Fitting Millions of Hyperparameters using the Implicit Function Theorem

Jonathan Lorraine, Haoping Xu, Paul Vicol, Roger Grosse, David Duvenaud

November 5, 2019

University of Toronto, Vector Institute, ElementAI

Overview

Introduction

Goal:

- Gradient-based hyperparameter optimization, which scales to problems with as many - or more - hyperparameters than parameters.

Why?

- Generalization of neural networks is critically tied to hyperparameters.
- Existing methods have various limitations - particularly for high-dimensional hyperparameters.
- Interesting new techniques if we aren't constrained to have low-dimensional hyperparameters!

Learned Data Augmentation

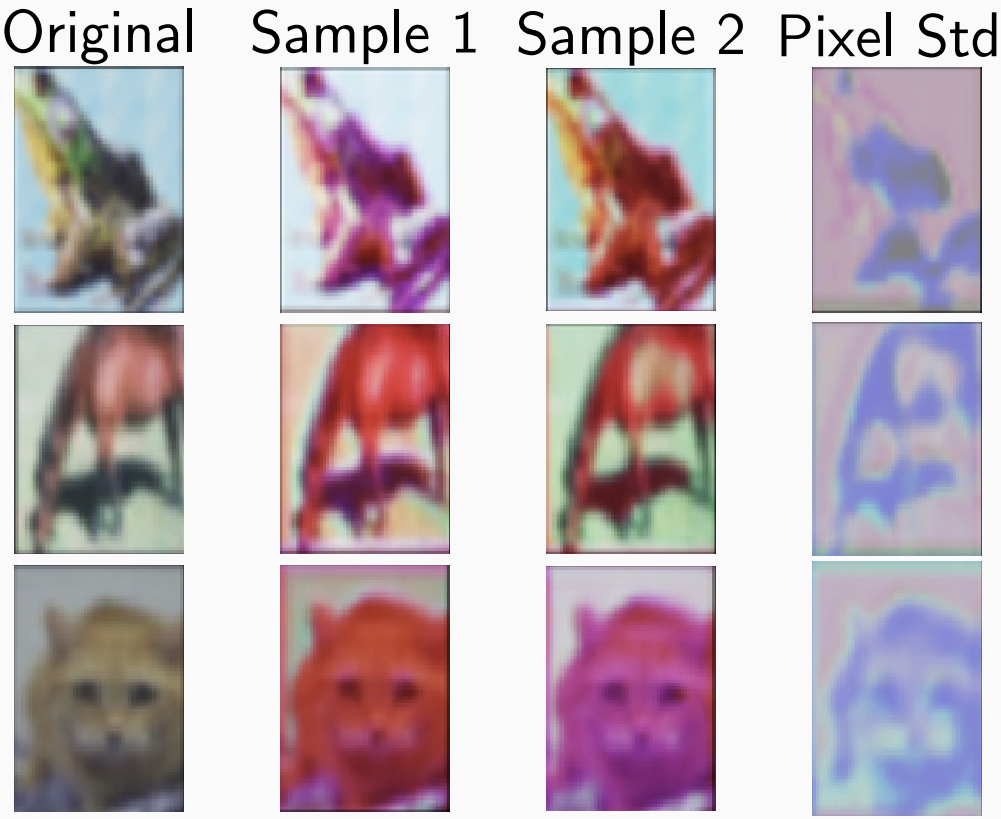


Figure 1: A visualization of the learned data augmentation. The original image is on the left, followed by samples and the standard deviation of the pixel intensities from the augmentation distribution.

The hyperparameters are weights in a U-Net [17], which learns a stochastic data augmentation: $\mathbf{x}' = U_{\boldsymbol{\lambda}}(\mathbf{x}, \epsilon), \epsilon \sim \mathcal{N}(0, I), \mathbf{x} \sim \mathcal{D}$.

Contributions

- We show how implicit differentiation is the limit of differentiating through optimization.
- We scale implicit differentiation to optimize hyperparameters of modern, deep nets with computationally feasible approximations to the inverse-Hessian of the training loss.
- We present a simple algorithm with few (hyper-hyper-)parameters, which scales to millions of hyperparameters with a similar cost to evaluating training gradients.

Hyperparameter Optimization is Nested Optimization

- $\mathcal{L}_T$ is training loss.
- $\mathcal{L}_V$ is validation loss.
- $\mathbf{w}$ are parameters.
- $\boldsymbol{\lambda}$ are hyperparameters.
- $\mathbf{w}^*(\boldsymbol{\lambda})$ are best parameters on train loss given hyperparameters:

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

- Want to optimize validation loss using optimal parameters:

$$\mathcal{L}_V^*(\boldsymbol{\lambda}) := \mathcal{L}_V(\mathbf{w}^*(\boldsymbol{\lambda}))$$

Hypergradients are indirect

- The gradient is difficult to compute because we need the **Jacobian of the best-response**, which requires differentiating through optimization:

$$\underbrace{\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \frac{\partial \mathcal{L}_V(\mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}^*(\boldsymbol{\lambda})} \frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}$$

Hypergradients are indirect

- The gradient is difficult to compute because we need the **Jacobian of the best-response**, which requires differentiating through optimization:

$$\underbrace{\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \frac{\partial \mathcal{L}_V(\mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}^*(\boldsymbol{\lambda})} \frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}$$

$$\frac{\partial \mathcal{L}_V^*}{\partial \boldsymbol{\lambda}} = \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}$$

Theorem (Implicit Function Theorem)

If $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}',\mathbf{w}'} = 0$ *for some* $(\boldsymbol{\lambda}', \mathbf{w}')$ *and regularity conditions are satisfied, then surrounding* $(\boldsymbol{\lambda}', \mathbf{w}')$ *there exists a function* $\mathbf{w}^*(\boldsymbol{\lambda})$ *s.t.* $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda},\mathbf{w}^*(\boldsymbol{\lambda})} = 0$ *and*

$$\frac{\partial \boldsymbol{w}^*}{\partial \boldsymbol{\lambda}} = -\left[\frac{\partial^2 \mathcal{L}_T}{\partial \boldsymbol{w} \partial \boldsymbol{w}^T}\right]^{-1} \frac{\partial^2 \mathcal{L}_T}{\partial \boldsymbol{w} \partial \boldsymbol{\lambda}^T}\Big|_{\boldsymbol{\lambda},\boldsymbol{w}^*(\boldsymbol{\lambda})}$$

- Idea used in "Gradient-based Hyperparameter Optimization", Bengio, 2000.
- But inverting $\frac{\partial^2 \mathcal{L}_\mathsf{T}}{\partial \mathbf{w} \partial \mathbf{w}^T}$ costs $\mathcal{O}(D^3)$ where D is number of parameters, so limited to toy problems.
- We scale to millions of parameters by approximating inverse Hessian product with efficient Jacobian-vector products.

Catches

- The IFT only holds in a neighborhood around a locally optimal $\mathbf{w}^*(\boldsymbol{\lambda})$.
- But, isn't the problem that it's difficult to evaluate $\mathbf{w}^*(\boldsymbol{\lambda})$, so we can apply the IFT?
- Not really - we can evaluate $\mathbf{w}^*(\boldsymbol{\lambda}')$ for some fixed $\boldsymbol{\lambda}'$ by doing gradient descent on $\mathbf{w}$.
- What's actually difficult is finding the best-response Jacobian at some fixed $\boldsymbol{\lambda}'$: $\left.\frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}'}$.

