HYDRA: Hypergradient Data Relevance Analysis for Interpreting Deep Neural Networks

Yuanyuan Chen[1], Boyang Li[1, 2*], Han Yu[1*], Pengcheng Wu[1], and Chunyan Miao[1*]

[1]School of Computer Science and Engineering, Nanyang Technological University
[2]Alibaba-NTU Singapore Joint Research Institute
{yuanyuan.chen, boyang.li, han.yu, pengcheng.wu, ascymiao}@ntu.edu.sg
*Corresponding authors

Abstract

The behaviors of deep neural networks (DNNs) are notoriously resistant to human interpretations. In this paper, we propose Hypergradient Data Relevance Analysis, or HYDRA, which interprets the predictions made by DNNs as effects of their training data. Existing approaches generally estimate data contributions around the final model parameters and ignore how the training data shape the optimization trajectory. By unrolling the hypergradient of test loss w.r.t. the weights of training data, HYDRA assesses the contribution of training data toward test data points throughout the training trajectory. In order to accelerate computation, we remove the Hessian from the calculation and prove that, under moderate conditions, the approximation error is bounded. Corroborating this theoretical claim, empirical results indicate the error is indeed small. In addition, we quantitatively demonstrate that HYDRA outperforms influence functions in accurately estimating data contribution and detecting noisy data labels. The source code are available at https://github.com/cyyever/aaai_hydra_8686.

Introduction

What makes neural networks do exactly what they do? This is a crucial question that lingers in the minds of machine learning researchers and practitioners. The underpinnings of deep neural networks (DNNs), including non-convex objective functions, stochastic optimization, and overparameterization, may implicitly regularize the network and improve generalization (e.g., Li and Liang 2018; Arora, Cohen, and Hazan 2018), but also hinder the interpretation of the network behaviors and the training process. The black-box nature of DNNs can render their predictions untrustworthy in the eyes of the general public and prohibit wide adoption. Conversely, good interpretability can enhance the training, debugging, and auditing of DNNs.

The focus of the current paper is to understand DNNs by attributing their predictions to the training data. That is, how training data influence network predictions on test data. However, as the training data are involved in the entirety of the complex training process, accurately capturing their influence on the final network is a challenge.
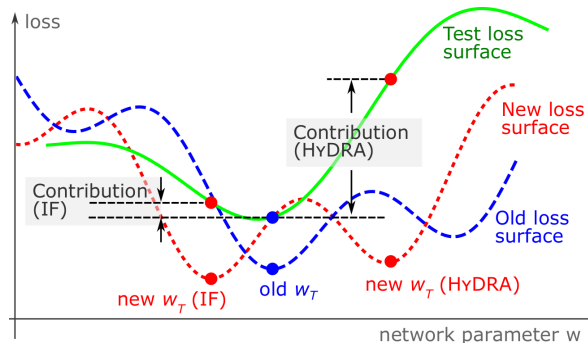
Figure 1: An illustration of HYDRA and influence functions (IF). After the removal of some training data points, the training loss shifts from the blue curve to the red curve. IF estimates the contribution within the convex region near the old w_T. HYDRA tracks the influence of data removal along the entire optimization process, possibly leading to a different local optimum.

The pioneering work of Koh and Liang (2017) proposes influence functions (IF), which measure the contribution of a training sample z_i to a test sample z^{test} as the change in the loss of z^{test} when the weight of training data z_i is changed marginally. This technique enables a number of applications (e.g. Alaa and van der Schaar 2020), but suffers from two drawbacks. First, it only measures the contribution around the final model parameters w_T and ignores the possibility that change in the weights of training data may lead to a local optimum different from w_T with a drastically different test loss. Second, IF relies on the inverse Hessian on the whole training data, which is computationally expensive to approximate. Hara, Nitanda, and Maehara (2019) analyze the entire trajectory but still rely on Hessians.

In this paper, we propose Hypergradient for Data Relevance Analysis (HYDRA) to address the aforementioned shortcomings. By unrolling the gradient of test loss w.r.t. the data weights through all training steps, HYDRA accounts for how changes in data weights shape the whole training process. As the optimization utilizes the gradient on the same data repeatedly, even marginal changes in the data weights can accumulate and eventually shift the point of convergence. Figure 1 illustrates such a scenario where

the local analysis of IF and the whole-of-trajectory analysis of HYDRA find separate local minima and contribution values. Further, to simplify computation, we propose an approximation method that eliminates the cumbersome Hessian or its inverse and establish an analytical upper bound on the approximation error.

Empirically, we verify that the approximation indeed results in small error and is closer to the whole-of-trajectory Hessian-aware measurements than IF. The approximation method of HYDRA achieves high correlation with the Hessian-aware measurements, whereas the local analysis of IF adds 11% to 16% to the approximation error by ignoring the optimization trajectory. In terms of computational time, the removal of Hessian reduces wall-clock running time by a factor of 22.0 on 4 TitanX Pascal GPUs. We also compare HYDRA and IF in their ability to identify data points with erroneous labels and find HYDRA to provide superior detection performance.

HYDRA: Hypergradient for Data Relevance Analysis

We first introduce some preliminaries. We aim to learn a function $f_{\boldsymbol{w}}(\cdot) : \mathcal{X} \to \mathcal{Y}$ parameterized by $\boldsymbol{w}$. The training dataset contains N data points and is denoted as $\mathcal{D}_{\text{train}} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \in \mathcal{X}$ and $\boldsymbol{y}_i \in \mathcal{Y}$ for all i. The function $f_{\boldsymbol{w}}$ is learned by minimizing the empirical risk $\mathcal{L}_{\text{train}}^{\text{er}}$ and the regularizer $R(\boldsymbol{w})$.

$$\mathcal{L}_{\text{train}}(\boldsymbol{w}) = \mathcal{L}_{\text{train}}^{\text{er}}(\boldsymbol{w}) + \lambda R(\boldsymbol{w}), \tag{1}$$

$$\mathcal{L}_{\text{train}}^{\text{er}}(\boldsymbol{w}) = \sum_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}_{\text{train}}} \left(1/N + \epsilon_i\right) \ell(\boldsymbol{x}_i, \boldsymbol{w}, \boldsymbol{y}_i), \tag{2}$$

where $\ell(\boldsymbol{x}_i, \boldsymbol{w}, \boldsymbol{y}_i)$ is the per-sample loss, such as cross entropy, and λ is a regularization coefficient. In this paper, we adopt the ℓ_2 regularizer $R(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w}$. The data weight ϵ_i is set to zero during training; its function will be explained shortly. See Table 1 for notations used in this paper.

