APPENDICES

Anonymous authors
Paper under double-blind review

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:

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

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

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

and

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

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 ± 100 to prevent one batch from having a dominating contribution to hypergradients.

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 $\sim$ 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.

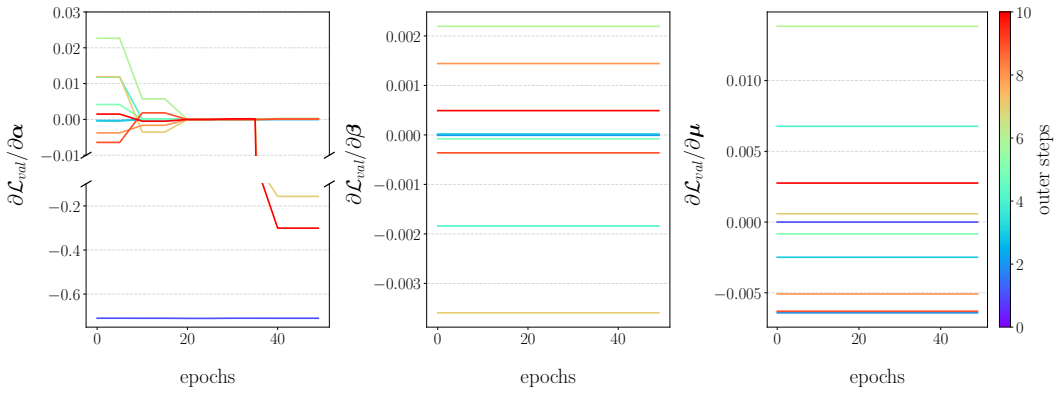

Figure 4: Hypergradients have a reasonable range but fail to always converge to zero when the validation performance stops improving.

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, as for CIFAR-10.

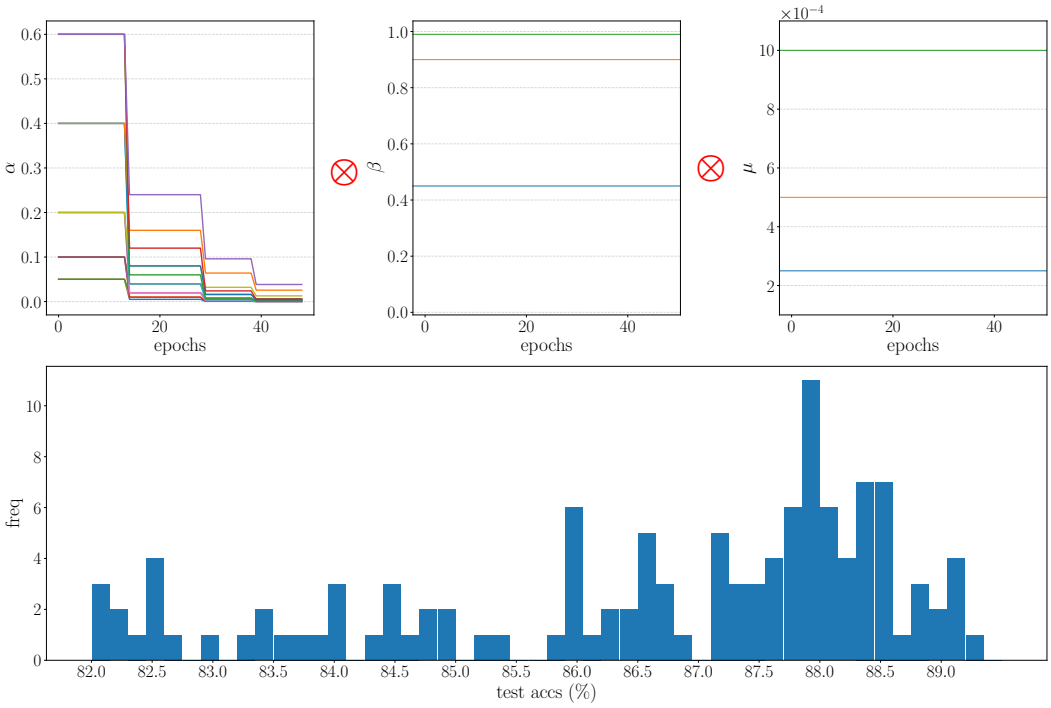

Figure 5: The combination of hyperparameters searched over for CIFAR-10 (top row) and the corresponding distribution of test accuracies (bottom row).