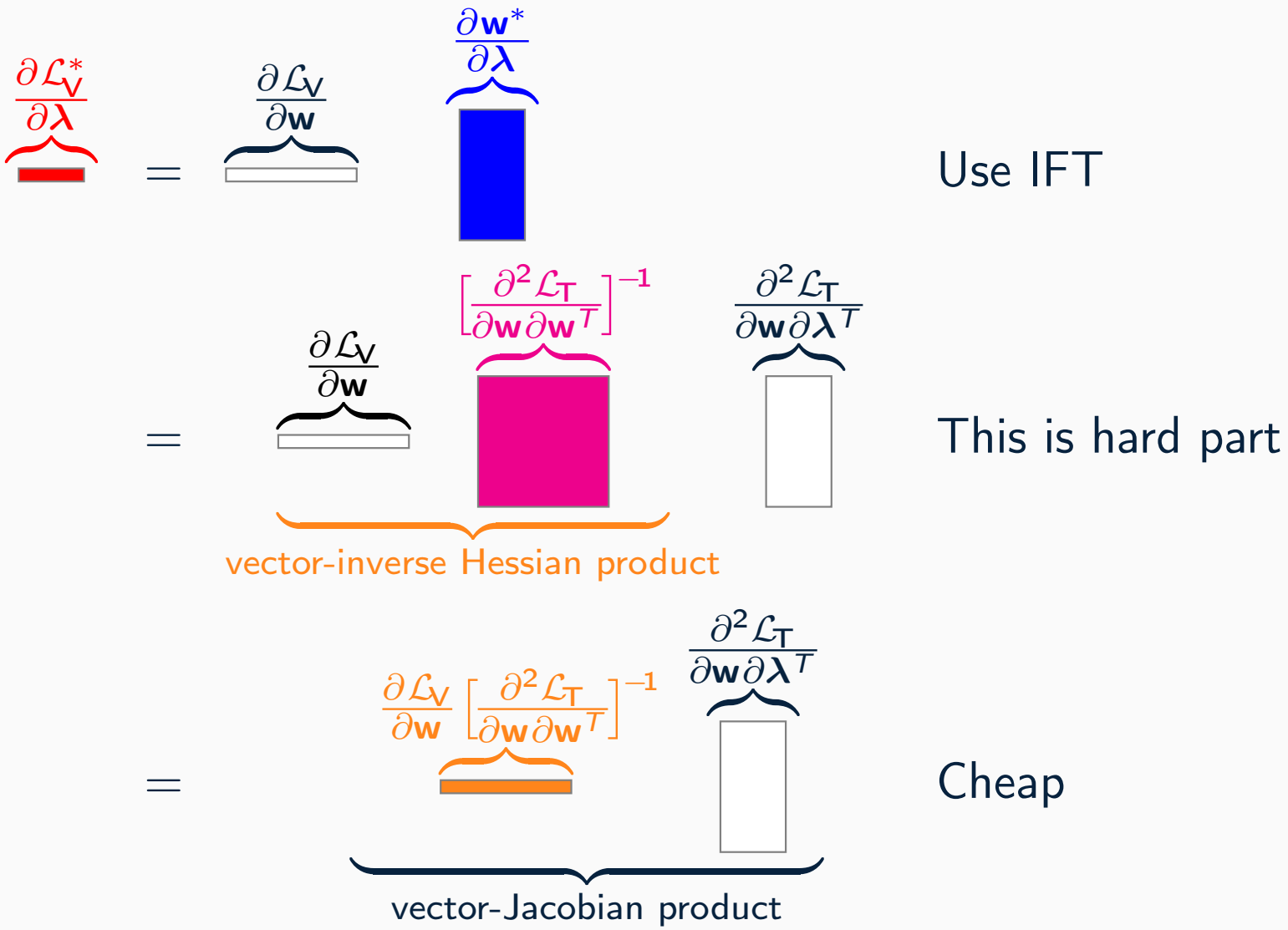
Catches

- In-practice $\|\mathbf{w}^*(\boldsymbol{\lambda}) - \widehat{\mathbf{w}^*}(\boldsymbol{\lambda})\| < \epsilon$, where $\widehat{\mathbf{w}^*}(\boldsymbol{\lambda}) = \text{SGD}(\mathbf{w}_0, \boldsymbol{\lambda})$.
- Check out HOAG [16] for results about consequences of this sub-optimality (i.e., $\mathbf{w}^*(\boldsymbol{\lambda}) \neq \widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$).
- If we know Lipschitz constants of the various functions (we don't), we can bound the error introduced by approximations.
- If $\mathbf{w}^*(\boldsymbol{\lambda}) \neq \widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$, we have an additional term for how the inverse-training-Hessian changes as the hyperparameters change: $\frac{\partial \left[\frac{\partial^2 \mathcal{L}_\text{T}}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1}}{\partial \boldsymbol{\lambda}} \frac{\partial \mathcal{L}_\text{T}}{\partial \mathbf{w}}$.
- If $\mathbf{w}^*(\boldsymbol{\lambda}) = \widehat{\mathbf{w}^*}(\boldsymbol{\lambda})$, then $\frac{\partial \left[\frac{\partial^2 \mathcal{L}_\text{T}}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1}}{\partial \boldsymbol{\lambda}} \frac{\partial \mathcal{L}_\text{T}}{\partial \mathbf{w}} = 0$, since $\frac{\partial \mathcal{L}_\text{T}}{\partial \mathbf{w}} = 0$ by optimality conditions.

Catches

- Also, we invert a large matrix.
- We use tractable inverse approximations consisting of only ***vector-Jacobian products***.
- In-practice $\| \left[\frac{\partial^2 \mathcal{L}_\mathsf{T}}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1} - \widehat{\left[\frac{\partial^2 \mathcal{L}_\mathsf{T}}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1}} \| < \epsilon$.
- Here, $\widehat{\left[\frac{\partial^2 \mathcal{L}_\mathsf{T}}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1}}$ is the result of some approximation like using CG or a Neumann series.
- Again, check out HOAG [16] for results about potential consequences of this.

$$\left. \frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}} \right|_{\boldsymbol{\lambda}'} = - \Big[\underbrace{\frac{\partial^2 \mathcal{L}_\mathsf{T}}{\partial \mathbf{w} \partial \mathbf{w}^T}}_{\text{training Hessian}} \Big]^{-1} \underbrace{\frac{\partial^2 \mathcal{L}_\mathsf{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}}_{\text{training mixed partials}} \Bigg|_{\boldsymbol{\lambda}', \mathbf{w}^*(\boldsymbol{\lambda}')} \qquad \text{(IFT)}$$


$\frac{\partial \mathcal{L}_V^*}{\partial \boldsymbol{\lambda}}$
$=$
$\frac{\partial \mathcal{L}_V}{\partial \mathbf{w}}$
$\frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}$
Use IFT
$=$
$\frac{\partial \mathcal{L}_V}{\partial \mathbf{w}}$
$\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1}$
$\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}$
This is hard part
vector-inverse Hessian product
$=$
$\frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1}$
$\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}$
Cheap
vector-Jacobian product

How can we efficiently approximate $v = \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1}$?

