN structure to aid in the inversion—e.g., by using KFAC [Martens and Grosse, 2015]. It is possible to approximate the Hessian with the Gauss-Newton matrix or Fisher Information matrix [Larsen et al., 1996]. Various works use an identity approximation to the inverse, which is equivalent to 1-step unrolled differentiation [Luketina et al., 2016, Ren et al., 2018, Balaji et al., 2018, Liu et al., 2018, Finn et al., 2017, Shavitt and Segal, 2018, Nichol et al., 2018, Mescheder et al., 2017].

Unrolled differentiation for HO. A key difficulty in nested optimization is approximating how the optimized inner parameters (i.e., NN weights) change with respect to the outer parameters (i.e., hyperparameters).

Method	Steps	Eval.	Hypergradient Approximation		
Exact IFT	∞	$\mathbf{w}^*(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \times$		$\left[\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1} \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}} \bigg|_{\mathbf{w}^*(\boldsymbol{\lambda})}$
Unrolled Diff. [Maclaurin et al., 2015]	i	$\mathbf{w}_0$	$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \times$	$\sum_{j \le i} \left[\prod_{k<j} I - \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}}\Big|_{\mathbf{w}_{i-k}}\right]$	$\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\Big|_{\mathbf{w}_{i-j}}$
L-Step Truncated Unrolled Diff.	i	$\mathbf{w}_L$	$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \times$	$\sum_{L \le j \le i} \left[\prod_{k<j} I - \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}}\Big|_{\mathbf{w}_{i-k}}\right]$	$\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\Big|_{\mathbf{w}_{i-j}}$
Larsen et al. [1996]	∞	$\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \times$		$\left[\frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}} \frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}}^T\right]^{-1} \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}} \bigg|_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Bengio [2000]	∞	$\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \times$		$\left[\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1} \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}} \bigg|_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
$T1 - T2$ [27]	1	$\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \times$		$[I]^{-1} \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}} \bigg|_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Ours	i	$\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} - \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \times$		$\left(\sum_{j \le i} \left[I - \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}}\right]^j\right) \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}} \bigg|_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Conjugate Gradient (CG) $\approx$	-	$\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$	$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} -$		$\left(\arg\min_{\mathbf{x}} \|\mathbf{x} \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}} - \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}}\|\right) \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}} \bigg|_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Hypernetwork	-	-	$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} + \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \times \quad \frac{\partial \mathbf{w}_\phi^*}{\partial \boldsymbol{\lambda}}$ where $\mathbf{w}_\phi^*(\boldsymbol{\lambda}) = \arg\min_\phi \mathcal{L}_{\mathrm{T}}(\boldsymbol{\lambda}, \mathbf{w}_\phi(\boldsymbol{\lambda}))$		
Bayesian Optimization	-	-	$\frac{\partial \mathbb{E}[\mathcal{L}_{\mathrm{V}}^*]}{\partial \boldsymbol{\lambda}}$ where $\mathcal{L}_{\mathrm{V}}^* \sim \text{Gaussian-Process}(\{\boldsymbol{\lambda}_i, \mathcal{L}_{\mathrm{V}}(\boldsymbol{\lambda}_i, \mathbf{w}^*(\boldsymbol{\lambda}_i))\})$		

Table 1: An overview of methods to approximate hypergradients. Some methods can be viewed as using an approximate inverse in the IFT, or as differentiating through optimization around an evaluation point. Here, $\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$ is an approximation of the best-response at a fixed $\boldsymbol{\lambda}$, which is often found with gradient descent.

Method	Memory Cost
Diff. through Opt.	$\mathcal{O}(PI + H)$
Linear Hypernet	$\mathcal{O}(PH)$
Self-Tuning Nets (STN)	$\mathcal{O}((P + H)K)$
Neumann/CG IFT	$\mathcal{O}(P + H)$

Table 2: Gradient-based methods for HO. Differentiation through optimization scales with the number of unrolled iterations I; the STN scales with bottleneck size K, while our method only scales with the weight and hyperparameter sizes P and H.