Similar to the training dataset $\mathcal{D}_{\text{train}}$, the test dataset with M data points is defined as $\mathcal{D}_{\text{test}} = \{\boldsymbol{x}_i^{\text{test}}, \boldsymbol{y}_i^{\text{test}}\}_{i=1}^M$ with $\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \varnothing$. The test loss $\mathcal{L}_{\text{test}}$ is a measurement of model generalizability.

$$\mathcal{L}_{\text{test}}(\boldsymbol{w}_T) = \frac{1}{M} \sum_{(\boldsymbol{x}_i^{\text{test}}, \boldsymbol{y}_i^{\text{test}}) \in \mathcal{D}_{\text{test}}} \ell(f(\boldsymbol{x}_i^{\text{test}}, \boldsymbol{w}_T), \boldsymbol{y}_i^{\text{test}}). \tag{3}$$

We usually optimize the loss function using vanilla gradient descent (GD) or GD with momentum. Vanilla GD iteratively updates the parameters $\boldsymbol{w}$ as

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \eta_t \boldsymbol{g}_{t-1}, \tag{4}$$

where η_t is the learning rate. In GD with momentum, we first update the momentum $\boldsymbol{v}_t$, followed by updating $\boldsymbol{w}_t$.

$$\boldsymbol{v}_t = p\boldsymbol{v}_{t-1} + \boldsymbol{g}_{t-1}, \tag{5}$$

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \eta_t \boldsymbol{v}_t, \tag{6}$$

where p is a constant in $(0, 1)$ that determines the strength of the accumulated momentum. We repeat the optimization for T steps and arrive at the final network parameters $\boldsymbol{w}_T$. The stochastic versions of the two algorithms simply replace $\boldsymbol{g}_t$ with the gradients on mini-batches of training data.

Table 1: Frequently Used Notations

$\mathcal{D}_{\text{train}}$	Training dataset
$\mathcal{D}_{\text{test}}$	Test dataset
N	Size of the training dataset
$\boldsymbol{w}_t$	Model parameters at step t
ϵ_i	Marginal weight for the ith training sample
$(\boldsymbol{x}_i, \boldsymbol{y}_i)$	i^{th} training sample
$(\boldsymbol{x}_j^{\text{test}}, \boldsymbol{y}_j^{\text{test}})$	j^{th} test sample
$R(\boldsymbol{w})$	Regularization term, $\frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w}$
λ	Regularization coefficient
η_t	Learning rate at step t
$\mathcal{L}_{\text{train}}(\boldsymbol{w})$	Training loss at model parameter $\boldsymbol{w}$
$\mathcal{L}_{\text{train}}^{\text{er}}(\boldsymbol{w})$	Empirical risk, $\mathcal{L}_{\text{train}}(\boldsymbol{w}) - \lambda R(\boldsymbol{w})$
$\mathcal{L}_{\text{test}}(\boldsymbol{w})$	Test loss at model parameter $\boldsymbol{w}$
$\boldsymbol{g}_t$	Model gradient at step t, $\partial \mathcal{L}_{\text{train}}(\boldsymbol{w}_t)/\partial \boldsymbol{w}_t$
H_t	Model Hessian at step t, $\partial^2 \mathcal{L}_{\text{train}}(\boldsymbol{w}_t)/\partial \boldsymbol{w}_t^2$
H_t^{er}	Empirical-risk Hessian, $\partial^2 \mathcal{L}_{\text{train}}^{\text{er}}(\boldsymbol{w}_t)/\partial \boldsymbol{w}_t^2$
$\nabla_{t,i}$	Shorthand for $d\boldsymbol{w}_t/d\epsilon_i$

Measuring Data Contribution

Recall that every training sample $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ is associated with the data weight $1/N + \epsilon_i$ in the training loss (Equation (2)). Thus, we can remove the ith sample from the training by setting ϵ_i to $-1/N$. The resulting change in the test loss can be estimated with a first-order Taylor expansion. We define the contribution of the ith training data point on model performance, $\mathcal{C}(i)$, as the estimated change (which becomes exact when $N \to \infty$)

$$\mathcal{C}(i) := -\frac{1}{N} \frac{d\mathcal{L}_{\text{test}}(\boldsymbol{w}_T)}{d\epsilon_i}\bigg|_{\epsilon_i=0}. \tag{7}$$

If the removal of the data sample $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ from the training data causes test loss to increase, the sample would have a positive contribution. Otherwise, the sample hurts generalization performance and makes a negative contribution.

We can measure the contribution to any portion of the test set by simply replacing $\mathcal{L}_{\text{test}}$ in Equation (7) with the loss on the data points in question. For example, the contribution by the ith training sample on model performance regarding the jth test sample, $\mathcal{C}(i, j)$, is computed as:

$$\mathcal{C}(i, j) := -\frac{1}{N} \frac{d\ell(\boldsymbol{x}_j^{\text{test}}, \boldsymbol{y}_j^{\text{test}}, \boldsymbol{w}_T)}{d\epsilon_i}\bigg|_{\epsilon_i=0}. \tag{8}$$

In Eq. 7, $\mathcal{L}_{\text{test}}$ does not directly depend on ϵ_i and only through $\boldsymbol{w}$, we can write the total derivative as

$$\frac{d\mathcal{L}_{\text{test}}(\boldsymbol{w}_T)}{d\epsilon_i} = \frac{\partial \mathcal{L}_{\text{test}}(\boldsymbol{w}_T)}{\partial \boldsymbol{w}_T} \frac{d\boldsymbol{w}_T}{d\epsilon_i} = \frac{\partial \mathcal{L}_{\text{test}}(\boldsymbol{w}_T)}{\partial \boldsymbol{w}_T} \nabla_{T,i}. \tag{9}$$

$\frac{\partial \mathcal{L}_{\text{test}}(\boldsymbol{w}_T)}{\partial \boldsymbol{w}_T}$ can be computed exactly using backpropagation. However, computing $\nabla_{T,i} = \frac{d\boldsymbol{w}_T}{d\epsilon_i}$ is not so straightforward as ϵ_i is involved in the entire optimization process. Below,

Algorithm 1: Hypergradient Computation

Input : training sample $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, training dataset size N, batch size B, iteration number T, learning rate η, momentum p, regularization coefficient λ

Output: $\nabla_{T,i}$: the hypergradient of $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ over the entire T training iterations

