NON-GREEDY GRADIENT-BASED HYPERPARAMETER OPTIMIZATION OVER LONG HORIZONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Gradient-based meta-learning has earned a widespread popularity in few-shot learning, but remains broadly impractical for tasks with long horizons (many gradient steps), due to memory scaling and gradient degradation issues. A common workaround is to learn meta-parameters online, but this introduces greediness which comes with a significant performance drop. In this work, we enable non-greedy meta-learning of hyperparameters over long horizons by sharing hyperparameters that are contiguous in time, and using the sign of hypergradients rather than their magnitude to indicate convergence. We implement this with forward-mode differentiation, which we extend to the popular momentum-based SGD optimizer. We demonstrate that the hyperparameters of this optimizer can be learned non-greedily without gradient degradation over $\sim 10^4$ inner gradient steps, by only requiring ~ 10 outer gradient steps. On CIFAR-10, we outperform greedy and random search methods for the same computational budget by nearly 10%. Code will be available upon publication.

1 INTRODUCTION

Deep neural networks have shown tremendous success on a wide range of applications, including classification (He et al., 2016), generative models (Brock et al., 2019), natural language processing (Devlin et al., 2018) and speech recognition (Oord et al., 2016).This success is in part due to effective optimizers such as SGD with momentum or Adam (Kingma & Ba, 2015), which require carefully tuned hyperparameters for each application. In recent years, a long list of heuristics to tune such hyperparameters has been compiled by the deep learning community, including things like: how to best decay the learning rate (Loshchilov & Hutter, 2017), how to scale hyperparameters with the budget available (Li et al., 2020), and how to scale learning rate with batch size (Goyal et al., 2017). Unfortunately these heuristics are often dataset specific and architecture dependent (Dong et al., 2020), and must constantly evolve to accommodate new optimizers (Loshchilov & Hutter, 2019), or new tools, like batch normalization which allows for larger learning rates and smaller weight decay (Ioffe & Szegedy, 2015).

With so many ways to choose hyperparameters, the deep learning community is at risk of adopting models based on how much effort went into tuning them, rather than their methodological insight. The field of hyperparameter optimization (HPO) aims to find hyperparameters that provide a good generalization performance automatically. Unfortunately, existing tools are rather unpopular for deep networks, largely owing to their inefficiency. Here we focus on gradient-based HPO, which calculates hypergradients, i.e. the gradient of some generalization loss with respect to each hyperparameter. Gradient-based HPO should be more efficient than black box methods as the dimensionality of the hyperparameter space increases, since it relies on gradients rather than trial and error. In practice however, learning hyperparameters with gradients has only been popular in few-shot learning tasks where the horizon is short. This is because long horizons cause hypergradient degradation, and incur a memory cost that makes reverse-mode differentiation prohibitive. Greedy alternatives alleviate both of these issues, but come at the cost of solving hyperparameters locally instead of globally. Forward-mode differentiation has been shown to offer a memory cost constant with horizon size, but it doesn't address gradient degradation and only scales to few hyperparameters, which has limited its use to the greedy setting as well.

To the best of our knowledge, this work demonstrates for the first time that gradient-based HPO can be applied for long horizon problems like CIFAR-10 without being greedy. Specifically, we make the following contributions: **(1)** we propose to share hyperparameters through time and show that this significantly reduces the variance of hypergradients, **(2)** we show that the sign of hypergradients is a better indicator of convergence than their magnitude and enables a small number of outer steps, **(3)** we combine the above in a forward-mode algorithm adapted to modern SGD optimizers with momentum and weight decay, and **(4)** we show that our method significantly outperforms random search and greedy alternatives when used with the same computational budget.

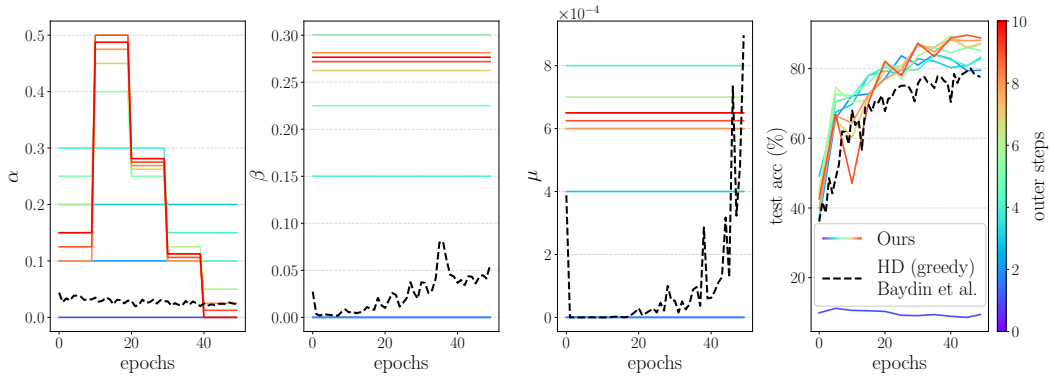

Figure 1: Our method applied to the SGD optimizer to learn (from left to right) the learning rate schedule $\boldsymbol{\alpha}$ the momentum $\boldsymbol{\beta}$ and weight decay $\boldsymbol{\mu}$ for a WRN-16-1 over 50 epochs of CIFAR-10. We use hyperparameter sharing over 10, 50 and 50 epochs respectively. All hyperparameters are initialized to zero and converge within just 10 outer steps to values that significantly outperform the online greedy alternative (Baydin et al., 2018), and match aggressively hand-tuned baselines for this setting (see Section 4.2).

2 Related Work

There are many ways to perform hyperparameter optimization (HPO), including Bayesian optimization (Snoek et al., 2015), reinforcement learning (Zoph & Le, 2017), evolutionary algorithms (Jaderberg et al., 2017) and gradient-based methods (Bengio, 2000). Here we focus on the latter, but a comparison of HPO methods can be found in (Feurer & Hutter, 2019). Modern work in meta-learning deals with various forms of gradient-based HPO, many examples of which are discussed in this survey (Hospedales et al., 2020). However, meta-learning typically focuses on the few-shot regime where horizons are conveniently short, while in this work we focus on long horizons.

