Optimizing Millions of Hyperparameters by Implicit Differentiation Appendix

A Extended Background

In this section we provide an outline of our notation (Table 5), and the proposed algorithm. Here, we assume we have access to to a finite dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) | i = 1 \dots n\}$, with n examples drawn from the distribution $p(\boldsymbol{x}, \boldsymbol{y})$ with support $\mathcal{P}$. We denote the input and target domains by $\mathcal{X}$ and $\mathcal{Y}$, respectively. Assume $\boldsymbol{y} : \mathcal{X} \to \mathcal{Y}$ is a function and we wish to learn $\hat{\boldsymbol{y}} : \mathcal{X} \times \mathbf{W} \to \mathcal{Y}$ with a NN parameterized by $\mathbf{w} \in \mathbf{W}$, s.t. $\hat{\boldsymbol{y}}$ is close to $\boldsymbol{y}$. We measure how close a predicted value is to a target with the prediction loss $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. Our goal is to minimize the expected prediction loss or population risk: $\arg\min_{\mathbf{w}} \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x})}[\mathcal{L}(\hat{\boldsymbol{y}}(\boldsymbol{x}, \mathbf{w}), \boldsymbol{y}(\boldsymbol{x}))]$. Since we only have access to a finite number of samples, we minimize the empirical risk: $\arg\min_{\mathbf{w}} {}^{1}/_{n} \sum_{\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}} \mathcal{L}(\hat{\boldsymbol{y}}(\boldsymbol{x}, \mathbf{w}), \boldsymbol{y}(\boldsymbol{x}))$.

Due to a limited size dataset $\mathcal{D}$, there may be a significant difference between the minimizer of the empirical risk and the population risk. We can estimate this difference by partitioning our dataset into training and validation datasets— $\mathcal{D}_{train}, \mathcal{D}_{valid}$. We find the minimizer over the training dataset $\mathcal{D}_{train}$, and estimate its performance on the population risk by evaluating the empirical risk over the validation dataset $\mathcal{D}_{valid}$. We introduce modifications to the empirical training risk to decrease our population risk, parameterized by $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$. These parameters for generalization are called the hyperparameters. We call the modified empirical training risk our training loss for simplicity and denote it $\mathcal{L}_{\mathrm{T}}(\boldsymbol{\lambda}, \mathbf{w})$. Our validation empirical risk is called validation loss for simplicity and denoted by $\mathcal{L}_{\mathrm{V}}(\boldsymbol{\lambda}, \mathbf{w})$. Often the validation loss does not directly depend on the hyperparameters, and we just have $\mathcal{L}_{\mathrm{V}}(\mathbf{w})$.

The population risk is estimated by plugging the training loss minimizer $\mathbf{w}^*(\boldsymbol{\lambda}) = \arg\min_{\mathbf{w}} \mathcal{L}_{\mathrm{T}}(\boldsymbol{\lambda}, \mathbf{w})$ into the validation loss for the estimated population risk $\mathcal{L}_{\mathrm{V}}^*(\boldsymbol{\lambda}) = \mathcal{L}_{\mathrm{V}}(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))$. We want our hyperparameters to minimize the estimated population risk: $\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}} \mathcal{L}_{\mathrm{V}}^*(\boldsymbol{\lambda})$. We can create a third partition of our dataset $\mathcal{D}_{test}$ to assess if we have overfit the validation dataset $\mathcal{D}_{valid}$ with our hyperparameters $\boldsymbol{\lambda}$.

B Extended Related Work

Independent HO: A simple class of HO algorithms involve making a number of independent hyperparameter selections, and training the model to completion on them. Popular examples include grid search and random search [Bergstra and Bengio, 2012]. Since each hyperparameter selection is independent, these algorithms are trivial to parallelize.