1 /* Initialization */
2 $\nabla_{0,i} \leftarrow \mathbf{0}$
3 $\frac{\mathrm{d}\boldsymbol{v}_0}{\mathrm{d}\epsilon_i} \leftarrow \mathbf{0}$
4 /* Training */
5 **for** $t \leftarrow 1$ **to** T **do**
6   **if** *current batch contains* $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ **then**
7         $\frac{\mathrm{d}\boldsymbol{v}_t}{\mathrm{d}\epsilon_i} \leftarrow p\frac{\mathrm{d}\boldsymbol{v}_{t-1}}{\mathrm{d}\epsilon_i} + \lambda\nabla_{t-1,i}$
8   **else**
9         $\frac{\mathrm{d}\boldsymbol{v}_t}{\mathrm{d}\epsilon_i} \leftarrow p\frac{\mathrm{d}\boldsymbol{v}_{t-1}}{\mathrm{d}\epsilon_i} + \lambda\nabla_{t-1,i} + \frac{N}{B}\boldsymbol{g}_{t,i}$
10   **end if**
11     $\nabla_{t,i} \leftarrow \nabla_{t-1,i} - \eta_t \frac{\mathrm{d}\boldsymbol{v}_t}{\mathrm{d}\epsilon_i}$
12 **end for**
13 **return** $\nabla_{T,i}$

---

we give recurrence equations for $\nabla_{T,i}$ in whole-batch gradient descent.

In vanilla gradient descent, $\nabla_{t,i}$ can be computed recurrently as a function of $\nabla_{t-1,i}$ as follows:

$$
\begin{aligned}
\nabla_{t,i} &= \nabla_{t-1,i} - \eta_t(H_{t-1}\nabla_{t-1,i} + \frac{\partial \boldsymbol{g}_{t-1}}{\partial \epsilon_i}) \\
&= \nabla_{t-1,i} - \eta_t H_{t-1}^{er}\nabla_{t-1,i} - \eta_t\lambda\nabla_{t-1,i} - \eta_t\boldsymbol{g}_{t-1,i},
\end{aligned}
\tag{10}
$$

where $\boldsymbol{g}_{t,i}$ denotes the gradient of sample i. Note that since $\boldsymbol{g}_t$ is a function of $\boldsymbol{w}_t$, we have $\frac{\mathrm{d}\boldsymbol{g}_t}{\mathrm{d}\epsilon_i} = \frac{\partial \boldsymbol{g}_t}{\partial \boldsymbol{w}_t}\nabla_{t,}$, which involves the Hessian.

Similarly, for gradient descent with momentum,

$$
\frac{\mathrm{d}\boldsymbol{v}_t}{\mathrm{d}\epsilon_i} = p\frac{\mathrm{d}\boldsymbol{v}_{t-1}}{\mathrm{d}\epsilon_i} + H_{t-1}^{er}\nabla_{t-1,i} + \lambda\nabla_{t-1,i} + \boldsymbol{g}_{t-1,i}. \tag{11}
$$

$$
\nabla_{t,i} = \nabla_{t-1,i} - \eta_t\frac{\mathrm{d}\boldsymbol{v}_t}{\mathrm{d}\epsilon_i}, \tag{12}
$$

In both cases we have the initial conditions

$$
\nabla_{0,i} = \mathbf{0} \text{ and } \frac{\mathrm{d}\boldsymbol{v}_0}{\mathrm{d}\epsilon_i} = \mathbf{0}. \tag{13}
$$

By recurrently computing $\nabla_{t,i}$ through the entire optimization trajectory, HYDRA holistically measures the contribution of ith data point to the neural network. The algorithm for computing $\nabla_{T,i}$, which is essential for deriving the values of $\mathcal{C}(i)$ and $\mathcal{C}(i,j)$ is shown in Algorithm 1.

Fast Approximation

We propose a fast approximation technique, which sets the Hessian H_{t-1}^{er} in Equation (10) and Equation (11) to zero with an analytical bound on the approximation error. Algorithm 1 shows the mini-batch version of the algorithm. Although the Hessian-vector product can be approximated in

$\mathcal{O}(|\boldsymbol{w}|)$ time using a method akin to finite difference (Pearlmutter 1994), the computation is slow as it requires (usually two) additional backpropagations and is susceptible to truncation and discretization errors just like finite difference. Consequently, removing the Hessian provides desirable simplification to the computation.

In the data contribution Equation (9), the term $\frac{\mathrm{d}\mathcal{L}_{\text{test}}(\boldsymbol{w}_T)}{\mathrm{d}\boldsymbol{w}_T}$ can always be computed exactly, so this approximation only affects the $\nabla_{t,i}$ term. With the moderate conditions below, we can show that applying the proposed approximation results in bounded difference between the true $\nabla_{t,i}$ and its approximation $\widetilde{\nabla}_{t,i}$. Additionally, if we let the learning rate decay exponentially, the difference vanishes after sufficient training iterations.

Condition 1. *The training loss $\mathcal{L}_{train}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w})$ is twice differentiable.*

Condition 2. *The optimization process converges. That is,*

$$
\lim_{t\to\infty} \boldsymbol{w}_t = \hat{\boldsymbol{w}}. \tag{14}
$$

Condition 3. *The empirical risk function $\mathcal{L}_{train}^{er}$ has Lipschitz-continuous gradients with Lipschitz constant L. Formally,*

$$
\left\|\frac{\partial\mathcal{L}_{train}^{er}(\boldsymbol{w}_1)}{\partial\boldsymbol{w}_1} - \frac{\partial\mathcal{L}_{train}^{er}(\boldsymbol{w}_1)}{\partial\boldsymbol{w}_2}\right\| \leq L\|\boldsymbol{w}_1 - \boldsymbol{w}_2\|, \ \forall\boldsymbol{w}_1, \boldsymbol{w}_2 \tag{15}
$$

Condition 4. *The learning rate sequence η_t is nonincreasing and lower-bounded by 0. That is,*

$$
\eta_t \geq \eta_{t+1} > 0, \forall t. \tag{16}
$$

Condition 5. *The product of the learning rate η_t and weight decay coefficient λ satisfy $0 < \eta_t\lambda < 1, \forall t.$*

Condition 6. *The contribution measure sequence $\nabla_{t,i}$ does not diverge as $t \to \infty$ and is bounded by a constant $M_{\boldsymbol{w}}$. More formally,*

$$
\lim_{t\to\infty}\left\|\nabla_{t,i}\right\| < M_{\boldsymbol{w}}, \ \forall i \tag{17}
$$

Theorem 1. *Under the above conditions and vanilla GD, the norm of the approximation error is bounded by*

$$
\left\|\nabla_{t,i} - \widetilde{\nabla}_{t,i}\right\| < LM_{\boldsymbol{w}}\frac{\eta_1}{\eta_t\lambda}. \tag{18}
$$

Theorem 2. *Under the above conditions, vanilla GD, and an exponential decay schedule for the learning rate η, the approximation error diminishes when t tends to infinity.*

$$
\lim_{t\to\infty}\left\|\nabla_{t,i} - \widetilde{\nabla}_{t,i}\right\| = 0 \tag{19}
$$

We can also relax the Lipschitz-continuity constraint and derive similar results. The proof details are in the supplemental material. In the experiments, we empirically verify these theoretical results and show the approximation indeed leads to small errors.