Gradient-based HPO. Using the gradient of some validation loss with respect to the hyperparameters is typically the preferred choice when the underlying optimization is differentiable. This is a type of bilevel optimization (Franceschi et al., 2018) which stems from earlier work on backpropagation through time (Werbos, 1990) and real-time recurrent learning (Williams & Zipser, 1989). Unfortunately, differentiating optimization is an expensive procedure in both time and memory, and most proposed methods are limited to small models and toy datasets (Domke, 2012; Maclaurin et al., 2015; Pedregosa, 2016). Efforts to make the problem more tractable include optimization shortcuts (Fu et al., 2016), truncation (Shaban et al., 2019) and implicit gradients (Rajeswaran et al., 2019; Lorraine et al., 2019). Truncation can be combined with our approach but produces biased gradients (Metz et al., 2019), while implicit differentiation is only applicable to hyperparameters that define the training loss (e.g. augmentation) but not to hyperparameters that define how the training loss is minimized (e.g. optimizer hyperparameters). Forward-mode differentiation (Williams & Zipser, 1989) boasts a memory cost constant with horizon size, but it doesn't address gradient degradation which has restricted its use to the greedy setting (Franceschi et al., 2017).

Greedy Methods. One trick that prevents gradient degradation and significantly reduces compute and memory cost is to solve the bilevel optimization greedily. This has become the default trick in various long-horizon problems, including HPO over optimizers (Luketina et al., 2016; Franceschi

et al., 2017; Baydin et al., 2018; Donini et al., 2019), architecture search (Liu et al., 2019), or curriculum learning Ren et al. (2018). This involves splitting the inner optimization problem into smaller chunks (often just one batch), and solving for hyperparameters over these smaller horizons instead; often in an online fashion. In this paper we expand upon previous observations (Wu et al., 2018) and take the view that greediness fundamentally solves for the wrong objective. Instead, the focus of our paper is to extend forward-mode differentiation methods to the non-greedy setting.

Gradient Degradation. Gradient degradation of some scalar w.r.t a parameter is a broad issue that arises when that parameter influences the scalar in a chaotic fashion, such as through long chains of nonlinear mappings. This manifests itself in HPO as vanishing or exploding hypergradients, due to low or high curvature components of the validation loss surface respectively. This leads to hypergradients with a large variance, which we denote as *hypervariance* (more in Section 4.1), and which prevents long-horizon optimization. This is usually observed in the context of recurrent neural networks (Bengio et al., 1993; 1994), but also in reinforcement learning (Parmas et al., 2018) and HPO (Maclaurin et al., 2015). Solutions like LSTMs (Hochreiter & Schmidhuber, 1997) and gradient clipping (Pascanu et al., 2013) have been proposed, but are respectively inapplicable and insufficient to our problem. Variational optimization (Metz et al., 2019) and preconditioning warp-layers (Flennerhag et al., 2020) can mitigate gradient degradation, but these methods are expensive in memory and therefore are limited to small architectures and/or a few hundred inner steps, while we differentiate over $\sim 10^4$ inner steps for Wide ResNets (Zagoruyko & Komodakis, 2016).

3 METHODOLOGY

3.1 PROBLEM STATEMENT

Consider a neural network with weights $\boldsymbol{\theta}$, trained to minimize a loss $\mathcal{L}_{train}$ over a dataset $\mathcal{D}_{train}$, with a gradient-based optimizer $\Phi : \boldsymbol{\theta}_{t+1} = \Phi(\boldsymbol{\theta}_t(\boldsymbol{\lambda}_{[0:t-1]}), \boldsymbol{\lambda}_{[t]})$. Here $\boldsymbol{\lambda} \in \mathbb{R}^{K \times T}$ is a hyperparameter matrix, with K the number of hyperparameters used per step (e.g. 2 if learning momentum and learning rate), and T the total number of update steps. Throughout this paper we write column indices in brackets to differentiate them from a variable evaluated at time t.

We would like to find $\boldsymbol{\lambda}^*$ such that the minimizer of $\mathcal{L}_{train}$, namely $\boldsymbol{\theta}^* \simeq \boldsymbol{\theta}_T(\boldsymbol{\lambda})$, also minimizes some generalization loss $\mathcal{L}_{val}$ on a validation dataset $\mathcal{D}_{val}$. Bilevel optimization (Stackelberg, 1952) is the usual framework to express this problem:

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}} \mathcal{L}_{val}(\boldsymbol{\theta}_T(\boldsymbol{\lambda}), \mathcal{D}_{val}) \tag{1}$$

$$\text{subject to} \quad \boldsymbol{\theta}_T = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{train}(\boldsymbol{\theta}, \mathcal{D}_{train}) \tag{2}$$

$$\text{solved with} \quad \boldsymbol{\theta}_{t+1} = \Phi(\boldsymbol{\theta}_t(\boldsymbol{\lambda}_{[0:t-1]}), \boldsymbol{\lambda}_{[t]}) \tag{3}$$

The inner loop (lower level) in Eq 2 expresses a constraint on the outer loop (upper level) in Eq 1. In gradient-based HPO, our task is to compute the hypergradient $d\mathcal{L}_{val}/d\boldsymbol{\lambda}$ and update $\boldsymbol{\lambda}$ accordingly. The horizon H refers to the number of update steps taken in the inner loop (to optimize $\boldsymbol{\theta}$) before one step is taken in the outer loop (to optimize $\boldsymbol{\lambda}$), and when solving Eq 1 non-greedily we have $H = T$. However, most modern approaches are greedy: they rephrase the above problem into a sequence of several independent problems of smaller horizons, where $\boldsymbol{\lambda}^*_{[t:t+H]}$ is learned in the outer loop subject to an inner loop optimization from $\boldsymbol{\theta}_t$ to $\boldsymbol{\theta}_{t+H}$ with $H \ll T$. Despite its popularity, we argue against greediness in Section 3.3.

3.2 FORWARD-MODE DIFFERENTIATION OF MODERN OPTIMIZERS

In modern gradient-based HPO, reverse-mode differentiation is used in the inner optimization problem (Eq 2) to optimize $\boldsymbol{\theta}$. However, the memory cost of using it for the outer optimization (Eq 1) is $\mathcal{O}(FH)$ where F is the memory cost of one forward pass through the network (weights plus activations). This is extremely limiting: for large networks, only $H \sim 10$ could be solved with modern GPUs, while problems like CIFAR-10 require $H \sim 10^4$. Instead, we must use forward-mode differentiation, which scales in memory as $\mathcal{O}(DN)$, where D is the number weights and N is the number

of learnable hyperparameters. The additional scaling with N is a limitation in many applications, but sharing hyperparameters (Section 3.3) conveniently allows for smaller values of N.