We often optimize the inner parameters with gradient descent, so we can simply differentiate through this optimization. Differentiation through optimization has been applied to nested optimization problems by Domke [2012], was scaled to HO for NNs by Maclaurin et al. [2015], and has been applied to various applications like learning optimizers Andrychowicz et al. [2016]. Franceschi et al. [2018] provides convergence results for this class of algorithms, while Franceschi et al. [2017] discusses forward- and reverse-mode variants.

As the number of gradient steps we backpropagate through increases, so does the memory and computational cost. Often, gradient descent does not exactly minimize our objective after a finite number of steps— it only approaches a local minimum. Thus, to see how the hyperparameters affect the local minima, we may have to unroll the optimization infeasibly far. Unrolling a small number of steps can be crucial for performance but may induce bias Wu et al. [2018]. Shaban et al. [2019] discusses connections between unrolling and the

IFT, and proposes to unroll only the last L-steps. Dr-MAD Fu et al. [2016] proposes an interpolation scheme to save memory.

We compare hypergradient approximations in Table 1, and memory costs of gradient-based HO methods in Table 2. We survey gradient-free HO in Appendix B.

4 Method

In this section, we discuss how HO is a uniquely challenging nested optimization problem and how to combine the benefits of the IFT and unrolled differentiation.

4.1 Hyperparameter Opt. is Pure-Response

Eq. 3 shows that the hypergradient decomposes into a *direct* and *indirect gradient*. The bottleneck in hypergradient computation is usually finding the indirect gradient because we must take into account how the optimized parameters vary with respect to the hyperparameters. A simple optimization approach is to neglect the indirect gradient and only use the direct gradient. This can be useful in zero-sum games like GANs Goodfellow et al. [2014] because they always have a non-zero direct term.

However, using only the direct gradient does not work in general games Balduzzi et al. [2018]. In particular, it does not work for HO because the direct gradient is identically 0 when the hyperparameters $\boldsymbol{\lambda}$ can only influence the validation loss by changing the optimized

weights $\mathbf{w}^*(\boldsymbol{\lambda})$. For example, if we use regularization like weight decay when computing the training loss, but not the validation loss, then the direct gradient is always 0.

If the direct gradient is identically 0, we call the game *pure-response*. Pure-response games are uniquely difficult nested optimization problems for gradient-based optimization because we cannot use simple algorithms that rely on the direct gradient like simultaneous SGD. Thus, we must approximate the indirect gradient.

4.2 Unrolled Optimization and the IFT

Here, we discuss the relationship between the IFT and differentiation through optimization. Specifically, we (1) introduce the recurrence relation that arises when we unroll SGD optimization, (2) give a formula for the derivative of the recurrence, and (3) establish conditions for the recurrence to converge. Notably, we show that the fixed points of the recurrence recover the IFT solution. We use these results to motivate a computationally tractable approximation scheme to the IFT solution. We give proofs of all results in Appendix D.

Unrolling SGD optimization—given an initialization $\mathbf{w}_0$—gives us the recurrence:

$$\mathbf{w}_{i+1}(\boldsymbol{\lambda}) = T(\boldsymbol{\lambda}, \mathbf{w}_i) = \mathbf{w}_i(\boldsymbol{\lambda}) - \alpha \frac{\partial \mathcal{L}_{\mathrm{T}}(\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda}))}{\partial \mathbf{w}} \qquad (5)$$

In our exposition, assume that $\alpha = 1$. We provide a formula for the derivative of the recurrence, to show that it converges to the IFT under some conditions.