Global HO: Some HO algorithms attempt to find a globally optimal hyperparameter setting, which can be important if the loss is non-convex. A simple example is random search, while a more sophisticated example is Bayesian optimization Močkus [1975], Snoek et al. [2012], Kandasamy et al. [2019]. These HO algorithms often involve re-initializing the hyperparameter and weights on each optimization iteration. This allows global optimization, at the cost of expensive re-training weights or hyperparameters.

Local HO: Other HO algorithms only attempt to find a locally optimal hyperparameter setting. Often these algorithms will maintain a current estimate of the best combination of hyperparameter and weights. On each optimization iteration, the hyperparameter is adjusted by a small amount, which allows us to avoid excessive re-training of the weights on each update. This is because the new optimal weights are near the old optimal weights due to a small change in the hyperparameters.

Learned proxy function based HO: Many HO algorithms attempt to learn a proxy function for optimization. The proxy function is used to estimate the loss for a hyperparameter selection. We could learn a proxy function for global or local HO . We can learn a useful proxy function over any node in our computational graph including the optimized weights. For example, we could learn how the optimized weights change w.r.t. the hyperparameters Lorraine and Duvenaud [2018], how the optimized predictions change w.r.t. the hyperparameters MacKay et al. [2019], or how the optimized validation loss changes w.r.t. the hyperparameters as in Bayesian Optimization. It is possible to do gradient descent on the proxy function to find new hyperparameters to query as in Bayesian optimization. Alternatively, we could use a non-differentiable proxy function to get cheap estimates of the validation loss like SMASH Brock et al. [2018] for architecture choices.

Table 5: **Notation**

HO	Hyperparameter optimization
NN	Neural network
IFT	Implicit Function Theorem
HVP / JVP	Hessian/Jacobian-vector product
$\boldsymbol{\lambda}, \mathbf{w}$	Hyperparameters and NN parameters/weights
n, m	Hyperparameter and NN parameter dimensionality
$\boldsymbol{\Lambda} \subseteq \mathbb{R}^n, \mathbf{W} \subseteq \mathbb{R}^m$	Hyperparameters and NN parameter domains
$\boldsymbol{\lambda}', \mathbf{w}'$	Arbitrary, fixed hyperparameters and weights
$\mathcal{L}_{\mathrm{T}}(\boldsymbol{\lambda}, \mathbf{w}), \mathcal{L}_{\mathrm{V}}(\boldsymbol{\lambda}, \mathbf{w})$	Training loss & validation loss
$\mathbf{w}^*(\boldsymbol{\lambda})$	Best-response of the weights to the hyperparameters
$\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$	An approximate best-response of the weights to the hyperparameters
$\mathcal{L}_{\mathrm{V}}^*(\boldsymbol{\lambda}) = \mathcal{L}_{\mathrm{V}}(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))$	The validation loss with best-responding weights
Red	(Approximations to) The validation loss with best-responding weights
$\mathbf{W}^* = \mathbf{w}^*(\boldsymbol{\Lambda})$	The domain of best-responding weights
$\boldsymbol{\lambda}^*$	The optimal hyperparameters
$\boldsymbol{x}, \boldsymbol{y}$	An input and its associated target
$\mathcal{X}, \mathcal{Y}$	The input and target domains respectively
$\mathcal{D}$	A data matrix consisting of tuples of inputs and targets
$\boldsymbol{y}(\boldsymbol{x}, \mathbf{w})$	A predicted target for a input data and weights
$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}}, \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}}$	The (validation loss hyperparameter / parameter) direct gradient
Green	(Approximations to) The validation loss direct gradient.
$\frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}$	The best-response Jacobian
Blue	(Approximations to) The (Jacobian of the) best-response of the weights to the hyperparameters
$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}$	The indirect gradient
$\frac{\partial \mathcal{L}_{\mathrm{V}}^*}{\partial \boldsymbol{\lambda}}$	A hypergradient: sum of validation losses direct and indirect gradient
$\left[\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1}$	The training Hessian inverse
Magenta	(Approximations to) The training Hessian inverse
$\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \left[\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1}$	The vector - Inverse Hessian product.
Orange	(Approximations to) The vector - Inverse Hessian product.
$\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}$	The training mixed partial derivatives
I	The identity matrix