In this section we give a background on forward-mode differentiation and extend it to the most popular optimizer, namely SGD with momentum and weight decay. To the best of our knowledge, previous work focuses on simpler versions of this optimizer, usually by removing momentum and weight decay, and only learns the learning rate, greedily. First, we use the chain rule to write:

$$\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}} = \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{\theta}_T} \frac{d\boldsymbol{\theta}_T}{d\boldsymbol{\lambda}} \tag{4}$$

The first term on the RHS is trivial and can be obtained with reverse-mode differentiation as usual. The second term is more problematic because $\boldsymbol{\theta}_T = \boldsymbol{\theta}_T(\boldsymbol{\theta}_{T-1}(\boldsymbol{\theta}_{T-2}(...), \boldsymbol{\lambda}_{[T-2]}), \boldsymbol{\lambda}_{[T-1]})$. We use the chain rule again to calculate this term recursively:

$$\frac{d\boldsymbol{\theta}_t}{d\boldsymbol{\lambda}} = \left.\frac{\partial \boldsymbol{\theta}_t}{\partial \boldsymbol{\theta}_{t-1}}\right|_{\boldsymbol{\lambda}} \frac{d\boldsymbol{\theta}_{t-1}}{d\boldsymbol{\lambda}} + \left.\frac{\partial \boldsymbol{\theta}_t}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\theta}_{t-1}} \quad \text{which we write as} \quad \boldsymbol{Z}_t = \boldsymbol{A}_t \boldsymbol{Z}_{t-1} + \boldsymbol{B}_t \tag{5}$$

where $\boldsymbol{Z}_t \in \mathbb{R}^{D \times N}$, $\boldsymbol{A}_t \in \mathbb{R}^{D \times D}$ and $\boldsymbol{B}_t \in \mathbb{R}^{D \times N}$. We use Pytorch's update rule for SGD (Paszke et al., 2019), with learning rate α_t, momentum β_t, weight decay μ_t and velocity $\boldsymbol{v}_t$:

$$\boldsymbol{\theta}_t = \Phi(\boldsymbol{\theta}_{t-1}) = \boldsymbol{\theta}_{t-1} - \alpha \boldsymbol{v}_t \quad \text{where} \quad \boldsymbol{v}_t = \beta_t \boldsymbol{v}_{t-1} + (\partial \mathcal{L}_{train} / \partial \boldsymbol{\theta}_{t-1}) + \mu_t \boldsymbol{\theta}_{t-1} \tag{6}$$

Now let us consider the case where we learn the learning rate schedule, namely $\boldsymbol{\lambda} = \boldsymbol{\alpha}$. If we use the update rule without momentum (Donini et al., 2019), $\boldsymbol{B}_t$ is conveniently sparse: it is a $D \times N$ matrix that only has one non-zero column corresponding to the hyperparameters used at step t, i.e. α_t. However we introduce momentum and therefore the velocity depends on the hyperparameters of previous steps. A further recursive term $\boldsymbol{C}_t = (\partial \boldsymbol{v}_t / \partial \boldsymbol{\lambda})$ must now be considered to get exact hypergradients. Putting it together (see Appendix A) we obtain:

$$\begin{cases} \boldsymbol{A}_t^{\boldsymbol{\alpha}} = \mathbb{1}^{D \times D} - \alpha_t \left(\dfrac{\partial^2 \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}^2} + \mu_t \mathbb{1}^{D \times D} \right) \\ \boldsymbol{B}_t^{\boldsymbol{\alpha}} = -\beta_t \alpha_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} - \delta_t^{D \times N} \left(\beta_t \boldsymbol{v}_{t-1} + \dfrac{\partial \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}} + \mu_t \boldsymbol{\theta}_{t-1} \right) \\ \boldsymbol{C}_t^{\boldsymbol{\alpha}} = \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} + \left(\mu_t \mathbb{1}^{D \times D} + \dfrac{\partial^2 \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}^2} \right) \boldsymbol{Z}_{t-1}^{\boldsymbol{\alpha}} \end{cases} \tag{7}$$

where $\delta_t^{D \times N}(\boldsymbol{q})$ turns a vector $\boldsymbol{q}$ of size D into a zero matrix of size $D \times N$ where the column corresponding to the hyperparameter used at step t is set to $\boldsymbol{q}$. A similar technique can be applied to momentum and weight decay to get $\boldsymbol{Z}_T^{\boldsymbol{\beta}}$ and $\boldsymbol{Z}_T^{\boldsymbol{\mu}}$ (Appendix A). The first-order approximation of the above system corresponds to setting the Hessian to zero, but in practice we find it too crude of an approximation for large horizons. All hypergradient derivations in this paper were thoroughly checked with finite differences.

3.3 Enabling non-greediness

Greedy approaches (Luketina et al., 2016; Franceschi et al., 2017; Baydin et al., 2018; Donini et al., 2019) have many advantages: they mitigate gradient degradation issues, lower training time, and require less memory in reverse-mode differentiation. However, these methods look for $\boldsymbol{\lambda}^*$ that does well locally rather than globally, i.e. they constrain $\boldsymbol{\lambda}^*$ to a subspace of solutions such that $\boldsymbol{\theta}_H^*, \boldsymbol{\theta}_{2H}^*, ..., \boldsymbol{\theta}_T^*$ all yield good validation performances. In the same way that running a marathon by optimizing for a sequence of 100m sprints is sub optimal, greediness is a poor proxy to minimize $\mathcal{L}_{val}(\boldsymbol{\theta}_T)$. In our experiments, we found that getting competitive hyperparameters with greediness often revolves around tricks such as online learning with a very low outer learning rate combined with hand-tuned initial hyperparameter values, to manually prevent convergence to small values. But solving the greedy objective correctly leads to poor solutions, a special case of which was previously

Algorithm 1: Non-greedy learning of the learning rate schedule $\boldsymbol{\alpha}$, using forward-mode differentiation with hyperparameter sharing and a sign-based outer optimizer.

for *o in* $1, 2, ...$ **do**
  initialize: $\mathcal{D}_{train}, \mathcal{D}_{val}, \boldsymbol{\theta} \in \mathbb{R}^D, \boldsymbol{\alpha} = \mathbf{0}^{N_\alpha}, Z^{\boldsymbol{\alpha}} = \mathbf{0}^{D \times N_\alpha}, C^{\boldsymbol{\alpha}} = \mathbf{0}^{D \times N_\alpha}, \boldsymbol{\gamma} \in \mathbb{R}^{N_\alpha}$
  for *i in* $1, 2, ..., H$ **do**
    $\boldsymbol{x}_{train}, \boldsymbol{y}_{train} \sim \mathcal{D}_{train}$
    $\boldsymbol{g}_{train} = \partial \mathcal{L}_{train}(\boldsymbol{x}_{train}, \boldsymbol{y}_{train}) / \partial \boldsymbol{\theta}$
    if *truncate==False* **then**
      $\mathcal{H} Z^{\boldsymbol{\alpha}} = \partial(\boldsymbol{g}_{train} Z^{\boldsymbol{\alpha}}) / \partial \boldsymbol{\theta}$
      $Z^{\boldsymbol{\alpha}} = A^{\boldsymbol{\alpha}} Z^{\boldsymbol{\alpha}} + B^{\boldsymbol{\alpha}}$ (Eq 7)
    update $\boldsymbol{\theta}$ and $\boldsymbol{v}$ (Eq 6)
  $\boldsymbol{g}_{val} = \partial \mathcal{L}_{val}(\mathcal{D}_{val}) / \partial \boldsymbol{\theta}$
  $\boldsymbol{s}_o = \mathrm{sgn}(\boldsymbol{g}_{val} Z^{\boldsymbol{\alpha}})$ `// hypergradient signs`
  for *n in* $1, 2, ..., N_\alpha$ **do**
    if $\boldsymbol{s}_{o,[n]} \neq \boldsymbol{s}_{o-1,[n]}$ **then**
      $\boldsymbol{\gamma}_{[n]} \leftarrow \boldsymbol{\gamma}_{[n]} / 2$
  $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \boldsymbol{s}_o \odot \boldsymbol{\gamma}$