Lemma. *Given the recurrence from unrolling SGD optimization in Eq. 5, we have:*

$$\frac{\partial \mathbf{w}_{i+1}}{\partial \boldsymbol{\lambda}} = -\sum_{j \leq i} \left[\prod_{k < j} I - \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}} \Big|_{\boldsymbol{\lambda}, \mathbf{w}_{i-k}(\boldsymbol{\lambda})} \right] \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}} \Big|_{\boldsymbol{\lambda}, \mathbf{w}_{i-j}(\boldsymbol{\lambda})}$$
(6)

This recurrence converges to a fixed point if the transition Jacobian $\frac{\partial T}{\partial \mathbf{w}}$ is contractive, by the Banach Fixed-Point Theorem [Banach, 1922]. Theorem 2 shows that the recurrence converges to the IFT if we start at locally optimal weights $\mathbf{w}_0 = \mathbf{w}^*(\boldsymbol{\lambda})$, and the transition Jacobian $\frac{\partial T}{\partial \mathbf{w}}$ is contractive. We leverage that if an operator U is contractive, then the Neumann series $\sum_{i=0}^{\infty} U^k = (\mathrm{Id} - U)^{-1}$.

Theorem 2 (Neumann-SGD). *Given the recurrence from unrolling SGD optimization in (5), if $\mathbf{w}_0 = \mathbf{w}^*(\boldsymbol{\lambda})$:*

$$\frac{\partial \mathbf{w}_{i+1}}{\partial \boldsymbol{\lambda}} = - \left(\sum_{j \leq i} \left[I - \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}} \right]^j \right) \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}} \Bigg|_{\mathbf{w}^*(\boldsymbol{\lambda})} \qquad (7)$$

and if $I + \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}}$ is contractive:

$$\lim_{i \to \infty} \frac{\partial \mathbf{w}_{i+1}}{\partial \boldsymbol{\lambda}} = - \left[\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}} \right]^{-1} \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}} \Bigg|_{\mathbf{w}^*(\boldsymbol{\lambda})} \qquad (8)$$

This result is also shown in Shaban et al. [2019], but they use a different approximation for computing the hypergradient—see Table 1. Instead, we use the following best-response Jacobian approximation, where i controls the trade-off between computation and error bounds:

$$\frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}} \approx - \left(\alpha \sum_{j \leq i} \left[I - \alpha \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}} \right]^j \right) \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}} \Bigg|_{\mathbf{w}^*(\boldsymbol{\lambda})} \qquad (9)$$

Shaban et al. [2019] use an approximation that scales memory linearly in i, while ours is constant. We save memory because we reuse last $\mathbf{w}$ i times, while [Shaban et al., 2019] needs the last i $\mathbf{w}$'s. Scaling the Hessian by the learning rate α is key for convergence. Our algorithm has the following main advantages relative to other approaches:

- It requires a constant amount of memory, unlike other unrolled differentiation methods Maclaurin et al. [2015], Shaban et al. [2019].

- It is more stable than conjugate gradient, like unrolled differentiation methods Liao et al. [2018], Shaban et al. [2019].

4.3 Scope and Limitations

The assumptions necessary to apply the IFT are as follows: (1) $\mathcal{L}_V : \boldsymbol{\Lambda} \times \mathbf{W} \to \mathbb{R}$ is differentiable, (2) $\mathcal{L}_{\mathrm{T}} : \boldsymbol{\Lambda} \times \mathbf{W} \to \mathbb{R}$ is twice differentiable with an invertible Hessian at $\mathbf{w}^*(\boldsymbol{\lambda})$, and (3) $\mathbf{w}^* : \boldsymbol{\Lambda} \to \mathbf{W}$ is differentiable.

We need continuous hyperparameters to use gradient-based optimization, but many discrete hyperparameters (e.g., number of hidden units) have continuous relaxations [Maddison et al., 2017, Jang et al., 2016]. Also, we can only optimize hyperparameters that change the loss manifold, so our approach is not straightforwardly applicable to optimizer hyperparameters.