Time and Space Complexity

It takes $\mathcal{O}(|\boldsymbol{w}|)$ time and space to compute the parameter gradients. Using the approach in Pearlmutter (1994), Hessian-vector products can be computed in $\mathcal{O}(|\boldsymbol{w}|)$ time and space. We also need $\mathcal{O}(|\boldsymbol{w}|)$ extra space to store previous $\nabla_{t-1,i}$, $\frac{d\boldsymbol{v}_{t-1}}{d\epsilon_i}$ and $\boldsymbol{g}_{t-1,i}$. Therefore, we need a total of $\mathcal{O}(T|\boldsymbol{w}|)$ time and $\mathcal{O}(|\boldsymbol{w}|)$ space to trace a single training sample. Tracing all training samples simultaneously requires $\mathcal{O}(T|\boldsymbol{w}||\mathcal{D}_{\text{train}}|)$ time and $\mathcal{O}(|\boldsymbol{w}||\mathcal{D}_{\text{train}}|)$ space.

Related Work

Influence Functions. As a precursor to our work, Koh and Liang (2017) measures the influence of a training data point $\boldsymbol{z}_i = (\boldsymbol{x}_i, \boldsymbol{y}_i)$ on a test data point $\boldsymbol{z}^{\text{test}} = (\boldsymbol{x}^{\text{test}}, \boldsymbol{y}^{\text{test}})$ as $\text{IF}(\boldsymbol{z}_i, \boldsymbol{z}^{\text{test}})$

$$\text{IF}(\boldsymbol{z}_i, \boldsymbol{z}^{\text{test}}) = -\frac{\partial \ell(\boldsymbol{x}^{\text{test}}, \boldsymbol{y}^{\text{test}}, \boldsymbol{w}_T)}{\partial \boldsymbol{w}_T}^{\top} H_T^{-1} \frac{\partial \ell(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{w}_T)}{\partial \boldsymbol{w}_T} \tag{20}$$

The difficulty of Equation (20) lies in computing the inverse of Hessian H_T^{-1}, for which the naive approach is infeasible due to the size of modern DNNs. Koh & Liang introduced two methods for this purpose. First, when H is positive definite, we find $H^{-1}\boldsymbol{v}$ as the vector ϕ that minimizes $\phi^{\top} H \phi + \boldsymbol{v}^{\top} \phi$ using conjugate gradient. When H is not positive definite or is singular, we increment its diagonal by a small damping factor.

The second method to find $H^{-1}\boldsymbol{v}$ is to iterate the following until convergence,

$$H^{-1}\boldsymbol{v} \leftarrow \boldsymbol{v} + \left(I - \frac{\partial^2 \ell(\boldsymbol{x}_d, \boldsymbol{y}_d, \boldsymbol{w}_T)}{\partial \boldsymbol{w}_T^2} \right) H^{-1}\boldsymbol{v}, \tag{21}$$

where the data $(\boldsymbol{x}_d, \boldsymbol{y}_d)$ is randomly sampled in each iteration and the initial $H^{-1}\boldsymbol{v}$ is set to $\boldsymbol{v}$. A scaling factor is applied to the $H_{k-1}^{-1}\boldsymbol{v}$ term to ensure that the largest eigenvalue of H^{-1} is less than one. In practice, we use the Monte-Carlo expectation from multiple runs. Both methods adopt the approximate Hessian-vector product (Pearlmutter 1994).

Although these techniques render the time and space complexity manageable, dealing with the inverse Hessian is still tedious and slow. Numerical considerations such as damping and scaling factors require careful treatment. Errors can be introduced by the variance of the Monte-Carlo expectation and the errors in the Hessian-vector product. In comparison, HYDRA offers a simplified method that removes the Hessian and a theoretical upper bound for the introduced error.

Interpreting DNNs. A DNN prediction can be interpreted by the training data that have the most influence on the prediction. The work of Cook and Weisberg (1980) on linear regression is one of the earliest technique along this line of thought. Sharchilev et al. (2018) study tree ensembles. Koh et al. (2019) extend influence functions to the effects of groups of data points. Chen et al. (2020) investigate the influence of data used in the pretraining on the finetuning task. Noting the outliers and mislabled data often have outsize influence, Barshan, Brunet, and Dziugaite (2020) propose RelatIF to accurately measure local influence. Other data-based interpretations utilize kernel functions (Yeh et al. 2018; Khanna et al. 2018), Shapley values (Jia et al. 2019a,b; Ghorbani and Zou 2019), and network layers that identify prototypes and object parts (Branson et al. 2014; Kim, Khanna, and Koyejo 2016; Zhang, Wu, and Zhu 2018; Chen et al. 2019).

Similar to our work, Hara, Nitanda, and Maehara (2019) estimate the change of model parameters after the removal of a point by extrapolating via the Hessians along the training trajectory. In comparison, HYDRA directly estimates the data contribution instead of the model parameters and disregards the unwieldy Hessian, which Hara, Nitanda, and Maehara (2019) crucially depend on. Yoon, Arik, and Pfister (2019) present a reinforcement learning approach that optimizes validation loss by assigning selection probabilities to data points. The probabilities can serve as data valuation, but cannot explain individual model predictions.

Neural networks can be interpreted from other perspectives. Network weights can be visualized (Zeiler and Fergus 2014; Bau et al. 2018; Fong and Vedaldi 2018); behaviors of a complex network can be approximated with simple models that are easy to understand (Ribeiro, Singh, and Guestrin 2016; Zhou, Zhou, and Hooker 2018; Lakkaraju et al. 2017; Chen et al. 2019; Ahern et al. 2019). In addition, model predictions can be explained by identifying important features in the input (Simonyan, Vedaldi, and Zisserman 2013; Springenberg et al. 2014; Smilkov et al. 2017; Selvaraju et al. 2017; Sundararajan, Taly, and Yan 2017; Du et al. 2018).