described as the "short-horizon bias" (Wu et al., 2018) when learning the learning rate. Finally, we found that most of the greedy literature uses the test set as the validation set, which creates a risk of meta-overfitting to the test set. Throughout this work, we carve out a validation set from our training set instead.

The main challenge of doing non-greedy bilevel optimization over long horizons is gradient degradation. In HPO this arises because a small change in the state of the network at inner step t can cascade forward into a completely different weight trajectory given enough subsequent steps, yielding very different hypergradients. We refer to this phenomenon as *hypervariance*, and quantify it in Section 4.1.

Hyperparameter Sharing. One way to iron out this hypervariance would be to take many outer steps on the hyperparameters, with previous work suggesting that a few thousands would be required (Wu et al., 2018). But since each outer step requires computing the entire inner problem, this is intractable. Instead, we propose to use the long horizon to our advantage and *average out hypergradients through inner steps rather than outer steps*. This is equivalent to hyperparameter sharing: learning one hyperparameter for several contiguous steps in the inner loop. Assuming that contiguous steps have hypergradients drawn from the same distribution, the average hypergradient is more stable, since it is less likely to reflect a single batch in the inner loop. The key to this solution being effective is that it gets better as degradation gets worse: the larger the horizon, the more noisy individual hypergradients become, but the more samples are included in the average hypergradient, and so the less noisy the average hypergradient becomes.

Convergence From Hypergradient Signs. While hyperparameter sharing produces stable hypergradients with a reasonable range (see Appendix C), reaching zero hypergradients and thus convergence remains difficult, and often requires a large number of outer steps. We propose to use the sign of hypergradients as an indicator of convergence instead of their magnitude. More specifically, we update hyperparameters with positive (negative) hypergradients by an amount $+\gamma$ ($-\gamma$), and γ is decayed by a factor of 2 every time the corresponding hypergradient changes sign across consecutive outer steps. This allows for convergence after hypergradients have changed signs a few times, and we find this much more robust than using modern optimizers to update $\boldsymbol{\lambda}$ like Adam. This has the

added benefit of letting the user define the range of hyperparameter search more explicitly, rather than having it implicitly defined by the magnitude of hypergradients.

Putting it all together, Algorithm 1 shows how to learn a schedule of $N^{\boldsymbol{\alpha}}$ learning rates efficiently. Hyperparameter sharing is implemented by reusing the same $Z^{\boldsymbol{\alpha}}$ columns for several consecutive inner steps. The most expensive part per outer step is to compute the Hessian matrix product $\mathcal{H}Z^{\boldsymbol{\alpha}}$ at each inner step i. We can adapt the work of (Shaban et al., 2019) to forward mode differentiation by truncating alternating steps. We could also use functional forms for more complex schedules to be learned in terms of fewer hyperparameters, which would cheapen the calculation of $\mathcal{H}Z^{\boldsymbol{\alpha}}$. However, this typically results in including a stronger inductive bias about the general shape of each hyperparameter schedule, which can easily cloud the true performance of HPO algorithms. In practice, we use a similar form to Algorithm 1 to learn the learning rate, momentum and weight decay jointly.

4 Experiments

In Section 4.1 we quantify gradient degradation and show that non-greediness increases it while hyperparameter sharing significantly reduces it. In Section 4.2 we show that our method allows for stable non-greedy HPO on 3 datasets, and outperforms several other methods when used with the same computational budget. Implementation details for each of our experiments can be found in Appendix B.

4.1 Reducing Gradient Degradation

Gradient degradation in the context of HPO over long horizons isn't well understood. In this section we propose a simple metric to quantify this degradation (hypervariance), and we propose a method to reduce it (hyperparameter sharing). We also consider the fluctuation of the sign of the hypergradients to motivate the use of the outer optimizer described in Section 3.3.

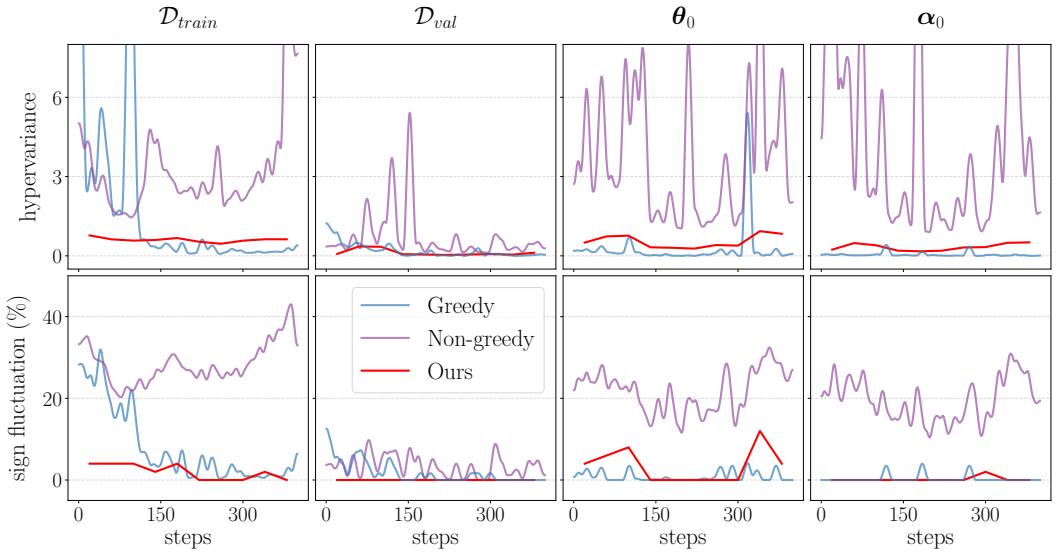

Figure 2: The hypervariance (top row) and sign fluctuation (bottom row), when learning several learning rates on the SVHN dataset. These are calculated as we perturb, from left to right, the choice of training data, the choice of validation data, the initial weights and the initial learning rates. We observe that non-greediness over long horizons is responsible for a large hypervariance and sign fluctuation (purple vs blue). However, we can preserve non-greediness and greatly reduce gradient degradation by sharing hyperparameters for each 40 contiguous inner steps (red).