How can we efficiently approximate $v = \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1}$?

Initial ideas:

- Solve for v with conjugate gradient (CG) - ex., Pedregosa [16].

How can we efficiently approximate $v = \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1}$?

Initial ideas:

- Solve for v with conjugate gradient (CG) - ex., Pedregosa [16].
- Approximate $\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1} \approx I$ - ex., Luketina et al. [11].

How can we efficiently approximate $v = \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1}$?

Initial ideas:

- Solve for v with conjugate gradient (CG) - ex., Pedregosa [16].
- Approximate $\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1} \approx I$ - ex., Luketina et al. [11].
- Build on-line approximation of $\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1}$ with KFAC [14].

Unrolled optimization [Domke 2012, Maclaurin et al., 2015] approximates best-response function $\mathbf{w}^*(\boldsymbol{\lambda}) \approx \mathbf{w}_i(\boldsymbol{\lambda})$, where

$$\mathbf{w}_{i+1}(\boldsymbol{\lambda}) := \underbrace{\mathbf{w}_i(\boldsymbol{\lambda}) + \frac{\partial \mathcal{L}_\mathsf{T}(\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda}))}{\partial \mathbf{w}}}_{\text{SGD Step}}$$

Unrolled optimization [Domke 2012, Maclaurin et al., 2015] approximates best-response function $\mathbf{w}^*(\boldsymbol{\lambda}) \approx \mathbf{w}_i(\boldsymbol{\lambda})$, where

$$\mathbf{w}_{i+1}(\boldsymbol{\lambda}) := \underbrace{\mathbf{w}_i(\boldsymbol{\lambda}) + \frac{\partial \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda}))}{\partial \mathbf{w}}}_{\text{SGD Step}}$$

If $\mathbf{w}_i \xrightarrow{i \to \infty} \mathbf{w}^*$, then $\frac{\partial \mathbf{w}_i}{\partial \boldsymbol{\lambda}} \xrightarrow{i \to \infty} \frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}$ (with some assumptions).

An Interesting Connection

- Note that $\frac{\partial \mathbf{w}_1}{\partial \boldsymbol{\lambda}} = I \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}|_{\boldsymbol{\lambda}, \mathbf{w}_0}$.
- This is (almost) the same gradient as using implicit differentiation with $\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1} \approx I$!
- If we unrolled from a locally optimal $\mathbf{w}^*(\boldsymbol{\lambda})$ instead of $\mathbf{w}_0$, they would be exactly the same.

- Idea: $\frac{\partial \mathbf{w}_i}{\partial \boldsymbol{\lambda}}$ is the same as using implicit differentiation with an approximation of $\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1}$, which is exact as $i \to \infty$ (if we unroll from a locally optimal $\mathbf{w}^*(\boldsymbol{\lambda})$).
- Consequence: Unify unrolled optimization and implicit differentiation.

Unifying Unrolled Optimization and Implicit Differentiation

Lemma (2)

Given the recurrence from unrolling SGD optimization ($\mathbf{w}_0$ given, $\mathbf{w}_{i+1}(\boldsymbol{\lambda}) = \mathbf{w}_i(\boldsymbol{\lambda}) + \frac{\partial \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda}))}{\partial \mathbf{w}}$), *we have:*

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

This gives an approximate best-response Jacobian provided by unrolling for i steps.

Ugly, but will show it massively simplifies if we unroll from $\mathbf{w}^*(\boldsymbol{\lambda})$ as initial condition instead of an arbitrary $\mathbf{w}_0$.

Aside

- Why did we choose to telescope the recurrence for $\mathbf{w}_i$ and group the terms like this?
- Because we want to use properties of Neumann Series converging to inverses.
- If T is contractive:

$$(Id - T)^{-1} = \sum_{i=0}^{\infty} T^k \tag{2}$$

- This is just a generalization of $\frac{1}{1-x} = \sum_{i=0}^{\infty} x^i$.
- Can generalize $\mathbf{w}_i(\boldsymbol{\lambda}) + \frac{\partial \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda}))}{\partial \mathbf{w}}$ to $\mathtt{opt}(\boldsymbol{\lambda}, \mathbf{w}_i(\boldsymbol{\lambda}))$ for optimizers $\mathtt{opt}$ besides SGD, where $\frac{\partial \mathtt{opt}}{\partial \boldsymbol{\lambda}} = T$.

Unification

Theorem

Given the SGD recurrence, if $\mathbf{w}_0 = \mathbf{w}^*(\boldsymbol{\lambda})$*:*

$$\frac{\partial \mathbf{w}_{i+1}}{\partial \boldsymbol{\lambda}} = \left(\sum_{j<i} \left[I + \frac{\partial^2 \mathcal{L}_\mathsf{T}}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^j \right) \frac{\partial^2 \mathcal{L}_\mathsf{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T} \Bigg|_{\mathbf{w}^*(\boldsymbol{\lambda})} \tag{3}$$

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

$$\lim_{i \to \infty} \frac{\partial \mathbf{w}_{i+1}}{\partial \boldsymbol{\lambda}} = - \left[\frac{\partial^2 \mathcal{L}_\mathsf{T}}{\partial \mathbf{w} \partial \mathbf{w}^T} \right]^{-1} \frac{\partial^2 \mathcal{L}_\mathsf{T}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T} \Bigg|_{\mathbf{w}^*(\boldsymbol{\lambda})} \tag{4}$$

Eq. 3 allows us to approximate the inverse-Hessian-vector product with a repeated vector-Hessian product by unrolling from $\mathbf{w}^*(\boldsymbol{\lambda})$.

Neumann Inverse Approximation

- Old trick: $(I - T)^{-1} = \sum_{j=0}^{\infty} T^j$ if T contractive.
- : So,

$$\left[\frac{\partial^2 \mathcal{L}_{\mathsf{T}}}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1} = \sum_{j=0}^{\infty} \left[I + \frac{\partial^2 \mathcal{L}_{\mathsf{T}}}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^j$$

$$\frac{\partial \mathcal{L}_{\mathsf{V}}}{\partial \mathbf{w}} \left[\frac{\partial^2 \mathcal{L}_{\mathsf{T}}}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1} = \frac{\partial \mathcal{L}_{\mathsf{V}}}{\partial \mathbf{w}} \sum_{j=0}^{\infty} \left[I + \frac{\partial^2 \mathcal{L}_{\mathsf{T}}}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^j$$

$$= \sum_{j=0}^{\infty} \left[\frac{\partial \mathcal{L}_{\mathsf{V}}}{\partial \mathbf{w}} + \frac{\partial \mathcal{L}_{\mathsf{V}}}{\partial \mathbf{w}} \frac{\partial^2 \mathcal{L}_{\mathsf{T}}}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^j$$

- We can compute each term using only vector-Jacobian products! Thus, cost per term is a constant multiple of evaluating training loss.