C Implicit Function Theorem

Theorem (Augustin-Louis Cauchy, Implicit Function Theorem)**.** Let $\frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}}(\boldsymbol{\lambda}, \mathbf{w}) : \boldsymbol{\Lambda} \times \mathbf{W} \to \mathbf{W}$ be a continuously differentiable function. Fix a point $(\boldsymbol{\lambda}', \mathbf{w}')$ with $\frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}}(\boldsymbol{\lambda}', \mathbf{w}') = 0$. If the Jacobian $J_{\mathbf{w}}^{\frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}}}(\boldsymbol{\lambda}', \mathbf{w}')$ is invertible, there exists an open set $U \subseteq \boldsymbol{\Lambda}$ containing $\boldsymbol{\lambda}'$ s.t. there exists a continuously differentiable function $\mathbf{w}^* : U \to \mathbf{W}$ s.t.:

$$\mathbf{w}^*(\boldsymbol{\lambda}') = \mathbf{w}' \text{ and } \forall \boldsymbol{\lambda} \in U, \frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}}(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))) = 0$$

Moreover, the partial derivatives of $\mathbf{w}^*$ in U are given by the matrix product:

$$\frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}(\boldsymbol{\lambda}) = -\left[J_{\mathbf{w}}^{\frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}}}(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})) \right]^{-1} J_{\boldsymbol{\lambda}}^{\frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}}}(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})))$$

Typically the IFT is presented with $\frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}} = f$, $\mathbf{w}^* = g$, $\boldsymbol{\Lambda} = \mathbb{R}^m$, $\mathbf{W} = \mathbb{R}^n$, $\boldsymbol{\lambda} = x$, $\mathbf{w} = y$, $\boldsymbol{\lambda}' = a$, $\mathbf{w}' = b$.

D Proofs

Lemma (1)**.** If the recurrence given by unrolling SGD optimization in Eq. 5 has a fixed point $\mathbf{w}_\infty$ (i.e., $0 = \frac{\partial \mathcal{L}_\mathrm{T}}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}, \mathbf{w}_\infty(\boldsymbol{\lambda})}$), then:

$$\frac{\partial \mathbf{w}_\infty}{\partial \boldsymbol{\lambda}} = -\left[\frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1} \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\bigg|_{\mathbf{w}_\infty(\boldsymbol{\lambda})}$$

Proof.

$$\Rightarrow \frac{\partial}{\partial \boldsymbol{\lambda}}\left(\frac{\partial \mathcal{L}_\mathrm{T}}{\partial \mathbf{w}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_\infty(\boldsymbol{\lambda})}\right) = 0 \qquad \text{given}$$

$$\Rightarrow \left(\frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}} I + \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}} \frac{\partial \mathbf{w}_\infty}{\partial \boldsymbol{\lambda}}\right)\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_\infty(\boldsymbol{\lambda})} = 0 \qquad \text{chain rule through } |_{\boldsymbol{\lambda}, \mathbf{w}_\infty(\boldsymbol{\lambda})}$$

$$\Rightarrow \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}} \frac{\partial \mathbf{w}_\infty}{\partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_\infty(\boldsymbol{\lambda})} = -\frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_\infty(\boldsymbol{\lambda})} \qquad \text{re-arrange terms}$$

$$\Rightarrow \frac{\partial \mathbf{w}_\infty}{\partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}} = -\left[\frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1} \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}} \qquad \text{left-multiply by } \left[\frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_\infty(\boldsymbol{\lambda})}$$

□

Lemma (2)**.** Given the recurrence from unrolling SGD optimization in Eq. 5 we have:

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

Proof.