Hypervariance Definition. Consider the hypergradients of the learning rates $\boldsymbol{\alpha}$. These are affected by four main factors: the training data, the validation data, the weights' initialization $\boldsymbol{\theta}_0$ and

the current learning rate schedule $\boldsymbol{\alpha}_0$. We can slightly perturb each factor individually P times while keeping all others constant. This gives us a distribution of hypergradients around a fixed point for each hyperparameter, which should have low variance in order for the outer optimization to be stable. We define the hypervariance to be a (unitless) inverse signal to noise ratio, namely the ratio of the standard deviation to the mean of hypergradients, computed for each hyperparameter. This makes the norm of the hypergradients irrelevant, which is desired since it can always be scaled as needed with some outer learning rate in practice.

Hypergradient Sign Fluctuation. In Section 3.3 we made the case for using the sign of hypergradients alone. Since hypervariance doesn't capture the fluctuation in the sign of hypergradients, we also measure this quantity by looking at the smallest quantity between P_+ and P_-, the number of positive and negative signs, expressed as a ratio: sign fluctuation $= \min(P_-/P, P_+/P)$. This quantity is capped at 50% which corresponds to hypergradients maximally uninformative regarding whether or not some hyperparameter should increase or decrease.

The variance and sign fluctuation of hypergradients are shown in Figure 2 when hypergradients are calculated for several learning rates, for a LeNet on the SVHN dataset. We observe that non-greediness makes hypergradients much more sensitive to small changes in the optimization landscape. For instance, changing the seed for the sampling of the training dataset produces hypergradients with a standard deviation that is over 6 times larger than the mean, making outer optimization very inefficient. However, sharing contiguous hyperparameters lowers the hypervariance and sign fluctuation drastically. In particular, the sign of hypergradients only fluctuates by an average of 1.5% across all factors of variation, making it a robust quantity for outer optimization.

4.2 Non-greedy HPO Over Long Horizons

The section above considered hypergradient noise around a fixed hyperparameter setting. In this section, we consider how that noise and its mitigation translate to the outer optimization process, where hyperparameters are updated at each outer step.

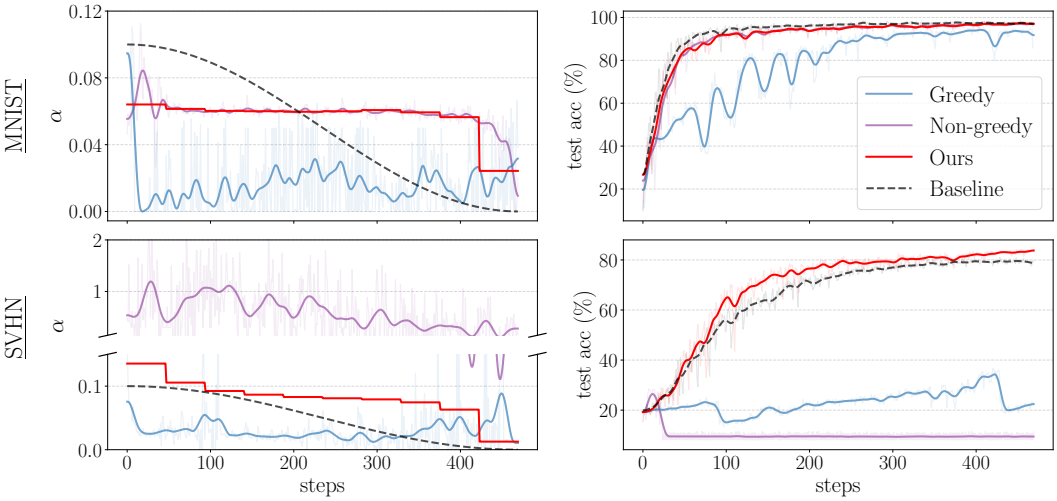

Figure 3: The learning rate schedule $\boldsymbol{\alpha}$ learned for the MNIST and SVHN datasets using a LeNet architecture over 500 inner gradient steps. We observe that on real-world datasets like SVHN, both greedy and non-greedy hyperparameter optimizations fail to learn decent learning rate schedules. However, sharing learning rates within neighbouring batches stabilizes non-greedy hypergradients and allows us to find schedules that can even outperform off-the-shelf schedules.

MNIST & SVHN. We start with HPO over medium length horizons with small networks, where reverse-mode differentiation can be used to learn many hyperparameters non-greedily as a baseline. In Figure 3, we learn the learning rate schedule starting from $\boldsymbol{\alpha} = \mathbf{0}$, in the maximally greedy and non-greedy setting, with and without hyperparameter sharing. We do not use tricks like online

Table 1: Test accuracy of our method compared to baselines ran for the same computational budget. MNIST and SVHN are optimized over 2 epochs ($\sim 10^3$ steps) while CIFAR-10 is optimized over 50 epochs ($\sim 10^4$ steps). Note how greediness gets worse as the dataset becomes more complex.

Method	MNIST	SVHN	CIFAR-10
Hand-tuned (best choice)	98.4 ± 0.1	85.1 ± 0.3	89.2 ± 0.2
Random Search	98.0 ± 0.2	82.2 ± 1.1	82.3 ± 0.7
Greedy (Baydin et al., 2018)	97.5 ± 0.1	84.0 ± 0.2	80.8 ± 0.2
Ours	$\mathbf{98.7} \pm 0.1$	$\mathbf{85.7} \pm 0.8$	$\mathbf{89.3} \pm 0.1$

learning as in (Baydin et al., 2018) to make greediness more transparent. As previously observed by Wu et al. (2018), greedy optimization leads to poor solutions with learning rates that are too small. While the vanilla non-greedy setting works well for simple datasets like MNIST, it fails to learn a good schedule for real-world datasets like SVHN, converging to much higher values than reasonable. By sharing hyperparameters, we stabilize the outer optimization by lowering hypervariance, and this allows us to learn a schedule that even beats an off-the-shelf baseline for this dataset.

CIFAR-10. We test Algorithm 1 on 50 epochs of CIFAR-10, for a Wide ResNet of 16 layers. Results are shown in Figure 1. We choose not to use larger architectures or more epochs to save compute time, and because we find that hyperparameters matter most for fewer epochs. Note also that we are not interested in the absolute performance of the model, but rather in the performance of the hyperparameters for any given model. To the best of our knowledge, we are the first to demonstrate a stable gradient-based optimization of the learning rate, momentum and weight decay for such a large number of steps ($\sim 10^4$). Interestingly, our algorithm mostly relies on high weight decays for regularization, which allows for smaller momentum with no performance drop.