Algorithm 1 `approxInverseJVP`$(\mathbf{v}, \mathbf{f})$: Neumann Series approximation of $\mathbf{v}\,[\frac{\partial \mathbf{f}}{\partial \mathbf{w}}]^{-1}$

1: Initialize $\mathbf{p} = \mathbf{v}$
2: **for** $j = 0 \ldots i$ **do**
3: $\quad \mathbf{v} \mathrel{+}= \texttt{grad}(\mathbf{f}, \mathbf{w}, \texttt{grad\_outputs} = \mathbf{v})$ ▷ Here, $\mathbf{f} = \frac{\partial \mathcal{L}_\mathsf{T}}{\partial \mathbf{w}}$
4: $\quad \mathbf{p} \mathrel{+}= \mathbf{v}$
5: **return p**

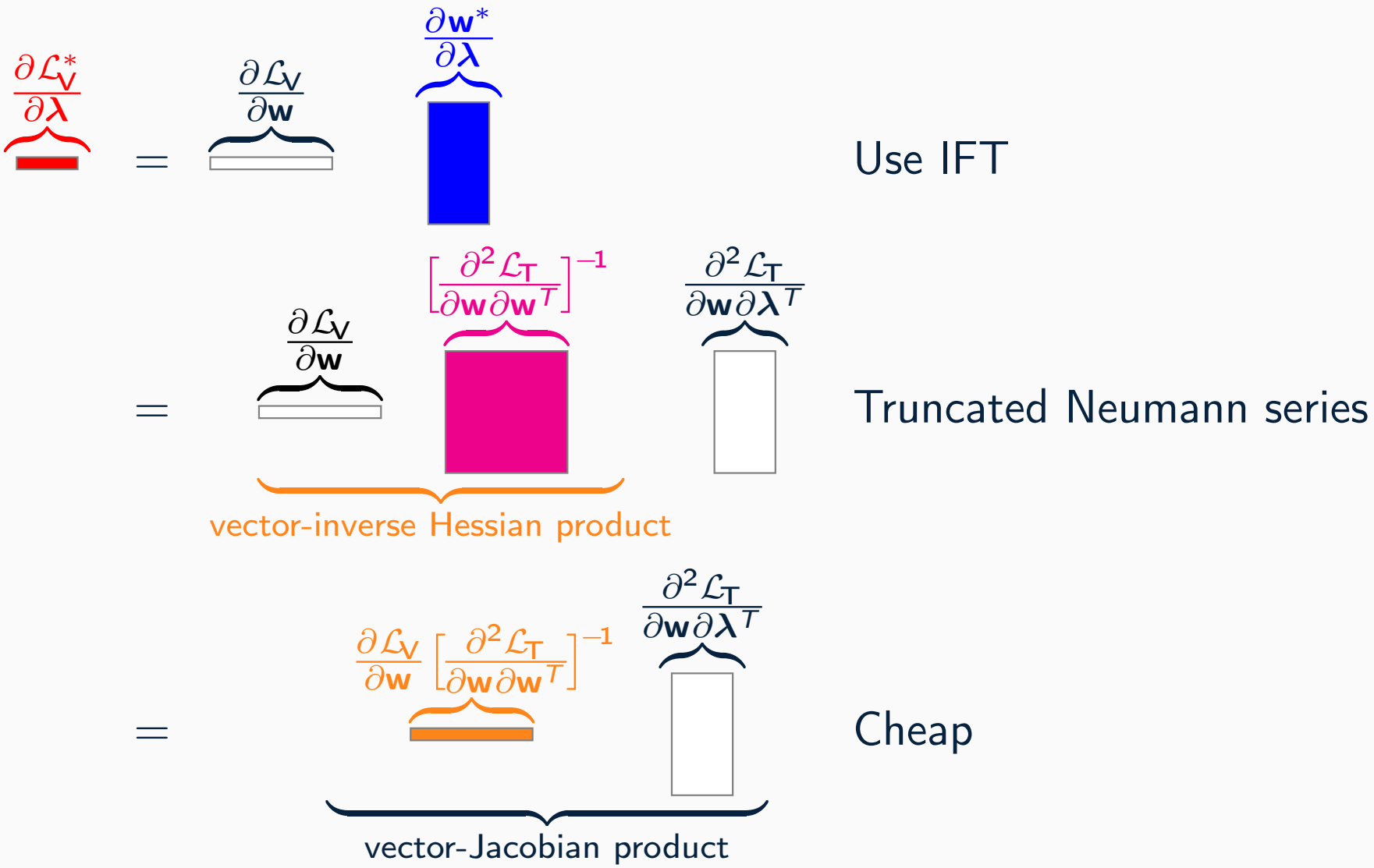
$\texttt{grad}(\mathbf{f}, \mathbf{w}, \texttt{grad\_outputs} = \mathbf{v})$ is PyTorch notation to compute the Jacobian-vector product $\frac{\partial \mathbf{f}}{\partial \mathbf{w}}\mathbf{v}$

Algorithm 2 Gradient-based parameter & hyperparam optimization

while not converged **do**
  for $k = 1 \dots N$ **do** ▷ ex., $N = 5$
    $\mathbf{w} \mathrel{-}= \frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}, \mathbf{w}}$
  $\boldsymbol{\lambda} \mathrel{-}= \texttt{IFT\_hypergradient}(\mathcal{L}_V, \mathcal{L}_T, \boldsymbol{\lambda}, \mathbf{w})$
return $\boldsymbol{\lambda}, \mathbf{w}$

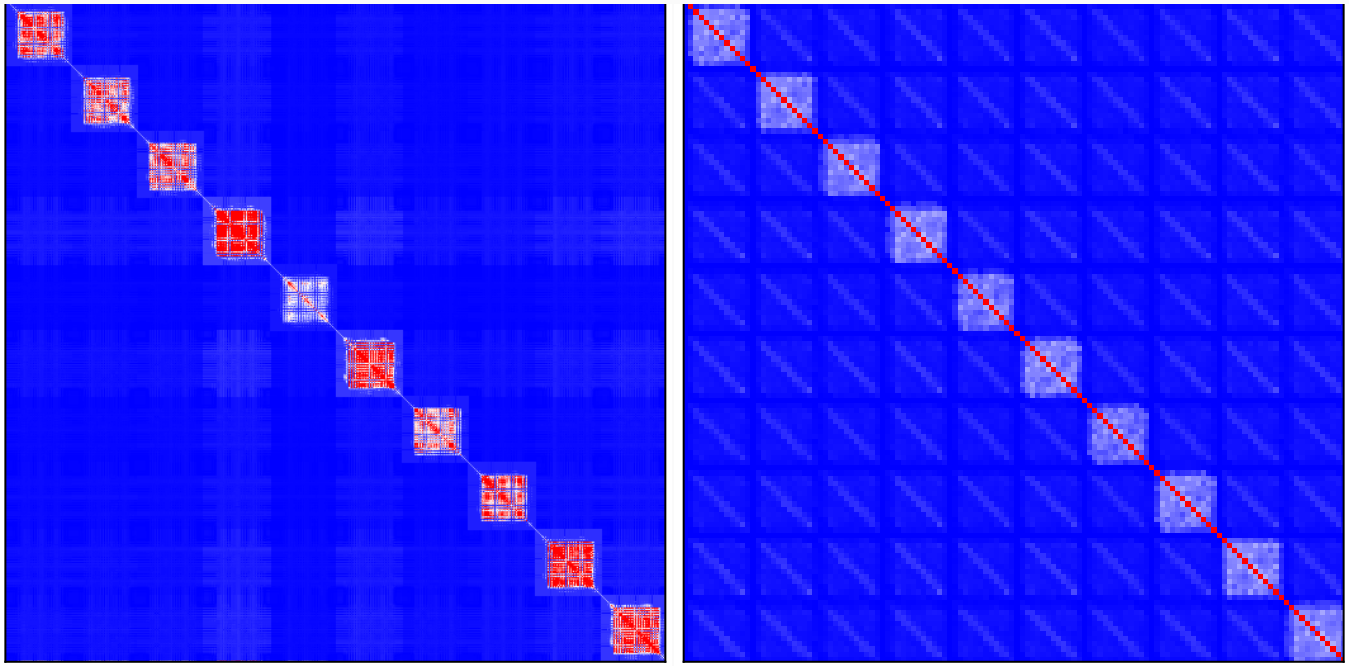
Algorithm 3 $\texttt{IFT\_hypergradient}(\mathcal{L}_V, \mathcal{L}_T, \boldsymbol{\lambda}', \mathbf{w}')$