$$\frac{\partial \mathbf{w}_{i+1}}{\partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}} = \frac{\partial}{\partial \boldsymbol{\lambda}}\left(\mathbf{w}_i(\boldsymbol{\lambda}) - \frac{\partial \mathcal{L}_\mathrm{T}}{\partial \mathbf{w}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda})}\right) \qquad \text{take derivative w.r.t. } \boldsymbol{\lambda}$$

$$= \frac{\partial \mathbf{w}_i}{\partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}} - \frac{\partial}{\partial \boldsymbol{\lambda}}\left(\frac{\partial \mathcal{L}_\mathrm{T}}{\partial \mathbf{w}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda})}\right) \qquad \text{chain rule}$$

$$= \frac{\partial \mathbf{w}_i}{\partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}} - \left(\frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}} \frac{\partial \mathbf{w}_i}{\partial \boldsymbol{\lambda}} + \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\right)\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda})} \qquad \text{chain rule through } |_{\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda})}$$

$$= -\frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda})} + \left(I - \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\right) \frac{\partial \mathbf{w}_i}{\partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda})} \qquad \text{re-arrange terms}$$

$$= -\frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda})} + \left(I - \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\right)\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda})} \cdot$$

$$\left(\left(I - \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\right) \frac{\partial \mathbf{w}_{i-1}}{\partial \boldsymbol{\lambda}} - \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\right)\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_{i-1}(\boldsymbol{\lambda})} \qquad \text{expand } \frac{\partial \mathbf{w}_i}{\partial \boldsymbol{\lambda}}$$

$$= -\frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda})} - \left(I - \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda})}\right) \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_{i-1}(\boldsymbol{\lambda})} +$$

$$\left[\prod_{k<2} I - \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_{i-k}(\boldsymbol{\lambda})}\right] \frac{\partial \mathbf{w}_{i-1}}{\partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}} \qquad \text{re-arrange terms}$$

$$= \cdots$$

$$\text{So, } \frac{\partial \mathbf{w}_{i+1}}{\partial \boldsymbol{\lambda}} = -\sum_{j \le i}\left[\prod_{k<j} I - \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_{i-k}(\boldsymbol{\lambda})}\right] \frac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\bigg|_{\boldsymbol{\lambda}, \mathbf{w}_{i-j}(\boldsymbol{\lambda})} \qquad \text{telescope the recurrence}$$

□

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

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

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

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

Proof.

$$\begin{aligned}
&\lim_{i \to \infty} \tfrac{\partial \mathbf{w}_{i+1}}{\partial \boldsymbol{\lambda}}\Big|_{\boldsymbol{\lambda}} && \text{take } \lim_{i \to \infty} \\
&= \lim_{i \to \infty} \left(-\sum_{j \leq i} \left[\prod_{k<j} I - \tfrac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\Big|_{\boldsymbol{\lambda}, \mathbf{w}_{i-k}(\boldsymbol{\lambda})}\right] \tfrac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\Big|_{\boldsymbol{\lambda}, \mathbf{w}_{i-j}(\boldsymbol{\lambda})}\right) && \text{by Lemma 2} \\
&= -\lim_{i \to \infty} \left(\sum_{j \leq i} \left[\prod_{k<j} I - \tfrac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\right] \tfrac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\right)\Bigg|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})} && \mathbf{w}_0 = \mathbf{w}^*(\boldsymbol{\lambda}) = \mathbf{w}_i \\
&= -\lim_{i \to \infty} \left(\sum_{j \leq i} \left[I - \tfrac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\right]^j\right) \tfrac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\Bigg|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})} && \text{simplify} \\
&= -\left[I - \left(I - \tfrac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\right)\right]^{-1} \tfrac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\Bigg|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})} && \text{contractive \& Neumann series} \\
&= -\left[\tfrac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1} \tfrac{\partial^2 \mathcal{L}_\mathrm{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\Bigg|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})} && \text{simplify}
\end{aligned}$$

□

E Experiments

We use PyTorch Paszke et al. [2017] as our computational framework. All experiments were performed on NVIDIA TITAN Xp GPUs.