Baseline Comparison. Finally, we compare our algorithm to baselines in Table 1. A common theme in meta-learning research has been the lack of appropriate baselines, with researchers often finding that random search (RS) can outperform complex algorithms, for instance in NAS (Li & Talwalkar, 2019) or automatic augmentation (Cubuk et al., 2020). We note that one of the reasons RS is a strong baseline is that it is non-greedy. Since we are using the sign of hypergradients, we can infer the range of search of our method, and perform RS over the same range for a comparable GPU time (~ 20 hours). We can also evaluate hypergradient descent (Baydin et al., 2018) with various initial hyperparameters in that range. Finally, we manually search for the best hyperparameters by using the ones that are most common in the literature, and evaluating multiple values around them (see Appendix D). Note that this manual baseline relies on strong inductive biases which wouldn't be available, say, when learning hyperparameters for a new unknown optimizer. Our method significantly outperforms both random search and hypergradient descent, while matching the performance of the best hand-tuned hyperparameters. The latter suggests that our method would be an efficient way to perform HPO for new applications or new optimizers, where good hyperparameter priors don't exist already.

5 Discussion

This work makes an important step towards gradient-based HPO for long horizons, by enabling non-greediness through the mitigation of gradient degradation. More specifically, we show that sharing contiguous hyperparameters is a simple and efficient way to reduce the variance of hypergradients. The nature of non-greedy bilevel optimization requires the inner optimization to be run several times, and so HPO over ImageNet-like datasets remains costly. However, we show that only ~ 10 outer steps are sufficient to converge to good hyperparameters, as opposed to the thousands conjectured in previous work (Wu et al., 2018). This is in part made possible by high quality hypergradients, and by using the fluctuation of their sign as a criterion for convergence. Extensions to this work could consider extending our method to discrete-valued hyperparameters with relaxation techniques, which would enable a larger range of non-greedy meta-learning applications. We hope that our work encourages the community to reconsider gradient-based hyperparameter optimization in terms of non-greediness, and pave the way towards a ubiquitous hyperparameter solver.

REFERENCES

Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*, 2018.

Y. Bengio, P. Frasconi, and P. Simard. The problem of learning long-term dependencies in recurrent networks. In *IEEE International Conference on Neural Networks*, 1993.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994.

Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural Comput.*, August 2000. ISSN 0899-7667. doi: 10.1162/089976600300015187.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

E. D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3008–3017, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Justin Domke. Generic methods for optimization-based modeling. In Neil D. Lawrence and Mark Girolami (eds.), *International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

Xuanyi Dong, Mingxing Tan, Adams Wei Yu, Daiyi Peng, Bogdan Gabrys, and Quoc V. Le. Autohas: Differentiable hyper-parameter and architecture search, 2020.

Michele Donini, Luca Franceschi, Massimiliano Pontil, Orchid Majumder, and Paolo Frasconi. Scheduling the Learning Rate via Hypergradients: New Insights and a New Algorithm. *arXiv e-prints*, art. arXiv:1910.08525, October 2019.

Matthias Feurer and Frank Hutter. *Hyperparameter Optimization*. Springer International Publishing, Cham, 2019. ISBN 978-3-030-05318-5. doi: 10.1007/978-3-030-05318-5_1.

Sebastian Flennerhag, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-learning with warped gradient descent. In *International Conference on Learning Representations*, 2020.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In Doina Precup and Yee Whye Teh (eds.), *International conference on Machine Learning*, Proceedings of Machine Learning Research, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

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*, 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 *IJCAI*, 2016.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei (eds.), *International conference on Machine Learning*, Proceedings of Machine Learning Research, Lille, France, 07–09 Jul 2015. PMLR.

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.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *International Conference on Learning Representations*, 2015.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *UAI*, 2019.

Mengtian Li, Ersin Yumer, and Deva Ramanan. Budgeted training: Rethinking deep neural network training under resource constraints. In *International Conference on Learning Representations*, 2020.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing Millions of Hyperparameters by Implicit Differentiation. *arXiv e-prints*, art. arXiv:1911.02590, November 2019.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

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

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based Hyperparameter Optimization through Reversible Learning. *arXiv e-prints*, art. arXiv:1502.03492, February 2015.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. volume 97 of *Proceedings of Machine Learning Research*, pp. 4556–4565, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. PIPPS: Flexible model-based policy search robust to the curse of chaos. volume 80 of *Proceedings of Machine Learning Research*, pp. 4065–4074, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on Machine Learning*, 2013.

Adam Paszke, 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. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International conference on Machine Learning*, 2016.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.

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

Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *AISTATS*, 2019.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on Machine Learning*, 2015.

H Stackelberg. *The Theory Of Market Economy*. Oxford University Press, 1952.

P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 1990.

R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1989.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*, 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Appendices

A: Forward-mode Hypergradient Derivations

Recall that we are interested in calculating

$$\boldsymbol{Z}_t = \boldsymbol{A}_t \boldsymbol{Z}_{t-1} + \boldsymbol{B}_t$$

recursively during the inner loop, where

$$\boldsymbol{Z}_t = \frac{d\boldsymbol{\theta}_t}{d\boldsymbol{\lambda}} \qquad \boldsymbol{A}_t = \left.\frac{\partial \boldsymbol{\theta}_t}{\partial \boldsymbol{\theta}_{t-1}}\right|_{\boldsymbol{\lambda}} \qquad \boldsymbol{B}_t = \left.\frac{\partial \boldsymbol{\theta}_t}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\theta}_{t-1}}$$

so that we can calculate the hypergradients on the final step using

$$\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}} = \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{\theta}_T} \boldsymbol{Z}_T$$

Each type of hyperparameter needs its own matrix $\boldsymbol{Z}_t$, and therefore its own matrices $\boldsymbol{A}_t$, and $\boldsymbol{B}_t$.