Hypergradient-based Optimization. Hypergradient is the gradient of the validation loss or test loss w.r.t. the hyperparameters (Bengio 1999; Domke 2012; Franceschi et al. 2017, 2018; Lorraine, Vicol, and Duvenaud 2020). With hyperparameters ϵ, model parameters $\boldsymbol{w}$, and validation loss $\mathcal{L}_{\text{val}}$, the hyperparameter optimization problem can be defined as the bi-level optimization

$$\epsilon^* = \underset{\epsilon}{\arg\min} \, \mathcal{L}_{\text{val}}(\epsilon, \boldsymbol{w}^*(\epsilon))$$
$$\text{s.t. } \boldsymbol{w}^*(\epsilon) = \underset{\boldsymbol{w}}{\arg\min} \, \mathcal{L}_{\text{train}}(\epsilon, \boldsymbol{w}). \tag{22}$$

Note we write $\boldsymbol{w}^*(\epsilon)$ to underscore the fact that ϵ influences the optimization of $\boldsymbol{w}^*$. Thus, in order to compute the gradient of $\mathcal{L}_{\text{val}}(\epsilon, \boldsymbol{w}^*(\epsilon))$ w.r.t. ϵ, we must find the total derivative $d\boldsymbol{w}^*(\epsilon)/d\epsilon$, which may be computed by unrolling $\boldsymbol{w}^*$ throughout the training trajectory or for only a few steps as an approximation.

Hypergradient enables gradient-based optimization of hyperparameters, such as learning rates (Donini et al. 2020; Metz et al. 2019), network architectures (Liu, Simonyan, and Yang 2019), or data augmentation (Lin et al. 2019). Maclaurin, Duvenaud, and Adams (2015) and Wang et al. (2018) distill a large training dataset into 10-100 data points. Bohdal, Yang, and Hospedales (2020) distill data labels instead of input features. Mehra and Hamm (2019) use penalty functions to avoid hypergradient in the bi-level optimization.

A key difference between HYDRA and works on hypergradient-based optimization is that we use the hypergradient $\nabla_{t,i}$ to assign credit to training data rather than to
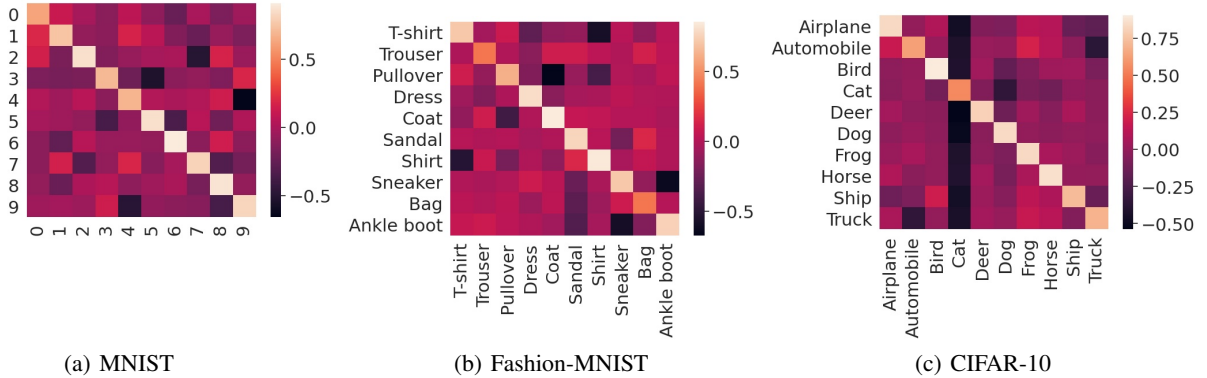
| (a) MNIST | (b) Fashion-MNIST | (c) CIFAR-10 |

Figure 2: Inter-class contribution shown as heatmaps. Rows represent classes of training data and columns represent classes of test data.

Table 2: Training samples with extreme influence on the test data, their ground-truth and predicted labels, model confidence, and contribution on the test set. A positive contribution means the training sample reduces overall test loss.

Training Sample	True vs. Predicted Labels	Model Conf.	Contribution to Test Data
	5 / 5	0.66	-1.0×10^{-4}
	8 / 8	0.73	4.8×10^{-5}
	Coat/Pullover	0.66	0.93×10^{-3}
	Deer/Deer	0.97	-1.01×10^{-5}

optimize ϵ. As a result, HYDRA is less vulnerable to accumulated inaccuracies in the hypergradient estimates over time. We take advantage of this by omitting the Hessian with bounded error.

Experimental Evaluation

Datasets and Networks

We use three image recognition datasets in our experiments: MNIST (Lecun et al. 1998), Fashion-MNIST (Xiao, Rasul, and Vollgraf 2017), and CIFAR-10 (Krizhevsky 2009). We choose two networks, LeNet-5 (Lecun et al. 1998) of 61,706 trainable parameters and DenseNet-40 (Huang et al. 2017) of 176,122 trainable parameters. Details of the datasets and networks are in the supplemental material.

Manual Inspection

In the first experiment, we verify that HYDRA provides results that agree with our intuition. We compute the mean and standard deviation of the contributions of all training data points on $\mathcal{D}_{\text{test}}$, which yields $(-3.17 \times 10^{-9}, 1.19 \times 10^{-6})$ for MNIST, $(-4.9858 \times 10^{-7}, 0.1 \times 10^{-3})$ for Fashion-MNIST, and $(-1.8676 \times 10^{-7}, 3.206 \times 10^{-6})$ for CIFAR-10. The mean values are very close to zero, and the standard deviations are substantially greater than the mean. This is consistent with our expectation as the contribution of single data point in a large dataset is likely rather small. The standard deviations indicate that some data points have extreme contribution values, which we examine below.

Influential Examples. Table 2 shows data points from the three datasets with extreme contributions. The first two rows are examples from MNIST. The third and fourth examples are from Fashion-MNIST and CIFAR-10, respectively. The first image is labeled as 5 but closely resembles 6. The second is labeled as 8 and has an unusual upper half. Their average contribution to all test data points is 4-5 orders of magnitudes higher than the average ($\approx 10^{-9}$). Due to their unusual appearances, these data points stand out from the rest of the training data and hence exert large influence on the model. Similarly, the two other examples have atypical appearances in their category and have large influences on the networks' predictions. Due to space considerations, we leave more examples to the supplemental material.

Inter-class Contribution. Next, we inspect the contribution from the training data of one class to the test data of another class. By the class label, we partition the training dataset into C subsets $\mathcal{S}_1, \ldots, \mathcal{S}_C$, where C is the number of classes. Similarly, we partition the test set into C subsets $\mathcal{T}_1, \ldots, \mathcal{T}_C$. The contribution $\mathcal{C}(\mathcal{S}_i, \mathcal{T}_j)$ from a training class $\mathcal{S}_i$ to a test class $\mathcal{T}_j$ is the average contribution over all possible pairs of samples

$$\mathcal{C}(\mathcal{S}_i, \mathcal{T}_j) := \frac{1}{|\mathcal{S}_i||\mathcal{T}_j|} \sum_{k \in \mathcal{S}_i} \sum_{k' \in \mathcal{T}_j} \mathcal{C}(k, k'). \qquad (23)$$

We plot the inter-class contribution matrices for MNIST,

Fashion-MNIST, and CIFAR-10 in Figure 2. The rows represent training data classes and the columns represent test data classes. As normalization, we divide every matrix entry by $\sqrt{\text{column sum} \times \text{row sum}}$.