For all CNN experiments we use the following optimization setup: for the NN weights we use Adam Kingma and Ba [2014] with a learning rate of 1e-4. For the hyperparameters we use RMSprop Hinton et al. [2012] with a learning rate of 1e-2.

E.1 Overfitting a Small Validation Set

We see our algorithm's ability to overfit the validation data (see Fig. 8). We use 50 training input, and 50 validation input with the standard testing partition for both MNIST and CIFAR-10. We check performance with logistic regression (Linear), a 1-Layer fully-connected NN with as many hidden units as input size (ex., $28 \times 28 = 784$, or $32 \times 32 \times 3 = 3072$), LeNet LeCun et al. [1998], AlexNet Krizhevsky et al. [2012], and ResNet44 He et al. [2016]. In all examples we can achieve 100 % training and validation accuracy, while the testing accuracy is significantly lower.

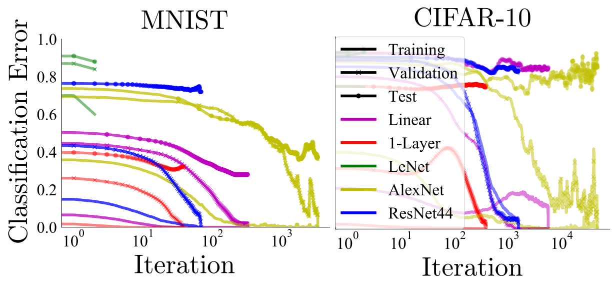

Figure 8: Overfitting validation data. Algorithm 1 can overfit the validation dataset. We use 50 training input, and 50 validation input with the standard testing partition for both MNIST and CIFAR-10. We check the performance with logistic regression (Linear), a 1-Layer fully-connected NN with as many hidden units as input size (ex., $28\times28 = 784$, or $32\times32\times3 = 3072$), LeNet LeCun et al. [1998], AlexNet Krizhevsky et al. [2012], and ResNet44 He et al. [2016]. Separate lines are plotted for the training, validation, and testing error. In all examples we achieve 100 % training and validation accuracy, while the testing accuracy is significantly lower.

E.2 Dataset Distillation

With MNIST we use the entire dataset in validation, while for CIFAR we use 300 validation data points.

E.3 Learned Data Augmentation

Augmentation Network Details: Data augmentation can be framed as an image-to-image transformation problem; inspired by this, we use a U-Net Ronneberger et al. [2015] as the data augmentation network. To allow for stochastic transformations, we feed in random noise by concatenating a noise channel to the input image, so the resulting input has 4 channels. For training, validation, and testing we evaluate the accuracy with an average over 10 augmented samples.

E.4 RNN Hyperparameter Optimization

We base our implementation on the AWD-LSTM codebase [1]. Similar to Gal and Ghahramani [2016] we used a 2-layer LSTM with 650 hidden units per layer and 650-dimensional word embeddings.

Overfitting Validation Data: We used a subset of 10 training sequences and 10 validation sequences, and tuned separate weight decays per parameter. The LSTM architecture we use has 13 280 400 weights, and thus an equal number of weight decay hyperparameters.

Optimization Details: For the large-scale experiments, we follow the training setup proposed in Merity et al. [2018]: for the NN weights, we use SGD with learning rate 30 and gradient clipping to magnitude 0.25. The learning rate was decayed by a factor of 4 based on the nonmonotonic criterion introduced by Merity et al. [2018] (i.e., when the validation loss fails to decrease for 5 epochs). To optimize the hyperparameters, we used Adam with learning rate 0.001. We trained on sequences of length 70 in mini-batches of size 40.