Consider first the derivation of these matrices for the learning rate, namely $\boldsymbol{\lambda} = \boldsymbol{\alpha}$. Recall that the update rule after substituting the velocity $\boldsymbol{v}_t$ in is

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha_t \left(\beta_t \boldsymbol{v}_{t-1} + \frac{\partial \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}} + \mu_t \boldsymbol{\theta}_{t-1} \right)$$

and therefore it follows directly that

$$\boldsymbol{A}_t^{\boldsymbol{\alpha}} = \mathbb{1}^{D \times D} - \alpha_t \left(\frac{\partial^2 \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}^2} + \mu_t \mathbb{1}^{D \times D} \right)$$

The calculation of $\boldsymbol{B}_t^{\boldsymbol{\alpha}}$ is a bit more involved in our work because when using momentum $\boldsymbol{v}_{t-1}$ is now itself a function of $\boldsymbol{\alpha}$. First we write

$$\begin{aligned}
\boldsymbol{B}_t^{\boldsymbol{\alpha}} &= -\beta_t \left(\frac{\partial \alpha_t}{\partial \boldsymbol{\alpha}} \boldsymbol{v}_{t-1} + \alpha_t \frac{\partial \boldsymbol{v}_{t-1}}{\partial \boldsymbol{\alpha}} \right) - \frac{\partial \alpha_t}{\partial \boldsymbol{\alpha}} \left(\frac{\partial \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}} + \mu_t \boldsymbol{\theta}_{t-1} \right) \\
&= -\beta_t \alpha_t \frac{\partial \boldsymbol{v}_{t-1}}{\partial \boldsymbol{\alpha}} - \delta_t^{D \times N} \left(\beta_t \boldsymbol{v}_{t-1} + \frac{\partial \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}} + \mu_t \boldsymbol{\theta}_{t-1} \right)
\end{aligned}$$

Now since

$$\boldsymbol{v}_t = \beta_t \boldsymbol{v}_{t-1} + \frac{\partial \mathcal{L}_{train}}{\boldsymbol{\theta}_{t-1}} + \mu_t \boldsymbol{\theta}_{t-1}$$

we can write the partial derivative of the velocity as an another recursive rule:

$$\begin{aligned}
\boldsymbol{C}_t^{\boldsymbol{\alpha}} &= \frac{\partial \boldsymbol{v}_t}{\partial \boldsymbol{\alpha}} \\
&= \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} + \frac{\partial^2 \mathcal{L}_{train}}{\partial \boldsymbol{\alpha} \partial \boldsymbol{\theta}_{t-1}} + \mu_t \frac{\partial \boldsymbol{\theta}_{t-1}}{\partial \boldsymbol{\alpha}} \\
&= \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} + \left(\mu_t \mathbb{1}^{D \times D} + \frac{\partial^2 \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}^2} \right) \frac{\partial \boldsymbol{\theta}_{t-1}}{\partial \boldsymbol{\alpha}}
\end{aligned}$$

And putting all together recovers the system:

$$\begin{cases} \boldsymbol{A}_t^{\boldsymbol{\alpha}} = \mathbb{1}^{D\times D} - \alpha_t \left(\frac{\partial^2 \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}^2} + \mu_t \mathbb{1}^{D\times D} \right) \\ \boldsymbol{B}_t^{\boldsymbol{\alpha}} = -\beta_t \alpha_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} - \delta_t^{D\times N} \left(\beta_t \boldsymbol{v}_{t-1} + \frac{\partial \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}} + \mu_t \boldsymbol{\theta}_{t-1} \right) \\ \boldsymbol{C}_t^{\boldsymbol{\alpha}} = \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} + \left(\mu_t \mathbb{1}^{D\times D} + \frac{\partial^2 \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}^2} \right) \boldsymbol{Z}_{t-1}^{\boldsymbol{\alpha}} \end{cases}$$

For learning the momentum and weight decay, a very similar approach yields

$$\begin{cases} \boldsymbol{A}_t^{\boldsymbol{\beta}} = \mathbb{1}^{D\times D} - \alpha_t \left(\frac{\partial^2 \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}^2} + \mu_t \mathbb{1}^{D\times D} \right) \\ \boldsymbol{B}_t^{\boldsymbol{\beta}} = -\beta_t \alpha_t \boldsymbol{C}_{t-1}^{\boldsymbol{\beta}} - \delta_t^{D\times N} (\alpha_t \boldsymbol{v}_{t-1}) \\ \boldsymbol{C}_t^{\boldsymbol{\beta}} = \delta_t^{D\times N} (\boldsymbol{v}_t) + \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\beta}} + \left(\mu_t \mathbb{1}^{D\times D} + \frac{\partial^2 \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}^2} \right) \boldsymbol{Z}_{t-1}^{\boldsymbol{\beta}} \end{cases}$$

and

$$\begin{cases} \boldsymbol{A}_t^{\boldsymbol{\mu}} = \mathbb{1}^{D\times D} - \alpha_t \left(\frac{\partial^2 \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}^2} + \mu_t \mathbb{1}^{D\times D} \right) \\ \boldsymbol{B}_t^{\boldsymbol{\mu}} = -\beta_t \alpha_t \boldsymbol{C}_{t-1}^{\boldsymbol{\mu}} - \delta_t^{D\times N} (\alpha_t \boldsymbol{\theta}_{t-1}) \\ \boldsymbol{C}_t^{\boldsymbol{\mu}} = \delta_t^{D\times N} (\boldsymbol{\theta}_{t-1}) + \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\mu}} + \left(\mu_t \mathbb{1}^{D\times D} + \frac{\partial^2 \mathcal{L}_{train}}{\partial \boldsymbol{\theta}_{t-1}^2} \right) \boldsymbol{Z}_{t-1}^{\boldsymbol{\mu}} \end{cases}$$

B: Implementation Details

Figure 1. We learn 5 values for the learning rates, 1 for the momentum and 1 for the weight decay, to make it comparable to the hyperparameters used in the literature for CIFAR-10 (see Appendix D). A batch size 256 is used, with 5% of the training set of each epoch set aside for validation. We found larger validation sizes not to be helpful. Hypergradient descent uses hyperparameters initialized at zero as well, and trains all hyperparameters online with an SGD outer optimizer with learning rate 0.2 and ± 1 clipping of the hypergradients. We used initial values $\gamma_\alpha = 0.1, \gamma_\beta = 0.15$ and $\gamma_\mu = 4 \times 10^{-4}$ but the performance barely changed when these values were multiplied or divided by 2. Truncation of only 15% of inner steps was used so as not to reduce performance. The Hessian matrix product is clipped to ± 10 to prevent one batch from having a dominating contribution to hypergradients. Strictly speaking, sharing hyperparameters corresponds to summing their hypergradients rather than averaging them, and therefore we divide this sum by the number of batches each hyperparameter is used for.

Figure 2. Here we wanted to isolate all factors responsible for hypervariance. We thus used float64 precision with batch size 64, as this reduced hypervariance across all methods. Clipping did not change the hypervariance drastically but was applied to ± 1 in the inner loop. We learned 10 learning rates for our method, namely one learning rate per 40 steps. The unperturbed learning rate was set to $\boldsymbol{\alpha}_0 = 0.05$. The perturbation for initial weights and hyperparameters (here $\boldsymbol{\alpha}$) corresponds to adding $\pm 1\%$ of their value, while perturbation of $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ correspond to a different sampling seed. For the plots, we used a moving average for the greedy and non-greedy lines for clarity.