$\mathbf{v}_1 = \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}', \mathbf{w}'}$
$\mathbf{v}_2 = \texttt{approxInverseJVP}(\mathbf{v}_1, \frac{\partial \mathcal{L}_T}{\partial \mathbf{w}})$
return $\texttt{grad}(\frac{\partial \mathcal{L}_T}{\partial \boldsymbol{\lambda}}, \mathbf{w}, \texttt{grad\_outputs} = \mathbf{v}_2)$ ▷ Approximates $\frac{\partial \mathcal{L}_V^*}{\partial \boldsymbol{\lambda}}$


$\frac{\partial \mathcal{L}_V^*}{\partial \boldsymbol{\lambda}}$
$=$
$\frac{\partial \mathcal{L}_V}{\partial \mathbf{w}}$
$\frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}$
Use IFT
$=$
$\frac{\partial \mathcal{L}_V}{\partial \mathbf{w}}$
$\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1}$
$\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}$
Truncated Neumann series
vector-inverse Hessian product
$=$
$\frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1}$
$\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}$
Cheap
vector-Jacobian product

Related algorithms

- Changing Alg. 1 by setting $i = 0$ recovers $T1 - T2$ [11].
- Changing Alg. 1 to use conjugate gradient is used in Bengio [2] and *HOAG* [16].
- Can change Alg. 1 to use KFAC [14].



Hessian Inverse Hessian

Figure 2: 2nd order info for logistic regression on MNIST. The 7850×7850 images have been down-sampled & clipped for visualization. Red & blue indicates a large & small magnitudes respectively.

Comparison

Method	Steps	Best-Response Jacobian Approximation
Exact IFT	∞	$\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1} \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\vert_{\mathbf{w}^*(\boldsymbol{\lambda})}$
Bengio [2], [9]	∞	$\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1} \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\vert_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
$T1 - T2$ [11]	1	$[I]^{-1} \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\vert_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Unrolled Diff. [3, 13]	i	$\sum_{j \leq i} \left[\prod_{k<j} I + \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\Big\vert_{\mathbf{w}_{i-k}}\right] \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\vert_{\mathbf{w}_{i-j}}$
Us - Hessian	i	$\left(\sum_{j<i} \left[I + \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^j\right) \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\vert_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Us - Gauss-Newton	i	$\left(\sum_{j<i} \left[I + \frac{\partial \mathcal{L}_T}{\partial \mathbf{w}} \frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}^T\right]^j\right) \frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^T}\Big\vert_{\widehat{\mathbf{w}^*}(\boldsymbol{\lambda})}$
Hypernetwork [10, 12]	-	$\frac{\partial \mathbf{w}_\phi^*}{\partial \boldsymbol{\lambda}}$ where $\mathbf{w}_\phi^*(\boldsymbol{\lambda}) = \arg\min_\phi \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w}_\phi(\boldsymbol{\lambda}))$

Experiments

A Sanity Check

- First, verify we can optimize the validation loss.
- We train models with many more parameters and hyperparameters than data points.
- Hopefully, we achieve near perfect training and validation performance.
- We train models with 50 train and validation datapoints on different architectures & datasets, with a separate weight decay hyperparameter for each weight.

Overfitting Small Problems

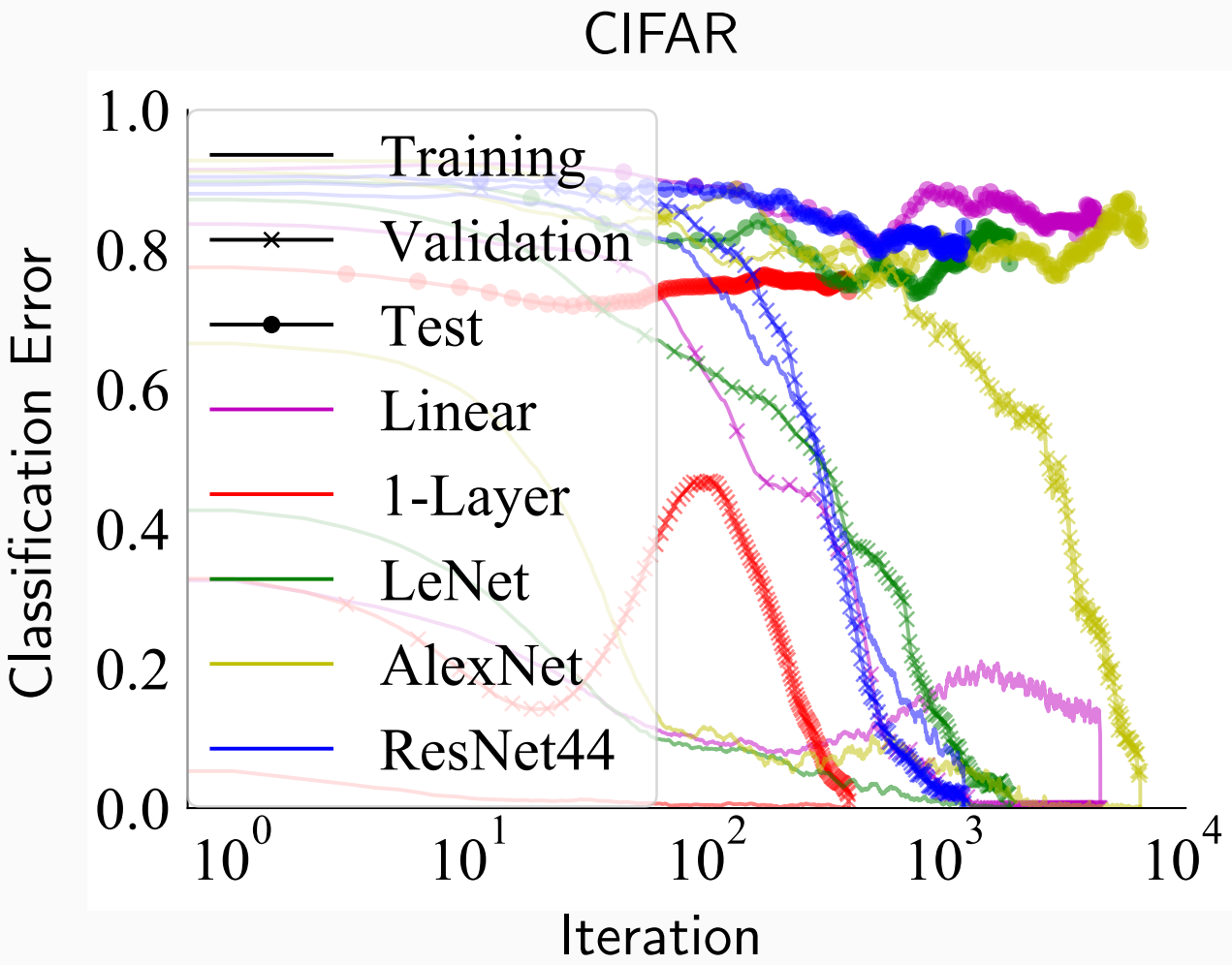


Figure 3: In all examples we are able to achieve 100 % training and validation accuracy, while the testing accuracy is significantly lower. An identity inverse approximation is used here.