To exactly compute hypergradients, we must find $(\boldsymbol{\lambda}', \mathbf{w}')$ s.t. $\frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}', \mathbf{w}'} = 0$, which we can only solve to a tolerance with an approximate solution denoted $\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$. Pedregosa [2016] shows results for error in $\mathbf{w}^*$ and the inversion.

5 Experiments

We first compare the properties of Neumann inverse approximations and conjugate gradient, with experiments similar to Liao et al. [2018], Maclaurin et al.

[2015], Shaban et al. [2019], Pedregosa [2016]. Then we demonstrate that our proposed approach can overfit the validation data with small training and validation sets. Finally, we apply our approach to high-dimensional HO tasks: (1) dataset distillation; (2) learning a data augmentation network; and (3) tuning regularization parameters for an LSTM language model.

HO algorithms that are not based on implicit differentiation or differentiation through optimization—such as [Jaderberg et al., 2017, Jamieson and Talwalkar, 2016, Bergstra and Bengio, 2012, Kumar et al., 2018, Li et al., 2017, Snoek et al., 2012]—do not scale to the high-dimensional hyperparameters we use. Thus, we cannot sensibly compare to them for high-dimensional problems.

5.1 Approximate Inversion Algorithms

In Fig. 3 we investigate how close various approximations are to the true inverse. We calculate the distance between the approximate hypergradient and the true hypergradient. We can only do this for small-scale problems because we need the exact inverse for the true hypergradient. Thus, we use a linear network on the Boston housing dataset [Harrison Jr and Rubinfeld, 1978], which makes finding the best-response $\mathbf{w}^*$ and inverse training Hessian feasible.

We measure the cosine similarity, which tells us how accurate the direction is, and the ℓ_2 (Euclidean) distance between the approximate and true hypergradients. The Neumann approximation does better than CG in cosine similarity if we take enough HO steps, while CG always does better for ℓ_2 distance.

In Fig. 4 we show the inverse Hessian for a fully-connected 1-layer NN on the Boston housing dataset. The true inverse has a dominant diagonal, motivating identity approximations, while using more Neumann terms yields structure closer to the true inverse.

5.2 Overfitting a Small Validation Set

In Fig. 5, we check the capacity of our HO algorithm to overfit the validation dataset. We use the same restricted dataset as in Franceschi et al. [2017, 2018] of 50 training and validation examples, which allows us to assess HO performance easily. We tune a separate weight decay hyperparameter for each NN parameter as in Balaji et al. [2018], Maclaurin et al. [2015]. We show the performance with a linear classifier, AlexNet Krizhevsky et al. [2012], and ResNet44 He et al. [2016]. For AlexNet, this yields more than 50 000 000 hyperparameters, so we can perfectly classify our validation data by optimizing the hyperparameters.

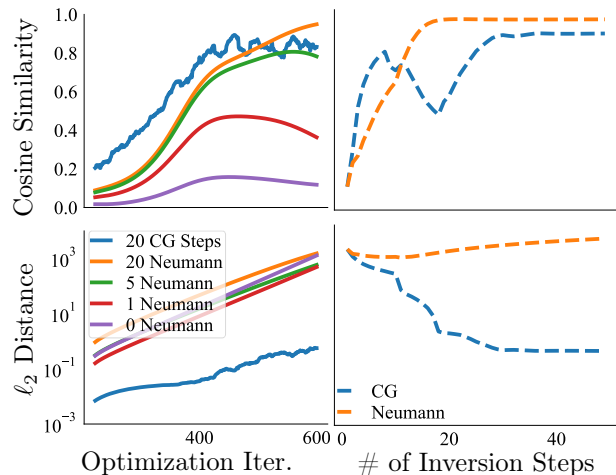Algorithm 1 achieves 100 % accuracy on the training