Figure 3. Here we used a batch size of 128 for both datasets to allow 2 epochs worth of inner optimization in about 500 inner steps. Clipping was restricted to ± 3 to show the effect of noisy hypergradients more clearly. Since MNIST and SVHN are cheap datasets to run on a LeNet architecture, we can afford 50 outer steps and early stopping based on validation accuracy.

Table 1. Each method is run for a similar GPU budget of about 20 hours on a 2080 GTX GPU. Random search and our method both consider 7 learning rates, 1 momentum and 1 weight decay. Truncation of more than 15% of inner steps starts slowly lowering the performance of our method, and so is kept at this minimum for these experiments. Note that random search struggles in finding good hyperparameters even though ~ 100 random hyperparameter settings are evaluated, because some hyperparameters can compromise the whole training process. This is the case of large learning rates, negative last learning rates, or momentums greater than one. Since random search is a memory-less trial-and-error method, any hyperparameter region that compromises learning altogether is very harmful. The error bars are calculated over 3 seeds for each entry in the table.

C: Hypergradients

Here we provide the raw hypergradients corresponding to the outer optimization shown in Figure 1. Note that the range of these hypergradients is made reasonable by the averaging of gradients coming from contiguous hyperparameters.

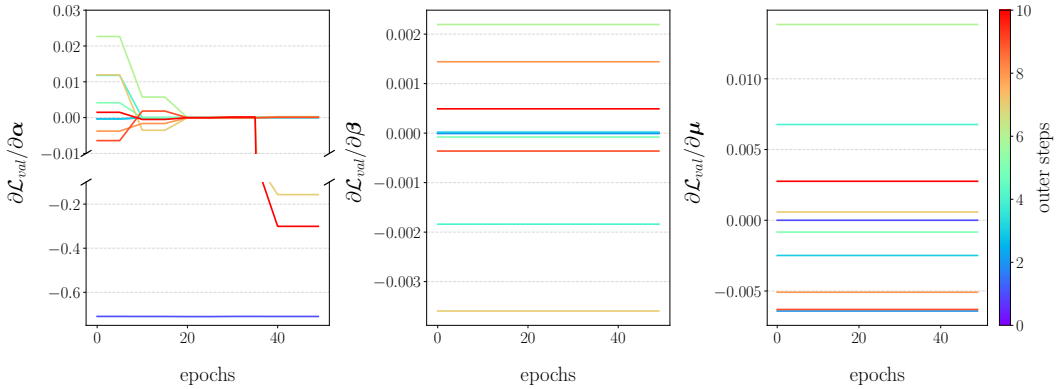

Figure 4: Hypergradients have a reasonable range but fail to always converge to zero when the validation performance stops improving.

D: Baselines

The objective here is to select the best hyperparameter setting that a deep learning practitioner would reasonably be expected to use, based on the hyperparameters used by the community for the datasets at hand. For CIFAR-10, the most common hyperparameter setting is the following: α is initialized at $\alpha_0 = 0.2$ (for batch size 256, as used in our experiments) and decayed by a factor $\eta = 0.2$ at $30\%, 60\%$ and 80% of the run (`MultiStep` in Pytorch); the momentum β is constant at 0.9, and the weight decay μ is constant at 5×10^{-4}. We search for combinations of hyperparameters around this setting. More specifically, we search over all combinations of $\alpha_0 = \{0.05, 0.1, 0.2, 0.4, 0.6\}$, $\eta = \{0.1, 0.2, 0.4\}$, $\beta = \{0.45, 0.9, 0.99\}$, and $\mu = \{2.5 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}$. This makes up a total of 135 hyperparameter settings, which we each run 3 times to get a mean and standard deviation. The distribution of those means are provided in Figure 5, and the best hyperparameter setting is picked based on validation performance. This is the value we report in Table 1 under *Hand-tuned (best)*.

MNIST and SVHN hyperparameters matter less, and in particular we observed no gain from using momentum and weight decay. The most popular learning rate schedules used for these datasets seem to be the cosine annealing one. We evaluate this schedule for $\alpha_0 = \{0.05, 0.1, 0.2, 0.4, 0.6\}$ and select the best hyperparameters based on validation performance.

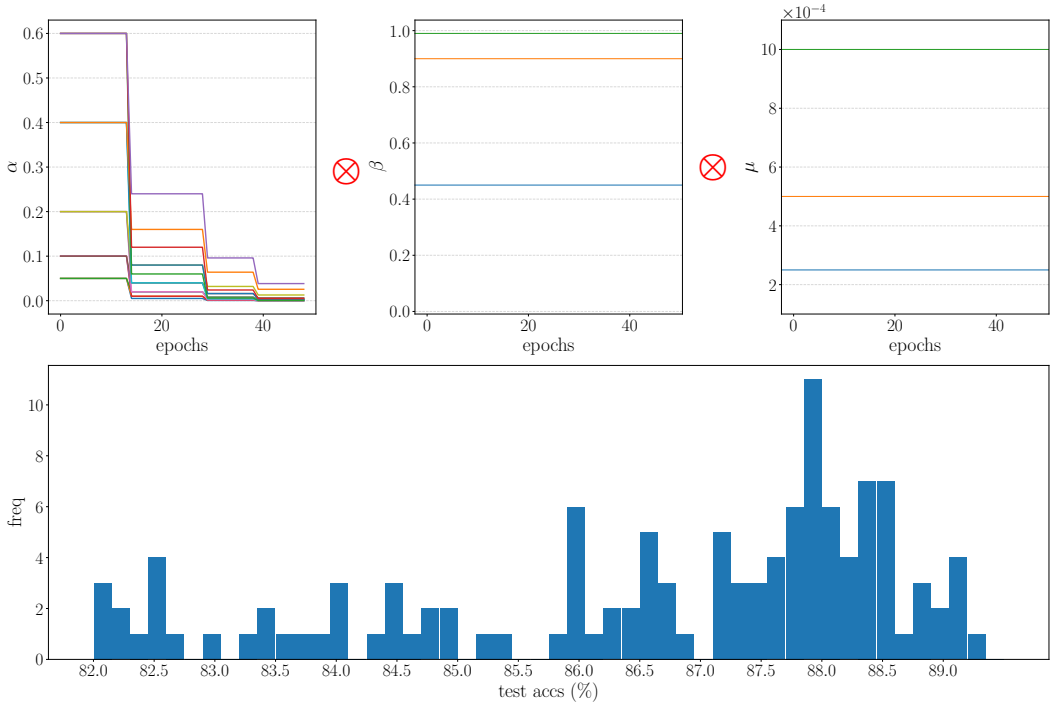

Figure 5: The combination of hyperparameters searched over for CIFAR-10 (top row) and the corresponding distribution of test accuracies (bottom row).