[1] `https://github.com/salesforce/awd-lstm-lm`

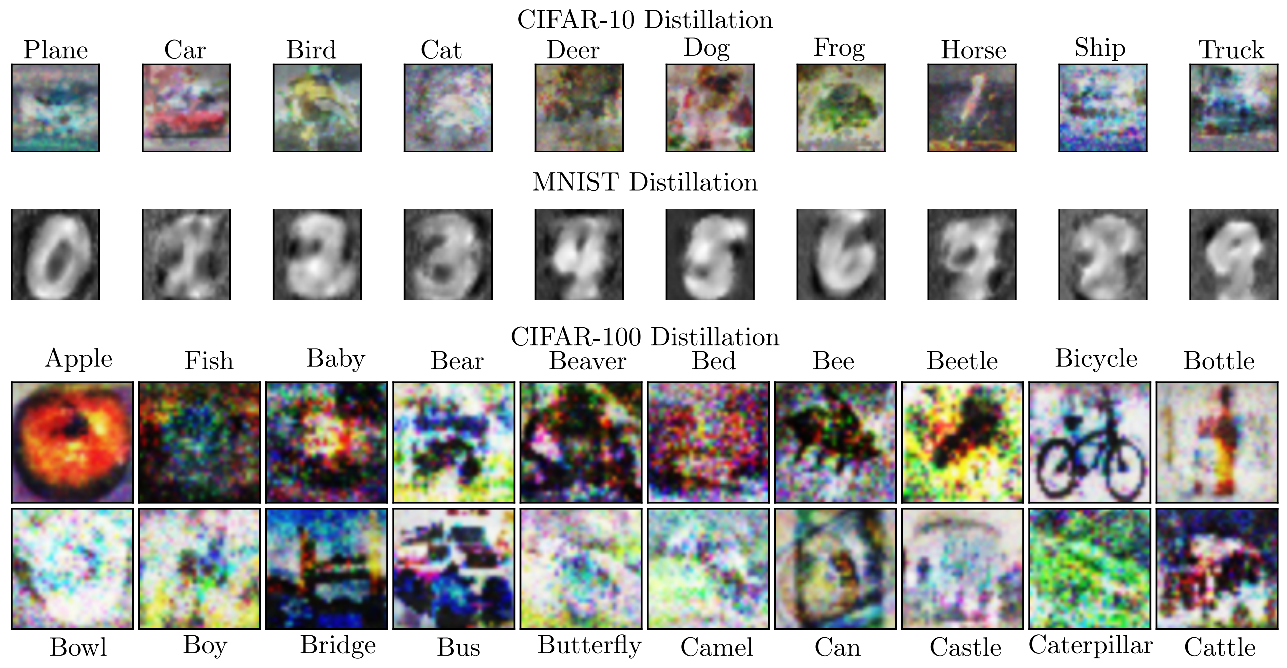

Figure 9: Distilled datasets for CIFAR-10, MNIST, and CIFAR-100. For CIFAR-100, we show the first 20 classes—the rest are in Appendix Fig. 10. We learn one distilled image per class, so after training a logistic regression classifier on the distillation, it generalizes to the rest of the data.