In all three datasets, the maximum contribution values for each row and column appear on the diagonal, indicating that training data always produce the most reduction in test loss for their own class. In MNIST, we observe symmetrically low contribution between the digit pairs (3,5), (2, 7), and (4,9). As the two classes in each pair have similar appearances, there is competition between them. Lowering the test loss of one class will likely increase the test loss of the other class. Similar symmetry exists for the pairs (Pullover, Coat), (T-shirt, Shirt), and (Sneaker, Ankle boot) in Fashion-MNIST, and between Automobile and Truck in CIFAR-10. We find these to be consistent with intuition.

Approximation Error

Theorems 1 and 2 show that, under moderate conditions, the error from disregarding the Hessian term is bounded. In order to test this hypothesis, we track the contribution of 500 samples from Fashion-MNIST, including 1% from every class, through the training trajectory using both the Hessian-aware method and the approximation of HYDRA. We train LeNet-5 for 200 epochs with the same hyperparameters as before and record the average ℓ_2 norm of the approximation error $\left\| \nabla_{t,i} - \widetilde{\nabla}_{t,i} \right\|_2$ every 4 epochs. Figure 3 shows the results. We observe that the error stabilizes to 0.74 after about 70 epochs, which agrees with the theoretical result that the approximation error is bounded.

We further benchmark HYDRA's approximation method and influence functions (Koh and Liang 2017) against the Hessian-aware method. While we recognize that it is difficult to establish any ground truth for data contribution values, both influence functions and HYDRA may be considered as approximations of the Hessian-aware method. Therefore, we take the Hessian-aware data contribution values as the gold standard. In the experiments below, we compute the data contribution from influence functions using the Monte Carlo expectation method (Equation 21) starting from epoch 100, when the optimization is close to convergence.

We adopt two evaluation metrics used by Koh et al. (2019). First, we measure the percentage of data points where HYDRA and influence functions erroneously flip the sign of the contribution (Figure 4). That is, a data point making a positive contribution is mistakenly assigned negative contribution and vice versa. The error rate of HYDRA converges after 100 epochs to an average of 0.014 with a standard deviation of 0.002. The error rate of influence functions is 0.130 on average with a standard deviation of 0.013. Second, we rank the data points by their contribution and compute Spearman's rank correlation against the Hessian-aware method (Figure 5). After 100 epochs, the correlation of HYDRA converges to 0.986 with a standard deviation of 0.002, whereas influence functions show substantially lower correlation with a mean of 0.820 that fluctuate more wildly (standard deviation = 0.024).

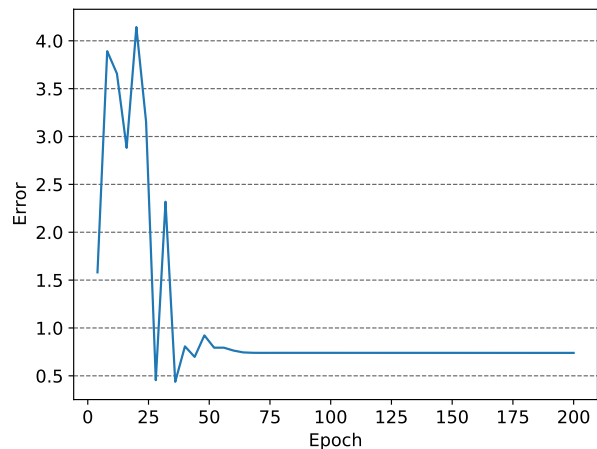In summary, the experimental comparison shows that HY-

Figure 3: HYDRA's hypergradient approximation error $\left\| \nabla_{t,i} - \widetilde{\nabla}_{t,i} \right\|$ averaged over 500 Fashion-MNIST data points.
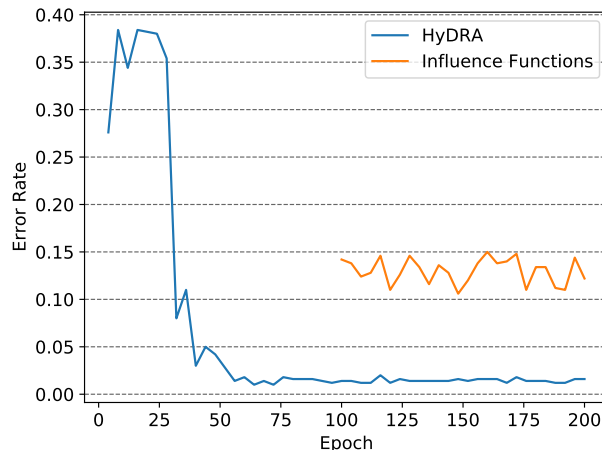

Figure 4: Sign error rates of HYDRA's approximation and influence functions on Fashion-MNIST

DRA's approximation to be not only more accurate but also an order of magnitude more stable than influence functions. This demonstrates that accounting for the entire trajectory is important in accurately estimating data influence. In these experiments, ignoring the optimization trajectory increases the error in the estimated contributions by 11.6% to 16.6%.

Importantly, in these experiments, removing the Hessian reduces the total running time by 22.0 folds. Tracking the contribution of the 500 data points for 200 epochs with Hessian-vector products took 59.5 hours on a server with 4 Nvidia TitanX Pascal GPUs, an Intel i7-5930K CPU, and 48 GB RAM. In contrast, the proposed fast approximation method reduced the training time to 2.7 hours.

Debugging Training Data

In this section, we investigate if HYDRA can help debug datasets with erroneous labels. In the first experiment, we compare different methods by their ability to clean noisy
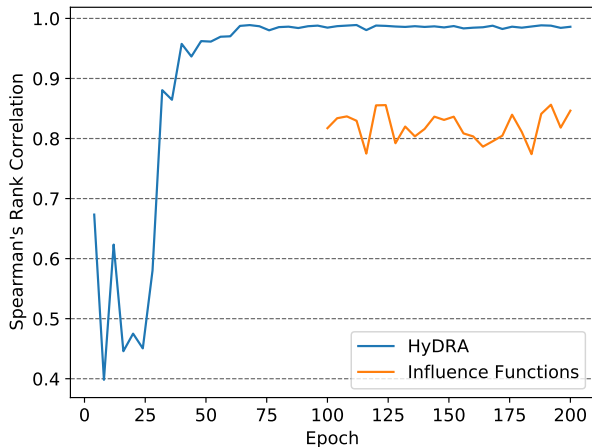
Figure 5: Spearman's rank correlation with the ground truth for HYDRA's approximation technique and influence functions on Fashion-MNIST

Table 3: Classification accuracy when different methods are used to clean the dataset with a known proportion of label noise.