Figure 3: Comparing approximate hypergradients for inverse Hessian approximations to true hypergradients. The Neumann scheme often has greater cosine similarity than CG, but larger ℓ_2 distance for equal steps.
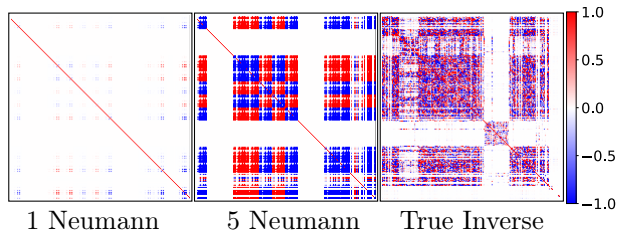

Figure 4: Inverse Hessian approximations preprocessed by applying `tanh` for a 1-layer, fully-connected NN on the Boston housing dataset as in Zhang et al. [2018].

and validation sets with significantly lower accuracy on the test set (Appendix E, Fig. 8), showing that we have a powerful HO algorithm. The same optimizer is used for weights and hyperparameters in all cases.
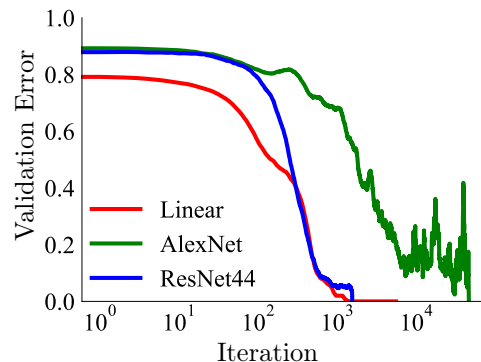

Figure 5: Algorithm 1 can overfit a small validation set on CIFAR-10. It optimizes for loss and achieves 100 % validation accuracy for standard, large models.

5.3 Dataset Distillation

Dataset distillation Maclaurin et al. [2015], Wang et al. [2018] aims to learn a small, synthetic training dataset from scratch, that condenses the knowledge contained

in the original full-sized training set. The goal is that a model trained on the synthetic data generalizes to the original validation and test sets. Distillation is an interesting benchmark for HO as it allows us to introduce tens of thousands of hyperparameters, and visually inspect what is learned: here, every pixel value in each synthetic training example is a hyperparameter. We distill MNIST and CIFAR-10/100 Krizhevsky [2009], yielding $28{\times}28{\times}10 = 7840$, $32{\times}32{\times}3{\times}10 = 30\,720$, and $32{\times}32{\times}3{\times}100 = 300\,720$ hyperparameters, respectively. For these experiments, all labeled data are in our validation set, while our distilled data are in the training set. We visualize the distilled images for each class in Fig. 9, recovering recognizable digits for MNIST and reasonable color averages for CIFAR-10/100.

Inverse Approx.	Validation	Test
0	92.5 ±0.021	92.6 ±0.017
3 Neumann	**95.1** ±0.002	94.6 ±0.001
3 Unrolled Diff.	95.0 ±0.002	**94.7** ±0.001
I	94.6 ±0.002	94.1 ±0.002

Table 3: Accuracy of different inverse approximations. Using 0 means that no HO occurs, and the augmentation is initially the identity. The Neumann approach performs similarly to unrolled differentiation [Maclaurin et al., 2015, Shaban et al., 2019] with equal steps and less memory. Using more terms does better than the identity, and the identity performed better than CG (not shown), which was unstable.

5.4 Learned Data Augmentation

Data augmentation is a simple way to introduce invariances to a model—such as scale or contrast invariance—that improve generalization Cubuk et al. [2018], Xie et al. [2019]. Taking advantage of the ability to optimize many hyperparameters, we learn data augmentation from scratch (Fig. 11).