Overfitting Small Problems

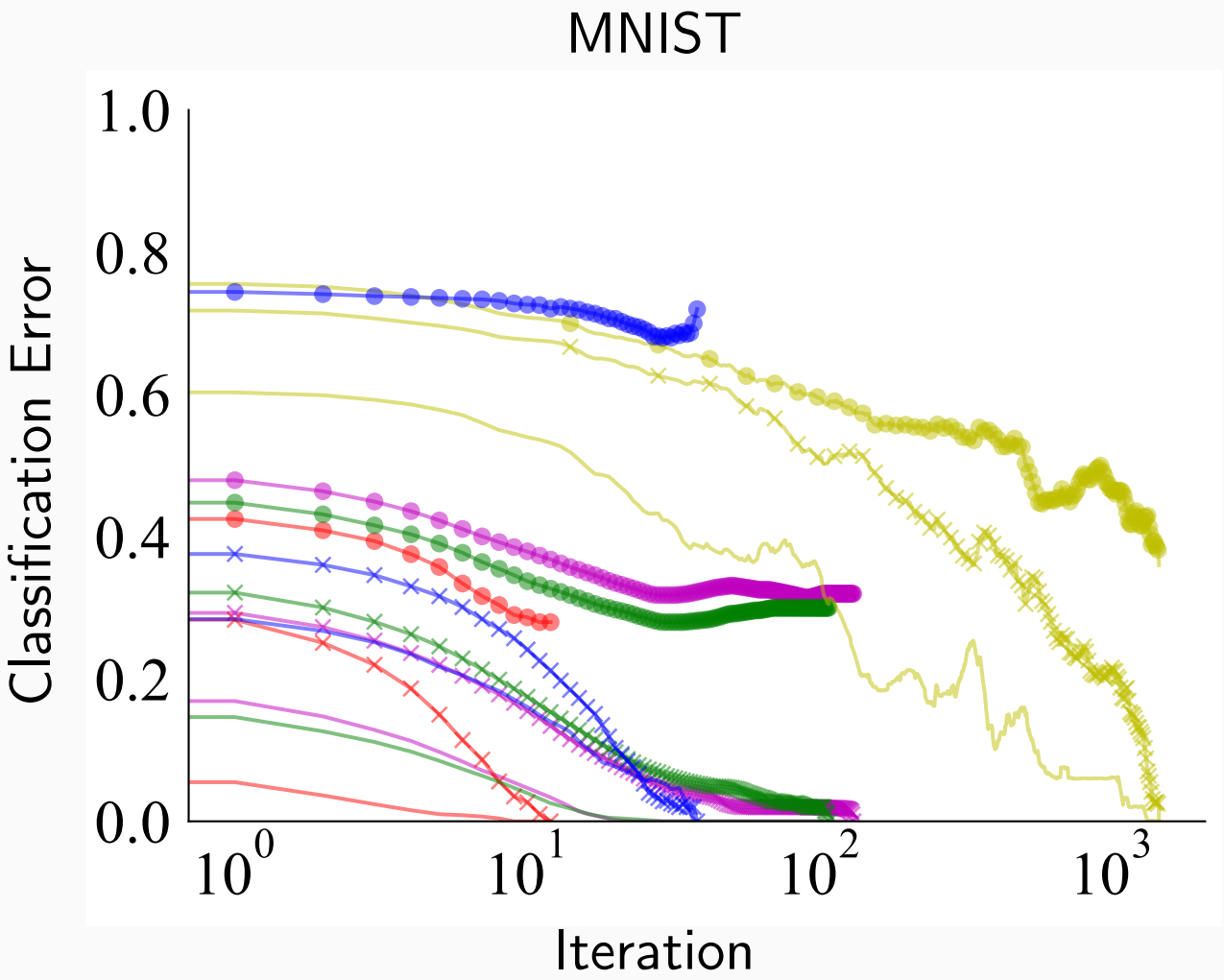


Figure 4: In all examples we are able to achieve 100 % training and validation accuracy, while the testing accuracy is significantly lower. An identity inverse approximation is used here.

Overfitting Small Problems

- Thus, we can successfully optimize high-dimensional hyperparameters of the L2 norm of each weight.
- More than $1,000,000$ hyperparameters in AlexNet [8] and ResNet [7]!
- But, our test performance is near random.
- Are there sensible ways to introduce millions of hyperparameters?
- How should training / validation / testing partitions be constructed when we have many hyperparameters?

Comparison with KFAC

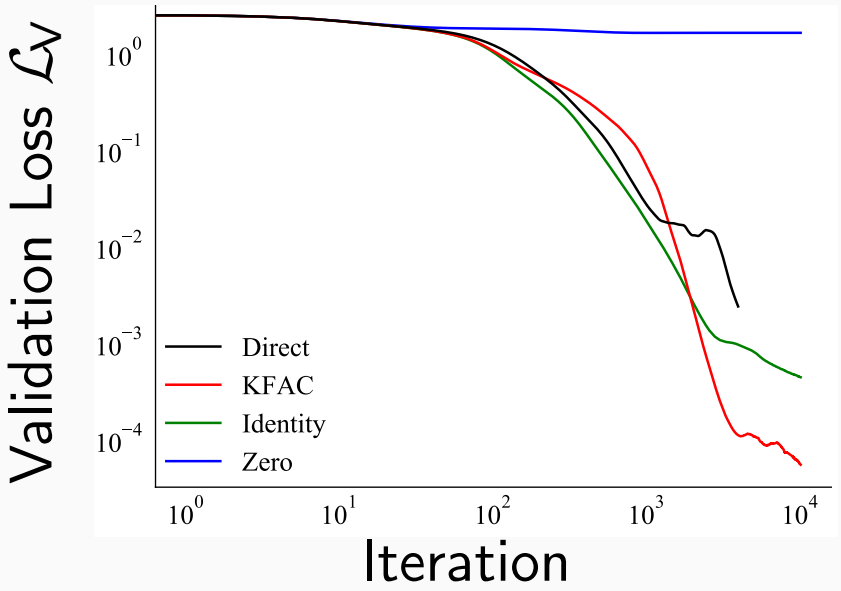


Figure 5: A comparison for different inversion approximations for our algorithm are presented. "Direct" means exact IFT. Exact inversion eventually fails from numerical errors.