Dataset	Wrong Labels	Method	Final Accuracy
MNIST	80%	No Filtering	72.93%
		Inf. Func.	90.07%
		HYDRA	**98.31**%
Fashion-MNIST	80%	No Filtering	78.52%
		Inf. Func.	63.20%
		HYDRA	**86.72**%
CIFAR-10	80%	No Filtering	31.36%
		Inf. Func.	42.12%
		HYDRA	**72.01**%

data. We create synthetic datasets with a known proportion $r\%$ of data points have randomly permuted, erroneous labels. We use either HYDRA or influence functions to estimate the data contribution and remove the least useful $r\%$ of training data. The remaining training data are used to retrain the network. As deep neural networks are known to be robust against noise (Rolnick et al. 2018), we adopt large error rates. The supplemental material contains detailed settings and additional experiments. Table 3 shows the results. HY-DRA demonstrates clear performance advantages over influence functions, which gradually increases as we go from the easy MNIST dataset to the most difficult CIFAR-10.

In the second experiment, we aim to detect data points with erroneous labels unsupervisedly by clustering the data points using their contribution to test data points. We create synthetic data by randomly selecting 50% of training data and uniformly assigning incorrect labels to them. After the models are trained, for each training data point, we com-

Table 4: The overlap between automatically identified clusters of training data and the gold-standard clusters of correctly labeled and randomly labeled data.

Dataset	Method	Overlap	
		Correct Labels	Random Labels
MNIST	HYDRA	97.13%	97.16%
	Inf. Func.	71.89%	67.19%
Fashion-MNIST	HYDRA	78.06%	77.73%
	Inf. Func.	41.26%	33.22%

pute a 1000-dimensional feature vector, which consists of its contributions to 1000 randomly sampled validation samples. This feature vector is then discretized to $\{+1, -1\}^{1000}$ based on the signs of the values.

After that, we gather the training data points with the same class label. Note that a half of these data is correctly labeled and the other half is incorrectly labeled. We cluster them by k-means on the 1000-dimensional feature vectors. The hope is that the clustering would separate correctly labeled data from the incorrectly labeled data. As a performance metric, we calculate the Jaccard index, or the intersection over union, between the identified clusters and the ground-truth partition. Higher Jaccard indices indicate better clusters. We repeat this for every class label and report the average.

Table 4 reports the average Jaccard indices over 10 classes. We observe that Fashion-MNIST poses a much more difficult challenge to the unsupervised task than MNIST and causes both methods to obtain lower performance. On both datasets, HYDRA outperforms influence functions; the performance gaps range from 25.25% to 44.51%. We attribute the superior performance of HYDRA to the accuracy of the estimated data contribution values.

Conclusions

We propose HYDRA for estimating the contribution of training data by differentiating the test loss against training data weights through time. To simplify computation, we provide a Hessian-free approximation to the exact derivative and establish an analytical upper bound for the error. The experiments compare HYDRA to influence functions (Koh and Liang 2017), and confirm that HYDRA provide more accurate estimates for data contribution, which facilitates the identification of label noise in the training data. With HY-DRA, the AI research community can be equipped with an effective and computationally efficient tool to intepret the influence of training samples to DNN predictions.

Acknowledgments

Potential Ethical Impact

The deployment of machine learning in critical areas such as law enforcement and human resources has raised concerns regarding potential bias and prejudice of such algorithms (Frazier et al. 2019; Poyiadzi et al. 2020). In many cases, the apparent bias is not due to algorithmic design but to existing stereotypes inadvertently captured by training data (Mehrabi et al. 2019; Bryant and Howard 2019). Therefore, the ability to trace an algorithmic prediction back to training data samples could help in mitigating and eventually eliminating model bias in machine learning. We caution that the elimination of bias requires systemic efforts that extend well beyond pure algorithmic research. Having said that, we believe this work could play a positive role in that direction.

References

Ahern, I.; Noack, A.; Guzman-Nateras, L.; Dou, D.; Li, B.; and Huan, J. 2019. NormLime: A New Feature Importance Metric for Explaining Deep Neural Networks. *arXiv Preprint 1909.04200*.

Alaa, A. M.; and van der Schaar, M. 2020. Frequentist Uncertainty in Recurrent Neural Networks via Blockwise Influence Functions. In *ICML*.

Arora, S.; Cohen, N.; and Hazan, E. 2018. On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization. *arXiv 1802.06509*.

Barshan, E.; Brunet, M.-E.; and Dziugaite, G. K. 2020. RelatIF: Identifying Explanatory Training Examples via Relative Influence. *arXiv Preprint 2003.11630*.

Bau, D.; Zhu, J.-Y.; Strobelt, H.; Bolei, Z.; Tenenbaum, J. B.; Freeman, W. T.; and Torralba, A. 2018. GAN Dissection: Visualizing and Understanding Generative Adversarial Networks. *arXiv preprint arXiv:1811.10597*.

Bengio, Y. 1999. Continuous Optimization of Hyper-Parameters. Technical report, Department of Computer Science and Operations Research, University of Montreal.

Bohdal, O.; Yang, Y.; and Hospedales, T. 2020. Flexible Dataset Distillation: Learn Labels Instead of Images. *arXiv 2006.08572*.

Branson, S.; Horn, G. V.; Belongie, S.; and Perona, P. 2014. Bird Species Categorization Using Pose Normalized Deep Convolutional Nets. In *BMVC*.

Bryant, D.; and Howard, A. 2019. A Comparative Analysis of Emotion-Detecting AI Systems with Respect to Algorithm Performance and Dataset Diversity. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*.

Chen, C.; Li, O.; Tao, C.; Barnett, A. J.; Su, J.; and Rudin, C. 2019. This Looks Like That: Deep Learning for Interpretable Image Recognition. In *NeurIPS*.

Chen, H.; Si, S.; Li, Y.; Chelba, C.; Kumar, S.; Boning, D.; and Hsieh, C.-J. 2020. Multi-Stage Influence Function. *arXiv 2007.09081*.

Chen, R.; Chen, H.; Huang, G.; Ren, J.; and Zhang, Q. 2019. Explaining Neural Networks Semantically and Quantitatively. In *ICCV*.

Cook, R. D.; and Weisberg, S. 1980. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics* 22(4): 495–508.