Specifically, we learn a data augmentation network $\tilde{\mathbf{x}} = \mathbf{f}_{\boldsymbol{\lambda}}(\mathbf{x}, \epsilon)$ that takes a training example $\mathbf{x}$ and noise $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, and outputs an augmented example $\tilde{\mathbf{x}}$. The noise ϵ allows us to learn stochastic augmentations. We parameterize $\mathbf{f}$ as a U-net [Ronneberger et al., 2015] with a residual connection from the input to the output, to make it easy to learn the identity mapping. The parameters of the U-net, $\boldsymbol{\lambda}$, are hyperparameters tuned for the validation loss—thus, we have 6659 hyperparameters. We trained a ResNet18 He et al. [2016] on CIFAR-10 with augmented examples produced by the U-net (that is simultaneously trained on the validation set).

Results for the identity and Neumann inverse approximations are shown in Table 3. We omit CG because it performed no better than the identity. We found that using the data augmentation network improves validation and test accuracy by 2-3%, and yields smaller variance between multiple random restarts. In [Mounsaveng et al., 2019], a different augmentation network architecture is learned with adversarial training.

5.5 RNN Hyperparameter Optimization

We also used our proposed algorithm to tune regularization hyperparameters for an LSTM Hochreiter and Schmidhuber [1997] trained on the Penn TreeBank (PTB) corpus Marcus et al. [1993]. As in Gal and Ghahramani [2016], we used a 2-layer LSTM with 650 hidden units per layer and 650-dimensional word embeddings. Additional details are provided in Appendix E.4.

Overfitting Validation Data. We first verify that our algorithm can overfit the validation set in a small-data setting with 10 training and 10 validation sequences (Fig. 6). The LSTM architecture we use has $13\,280\,400$ weights, and we tune a separate weight decay hyperparameter per weight. We overfit the validation set, reaching nearly 0 validation loss.
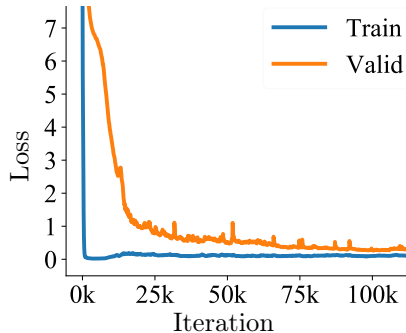

Figure 6: Alg. 1 can overfit a small validation set with an LSTM on PTB.

Large-Scale HO. There are various forms of regularization used for training RNNs, including variational dropout [Kingma et al., 2015] on the input, hidden state, and output; embedding dropout that sets rows of the embedding matrix to 0, removing tokens from all sequences in a mini-batch; DropConnect Wan et al.

[2013] on the hidden-to-hidden weights; and activation and temporal activation regularization. We tune these 7 hyperparameters simultaneously. Additionally, we experiment with tuning separate dropout/DropConnect rate for each activation/weight, giving $1\,691\,951$ total hyperparameters. To allow for gradient-based optimization of dropout rates, we use concrete dropout Gal et al. [2017].

Instead of using the small dropout initialization as in MacKay et al. [2019], we use a larger initialization of 0.5, which prevents early learning rate decay for our method. The results for our new initialization with no HO, our method tuning the same hyperparameters as MacKay et al. [2019] ("Ours"), and our method tuning many more hyperparameters ("Ours, Many") are shown in Table 4. We are able to tune hyperparameters more quickly and achieve better perplexities than the alternatives.

Method	Validation	Test	Time(s)
Grid Search	97.32	94.58	100k
Random Search	84.81	81.46	100k
Bayesian Opt.	72.13	69.29	100k
STN	70.30	67.68	25k
No HO	75.72	71.91	18.5k
Ours	69.22	66.40	18.5k
Ours, Many	**68.18**	**66.14**	18.5k

Table 4: Comparing HO methods for LSTM training on PTB. We tune millions of hyperparameters faster and with comparable memory to competitors tuning a handful. Our method competitively optimizes the same 7 hyperparameters as baselines from MacKay et al. [2019] (first four rows). We show a performance boost by tuning millions of hyperparameters, introduced with per-unit/weight dropout and DropConnect. "No HO" shows how the hyperparameter initialization affects training.