CIFAR-100 Distillation

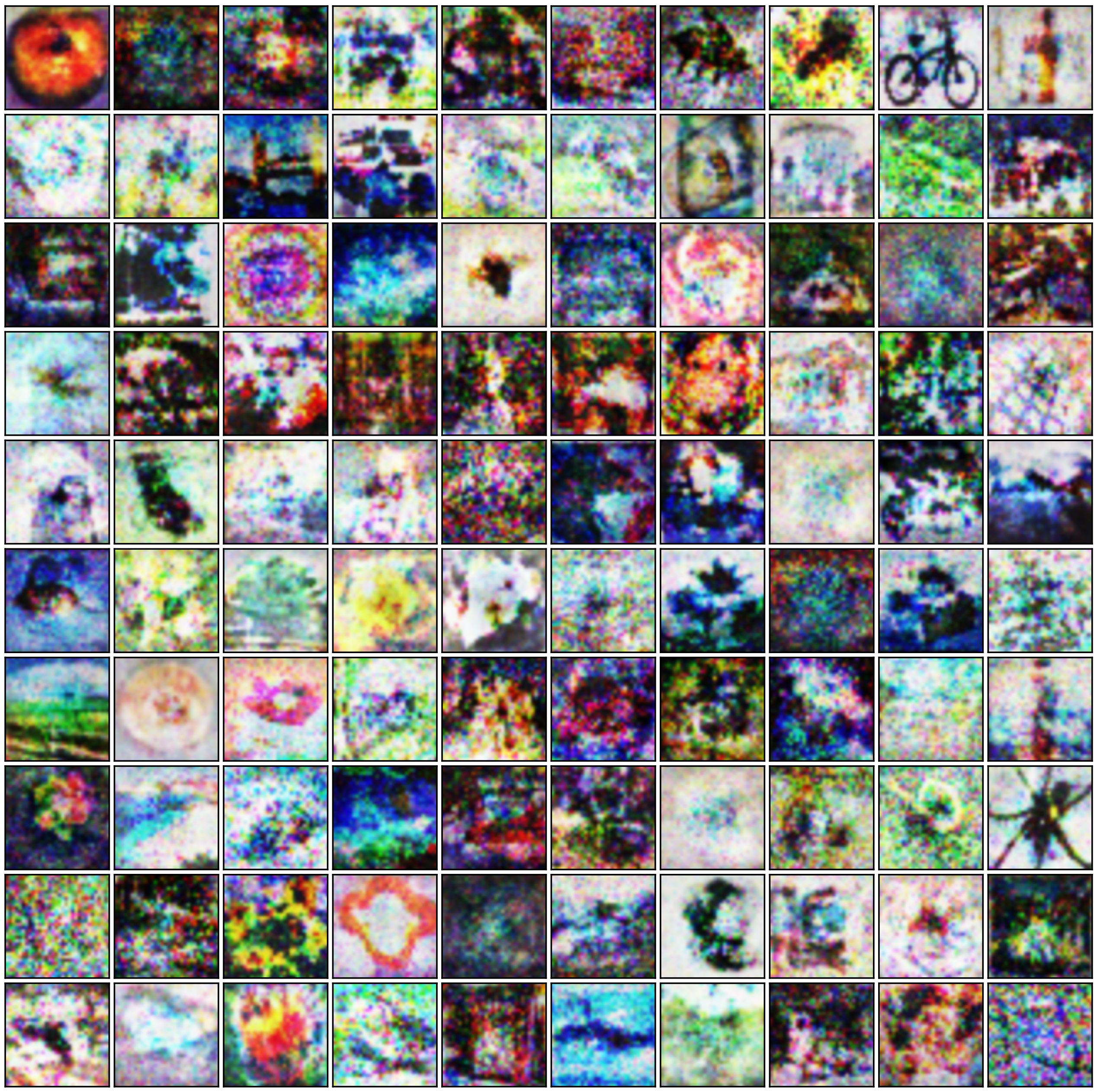

Figure 10: The complete dataset distillation for CIFAR-100. Referenced in Fig. 9.

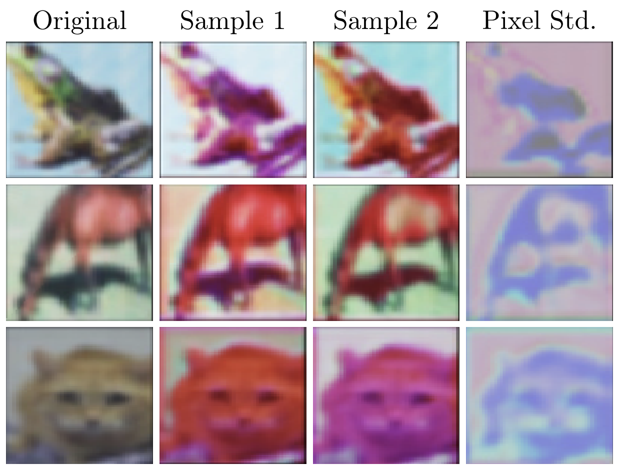

Figure 11: Learned data augmentations. The original image is on the left, followed by two augmented samples and the standard deviation of the pixel intensities from the augmentation distribution.