Takeaway - KFAC does well on logistic regression, but ***we were unable to achieve improved performance for deep networks***.

Dataset Distillation

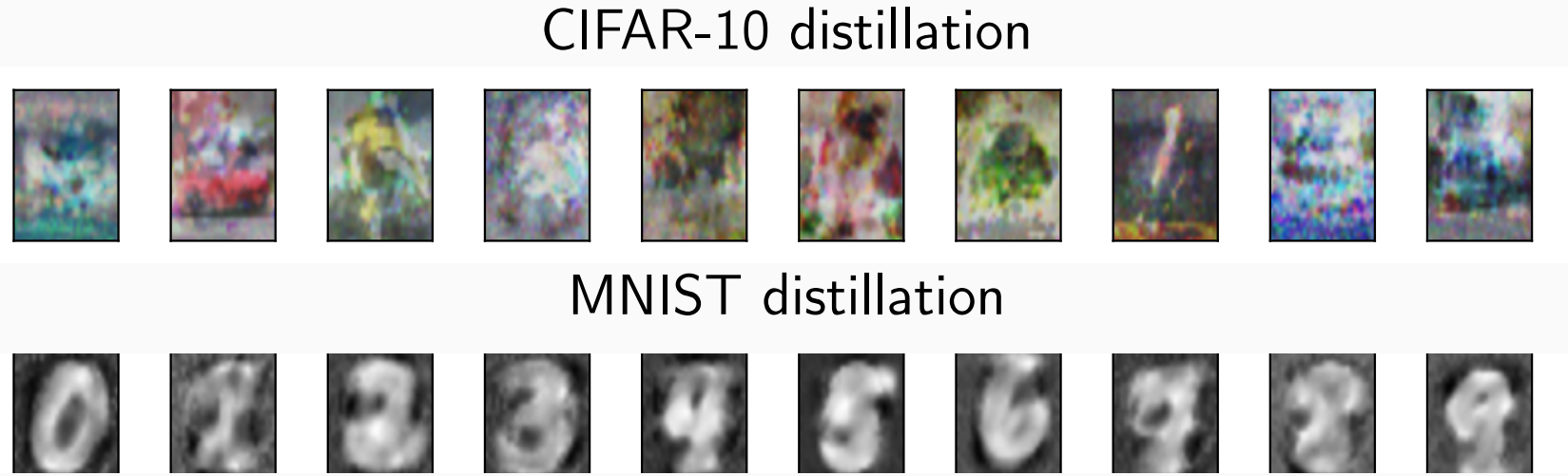
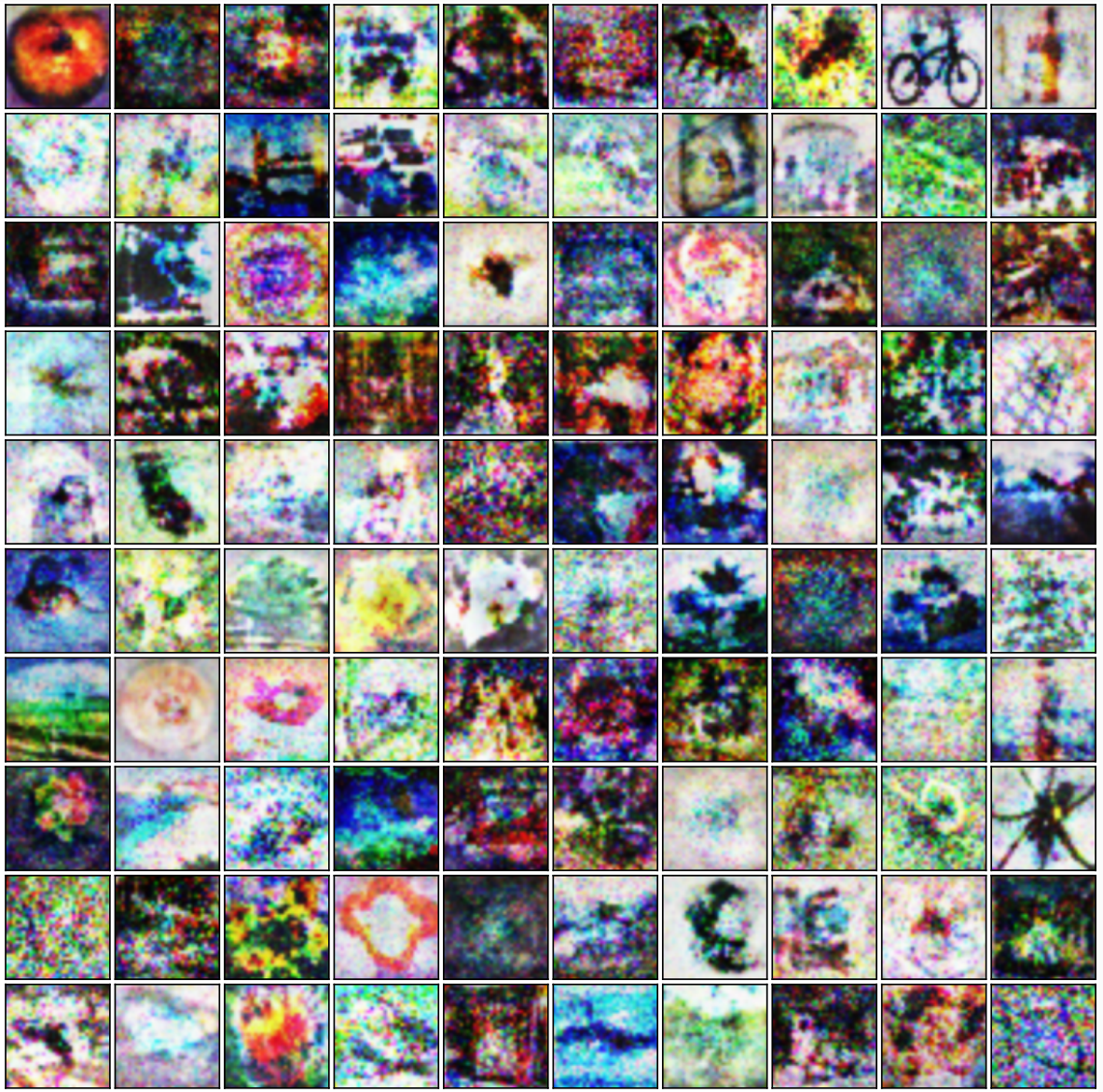


Figure 6: We learn 1 distilled image for each class, so after a logistic regression classifier has been trained on the distillation, it generalizes to the rest of the dataset. We fit 30 720 and 7840 hyperparameters for CIFAR-10 and MNIST respectively.