Domke, J. 2012. Generic Methods for Optimization-Based Modeling. In *AISTATS*.

Donini, M.; Franceschi, L.; Pontil, M.; Majumder, O.; and Frasconi, P. 2020. MARTHE: Scheduling the Learning Rate Via Online Hypergradients. In *IJCAI*.

Du, M.; Liu, N.; Song, Q.; and Hu, X. 2018. Towards explanation of DNN-based prediction with guided feature inversion. In *KDD*.

Fong, R.; and Vedaldi, A. 2018. Net2Vec: Quantifying and Explaining how Concepts are Encoded by Filters in Deep Neural Networks. *arXiv preprint arXiv:1801.03454*.

Franceschi, L.; Donini, M.; Frasconi, P.; and Pontil, M. 2017. Forward and Reverse Gradient-Based Hyperparameter Optimization. *arXiv 1703.01785*.

Franceschi, L.; Frasconi, P.; Salzo, S.; Grazzi, R.; and Pontil, M. 2018. Bilevel Programming for Hyperparameter Optimization and Meta-Learning. *arXiv 1806.04910*.

Frazier, S.; Nahian, M. S. A.; Riedl, M.; and Harrison, B. 2019. Learning Norms from Stories: A Prior for Value Aligned Agents. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*.

Ghorbani, A.; and Zou, J. 2019. Data Shapley: Equitable Valuation of Data for Machine Learning. In *International Conference on Machine Learning*, 2242–2251.

Hara, S.; Nitanda, A.; and Maehara, T. 2019. Data Cleansing for Models Trained with SGD. In *NeurIPS*.

Huang, G.; Liu, Z.; van der Maaten, L.; and Weinberger, K. Q. 2017. Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Jia, R.; Dao, D.; Wang, B.; Hubis, F. A.; Hynes, N.; Gurel, N. M.; Li, B.; Zhang, C.; Song, D.; and Spanos, C. 2019a. Towards Efficient Data Valuation Based on the Shapley Value. In *AISTATS*.

Jia, R.; Sun, X.; Xu, J.; Zhang, C.; Li, B.; and Song, D. 2019b. An Empirical and Comparative Analysis of Data Valuation with Scalable Algorithms. *arXiv Preprint 1911.07128*.

Khanna, R.; Kim, B.; Ghosh, J.; and Koyejo, O. 2018. Interpreting Black Box Predictions using Fisher Kernels. In *ICML*.

Kim, B.; Khanna, R.; and Koyejo, O. O. 2016. Examples are not enough, learn to criticize! Criticism for Interpretability. In *NeurIPS*.

Koh, P. W.; Ang, K.-S.; Teo, H. H. K.; and Liang, P. 2019. On the Accuracy of Influence Functions for Measuring Group Effects. In *NeurIPS*.

Koh, P. W.; and Liang, P. 2017. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning*.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Lakkaraju, H.; Kamar, E.; Caruana, R.; and Leskovec, J. 2017. Interpretable & explorable approximations of black box models. *arXiv preprint arXiv:1707.01154* .

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11): 2278–2324.

Li, Y.; and Liang, Y. 2018. Learning Overparameterized Neural Networks via Stochastic Gradient Descent on Structured Data. In *NeurIPS*.

Lin, C.; Guo, M.; Li, C.; Xin, Y.; Wu, W.; Lin, D.; Ouyang, W.; and Yan, J. 2019. Online Hyper-parameter Learning for Auto-Augmentation Strategy. *arXiv 1905.07373* .

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *ICLR*.

Lorraine, J.; Vicol, P.; and Duvenaud, D. 2020. Optimizing Millions of Hyperparameters by Implicit Differentiation. In *AISTATS*.

Maclaurin, D.; Duvenaud, D.; and Adams, R. 2015. Gradient-based hyperparameter optimization through reversible learning. In *ICML*.

Mehra, A.; and Hamm, J. 2019. Penalty Method for Inversion-Free Deep Bilevel Optimization. *arXiv 1911.03432* .

Mehrabi, N.; Morstatter, F.; Saxena, N.; Lerman, K.; and Galstyan, A. 2019. A Survey on Bias and Fairness in Machine Learning. *arXiv Preprint arXiv: 1908.09635* .

Metz, L.; Maheswaranathan, N.; Nixon, J.; Freeman, C. D.; and Sohl-Dickstein, J. 2019. Understanding and correcting pathologies in the training of learned optimizers. In *ICML*.

Pearlmutter, B. A. 1994. Fast Exact Multiplication by the Hessian. *Neural Computation* 6(1): 147–160.

Poyiadzi, R.; Sokol, K.; Santos-Rodriguez, R.; De Bie, T.; and Flach, P. 2020. FACE: Feasible and Actionable Counterfactual Explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *KDD*.

Rolnick, D.; Veit, A.; Belongie, S.; and Shavit, N. 2018. Deep Learning is Robust to Massive Label Noise. *arXiv Preprint 1705.10694* .

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *ICCV* .

Sharchilev, B.; Ustinovsky, Y.; Serdyukov, P.; and de Rijke, M. 2018. Finding influential training samples for gradient boosted decision trees. *arXiv preprint arXiv:1802.06640* .

Simonyan, K.; Vedaldi, A.; and Zisserman, A. 2013. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv 1312.6034* .

Smilkov, D.; Thorat, N.; Kim, B.; Viégas, F. B.; and Wattenberg, M. 2017. SmoothGrad: removing noise by adding noise. In *ICML*.

Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; and Riedmiller, M. 2014. Striving for Simplicity: The All Convolutional Net. In *ICLR Workshop*.

Sundararajan, M.; Taly, A.; and Yan, Q. 2017. Axiomatic Attribution for Deep Networks. In *Proceedings of the International Conference on Machine Learning*.

Wang, T.; Zhu, J.-Y.; Torralba, A.; and Efros, A. A. 2018. Dataset Distillation. *arXiv 1811.10959* .

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv 1708.07747* .

Yeh, C.-K.; Kim, J. S.; Yen, I. E. H.; and Ravikumar, P. 2018. Representer Point Selection for Explaining Deep Neural Networks. In *NeurIPS*.

Yoon, J.; Arik, S. O.; and Pfister, T. 2019. Data Valuation using Reinforcement Learning. In *ICML*.

Zeiler, M. D.; and Fergus, R. 2014. Visualizing and Understanding Convolutional Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 818–833.

Zhang, Q.; Wu, Y. N.; and Zhu, S.-C. 2018. Interpretable Convolutional Neural Networks. *CVPR* .

Zhou, Y.; Zhou, Z.; and Hooker, G. 2018. Approximation trees: Statistical stability in model distillation. *arXiv preprint arXiv:1808.07573* .