5.6 Effects of Many Hyperparameters

Given the ability to tune high-dimensional hyperparameters and the potential risk of overfitting to the validation set, should we reconsider how our training and validation splits are structured? Do the same heuristics apply as for low-dimensional hyperparameters (e.g., use $\sim 10\%$ of the data for validation)?

In Fig. 7 we see how splitting our data into training and validation sets of different ratios affects test performance. We show the results of jointly optimizing the NN weights and hyperparameters, as well as the results of fixing the final optimized hyperparameters and re-training the NN weights from scratch, which is a common technique for boosting performance [Goodfellow et al., 2016].

We evaluate a high-dimensional regime with a separate weight decay hyperparameter per NN parameter, and a low-dimensional regime with a single, global weight decay. We observe that: (1) for a single weight decay, the optimal combination of validation data and weight decay has similar test performance with and without re-training, because the optimal amount of validation data is small; and (2) for many weight decay, the optimal combination of validation data and weight decay is significantly affected by re-training, because the optimal amount of validation data needs to be large to fit our hyperparameters effectively.

For few hyperparameters, our results agree with the standard practice of using 10% of the data for validation and the other 90% for training. For many hyperparameters, our example shows we may need to use larger validation partitions for HO. If we use a large validation partition to fit the hyperparameters, it can be critical to re-train our model with all of the data.
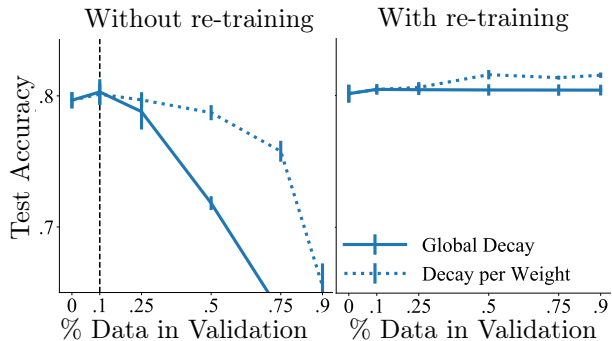

Figure 7: Test accuracy of logistic regression on MNIST, with different size validation splits. Solid lines correspond to a single global weight decay (1 hyperparameter), while dotted lines correspond to a separate weight decay per weight (many hyperparameters). The best validation proportion for test performance is different after re-training for many hyperparameters, but similar for few hyperparameters.

6 Conclusion

We present a gradient-based hyperparameter optimization algorithm that scales to high-dimensional hyperparameters for modern, deep NNs. We use the implicit function theorem to formulate the hypergradient as a matrix equation, whose bottleneck is inverting the Hessian of the training loss with respect to the NN parameters. We scale the hypergradient computation to large NNs by approximately inverting the Hessian, leveraging a relationship with unrolled differentiation.

We believe algorithms of this nature provide a path for practical nested optimization, where we have Hessians with known structure. Examples of this include GANs Goodfellow et al. [2014], and other multi-agent games Foerster et al. [2018], Letcher et al. [2018].

Acknowledgements

References

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8):1889–1900, 2000.

Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326, 2012.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, pages 1165–1173, 2017.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1563–1572, 2018.

Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 1723–1732, 2019.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135, 2017.

Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. *arXiv preprint arXiv:1909.04630*, 2019.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019.

Akshay Mehra and Jihun Hamm. Penalty method for inversion-free deep bilevel optimization. *arXiv preprint arXiv:1911.03432*, 2019.

Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *International Conference on Machine Learning*, pages 4331–4340, 2018.

Tae-Hoon Kim and Jonghyun Choi. ScreenerNet: Learning self-paced curriculum for deep neural networks. *arXiv preprint arXiv:1801.00904*, 2018.

Jiong Zhang, Hsiang-fu Yu, and Inderjit Dhillon. AutoAssist: A framework to accelerate training of deep neural networks. *arXiv preprint arXiv:1905.03381*, 2019.