Dataset Distillation

CIFAR-100 distillation - 300 720 hyperparameters



Learned Augmentations

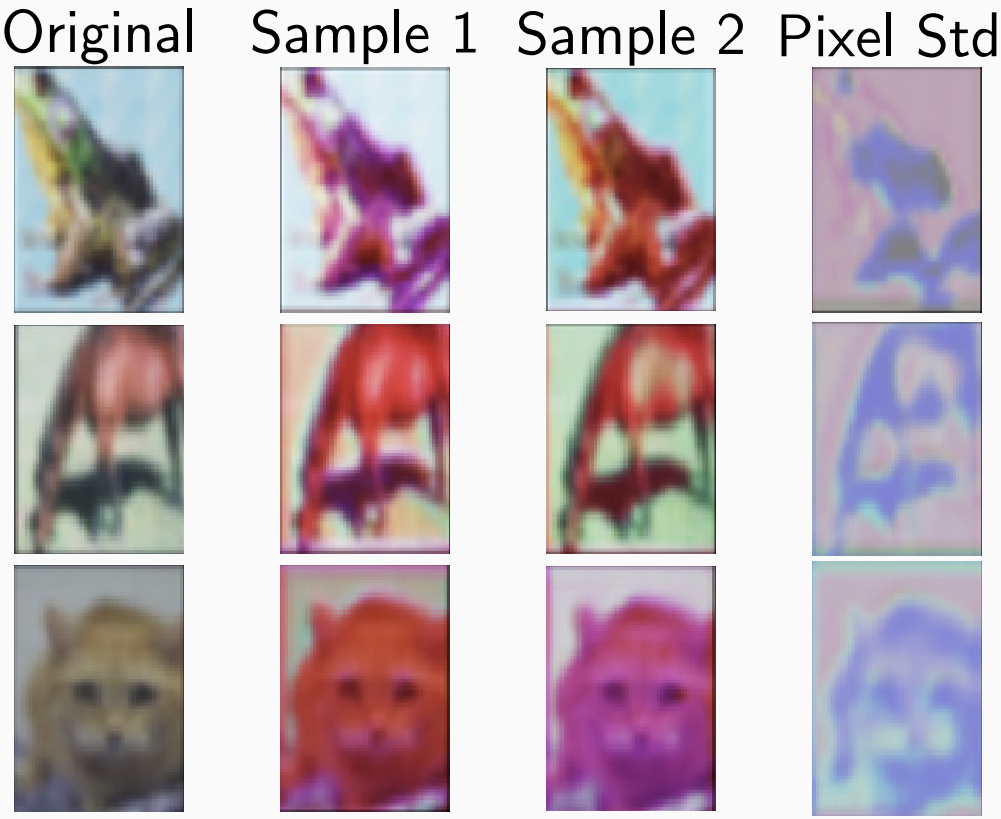


Figure 7: A visualization of the learned data augmentation. The original image is on the left, followed by samples and the standard deviation of the pixel intensities from the augmentation distribution.

The hyperparameters are weights in a U-Net [17], which learns a stochastic data augmentation: $\mathbf{x}' = U_{\boldsymbol{\lambda}}(\mathbf{x}, \epsilon), \epsilon \sim \mathcal{N}(0, I)$.

Learned Augmentation Results

	Accuracy Distribution		
Strategy	Train	Validation	Test
Standard Aug.	100 ±0.000 %	92.5 ±0.021 %	92.6 ±0.017 %
Learned Aug.	99.9 ±0.001 %	95.1 ±0.002 %	94.6 ±0.001 %

- Results are computed over 10 random seeds for a ResNet architecture.
- The baseline accuracy has a lower mean, and *much* higher variance than learned augmentations.
- Used 2-steps of Neumann series with a Gauss-Newton approximation.
- Total compute cost of learning the augmentations is about 3x baseline per iter, 2x per training run.

Limitations

- Can't use this for optimization hyperparameters
- not clear what can be done for discrete hypers.

Future Directions

- What about as an alternative for methods like LOLA [4], when learning in arbitrary Stackelberg games?
- What are interesting ways to use large number of hyperparameters?
- When does having a large number of hyperparameters just overfit the validation set?

Approximate Implicit Differentiation for GANs

- Can we use implicit differentiation to help with GAN [5] training?

$$G^* = \arg\min_G \mathcal{L}_G(G, D^*(G)), D^*(G) := \arg\min_D \mathcal{L}_D(G, D)$$

- Simultaneous SGD ignores the best-response Jacobian $\frac{\partial D^*}{\partial G}$.
- But, simple zero-sum GAN formulations where $\mathcal{L}_G = -\mathcal{L}_D$ always have a direct gradient. Maybe its fine to ignore the response?
- Metz et al. [15] use differentiation through optimization to approximate $\frac{\partial D^*}{\partial G}$.

Conclusion

- Implicit function theorem gives a family of tractable inverse approximations.
- Just using an identity approximation with efficient Jacobian-vector products works pretty well for tuning millions of hyperparameters.
- What to do with this new ability?

Thanks

Jonathan Lorraine

Haoping Xu

Paul Vicol

David Duvenaud

Roger Grosse

Extra Slides

Related Algorithms

- Try approximating $\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}\right]^{-1} = I$, which is equivalent to using only the first term from the Neumann series.
- This was proposed in $T1 - T2$ Luketina et al. [11], but they were only able to optimize 10 - 20 hyperparameters, where we are able to optimize $>1\,000\,000$ hyperparameters. It's not clear why they were unable to scale.
- Adding more terms from the Neumann series provides more accurate hypergradients.
- We can replace $\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}^T}$ with the Gauss-Newton matrix $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}^T \frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}$.
- This massively simplifies the computation because we only need to compute $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}$ once.

Related algorithms

- Changing Alg. 3 to use Hypernetworks [6] recovers self-tuning-nets [10, 12].
- Can change Alg. 3 to use Unrolled Optimization [3, 13].
- Changing Alg. 3 to use an acquisition function on a GP for tuples of $(\boldsymbol{\lambda}, \mathcal{L}_{\mathrm{V}}^{*}(\boldsymbol{\lambda}))$ recovers Bayesian Optimization-type algorithms [18].
  For Bayesian Optimization & Random Search it's common to re-initialize $\mathbf{w}$ in Alg. 2 after changing $\boldsymbol{\lambda}$, but this wastes ***a lot*** of computational effort.

Hyperparameter Optimization is a Pure-Response Game

- Simultaneous SGD uses ***only*** the direct gradient.
- This ***sometimes*** works for games with a non-zero direct gradient. Ex., zero-sum games [1].
- The direct gradient is often identically zero for hyperparameter optimization. ***This makes hyperparameter optimization uniquely hard!***
- Thus, we ***must*** approximate the indirect gradient - can't use simple choices like simultaneous SGD.

$$\underbrace{\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \overbrace{\underbrace{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}}}_{\text{hyperparameter direct grad.}}}^{\textit{Identically 0!}} + \overbrace{\underbrace{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}^*(\boldsymbol{\lambda})}}_{\text{parameter direct grad.}} \underbrace{\frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{best-response Jacobian}}}^{\text{hyperparameter indirect grad.}}$$

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

References

[1] David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. *arXiv preprint arXiv:1802.05642*, 2018.

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

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

[4] Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 122–130. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[5] 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.

[6] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[7] 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*, pages 770–778, 2016.

[8] 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.

[9] 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. IEEE, 1996.

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

[11] 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.

[12] 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 (ICLR)*,

2019.

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

[14] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[15] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

[16] Fabian Pedregosa. Hyperparameter optimization with approximate gradient. *arXiv preprint arXiv:1602.02355*, 2016.

[17] 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. Springer, 2015.

[18] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.