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Unsupervised data augmentation. *arXiv preprint arXiv:1904.12848*, 2019.

Jonas Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404, 1975.

Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R Collins, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Tuning hyperparameters without grad students: Scalable and robust Bayesian optimisation with Dragonfly. *arXiv preprint arXiv:1903.06694*, 2019.

Renjie Liao, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, KiJung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pages 3088–3097, 2018.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Bruce Christianson. Reverse aumulation and imploicit functions. *Optimization Methods and Software*, 9(4): 307–322, 1998.

Jan Larsen, Lars Kai Hansen, Claus Svarer, and M Ohlsson. 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*, pages 62–71, 1996.

Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *International Conference on Machine Learning*, pages 2952–2960, 2016.

Peter Ochs, René Ranftl, Thomas Brox, and Thomas Pock. Bilevel optimization with nonsmooth lower level problems. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 654–665, 2015.

Anonymous. On solving minimax optimization locally: A follow-the-ridge approach. *International Conference on Learning Representations*, 2019. URL https://openreview.net/pdf?id=Hkx7_1rKwS.

Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

Ahmad Beirami, Meisam Razaviyayn, Shahin Shahrampour, and Vahid Tarokh. On optimal generalizability in parametric learning. In *Advances in Neural Information Processing Systems*, pages 3455–3465, 2017.

Pang Wei Koh and Percy Liang. Understanding blackbox predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1885–1894. JMLR. org, 2017.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning*, pages 737–746, 2016.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research*, 18(1):4148–4187, 2017.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pages 2408–2417, 2015.

Yogesh Balaji, Swami Sankaranarayanan, and Rama Chellappa. Metareg: Towards domain generalization using meta-regularization. In *Advances in Neural Information Processing Systems*, pages 998–1008, 2018.

Ira Shavitt and Eran Segal. Regularization learning networks: Deep learning for tabular datasets. In *Advances in Neural Information Processing Systems*, pages 1379–1389, 2018.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of gans. In *Advances in Neural Information Processing Systems*, pages 1825–1835, 2017.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

Jie Fu, Hongyin Luo, Jiashi Feng, Kian Hsiang Low, and Tat-Seng Chua. DrMAD: Distilling reverse-mode automatic differentiation for optimizing hyperparameters of deep neural networks. In *International Joint Conference on Artificial Intelligence*, pages 1469–1475, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. In *International Conference on Machine Learning*, pages 363–372, 2018.

Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta Mathematicae*, 3:133–181, 1922.

C Maddison, A Mnih, and Y Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pages 240–248, 2016.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

Manoj Kumar, George E Dahl, Vijay Vasudevan, and Mohammad Norouzi. Parallel architecture and hyperparameter search via successive halving and classification. *arXiv preprint arXiv:1805.10255*, 2018.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: a novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

David Harrison Jr and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1): 81–102, 1978.

Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. In *International Conference on Machine Learning*, pages 5847–5856, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

Saypraseuth Mounsaveng, David Vazquez, Ismail Ben Ayed, and Marco Pedersoli. Adversarial learning of general transformations for data augmentation. *International Conference on Learning Representations*, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1027–1035, 2016.

Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using Dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590, 2017.

Matthew MacKay, Paul Vicol, Jon Lorraine, David Duvenaud, and Roger Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *International Conference on Learning Representations*, 2019.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 122–130, 2018.

Alistair Letcher, Jakob Foerster, David Balduzzi, Tim Rocktäschel, and Shimon Whiteson. Stable opponent shaping in differentiable games. *arXiv preprint arXiv:1811.08469*, 2018.

Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.

Andrew Brock, Theodore Lim, James Millar Ritchie, and Nicholas Weston. SMASH: One-Shot Model Architecture Search through HyperNetworks. In *International Conference on Learning Representations*, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning. Lecture 6a. Overview of mini-batch gradient descent. 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 (11):2278–2324, 1998.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. *International Conference on Learning Representations*, 2018.