Understanding Black-box Predictions via Influence Functions

Pang Wei Koh [1] **Percy Liang** [1]

Abstract

How can we explain the predictions of a black-box model? In this paper, we use influence functions — a classic technique from robust statistics — to trace a model's prediction through the learning algorithm and back to its training data, thereby identifying training points most responsible for a given prediction. To scale up influence functions to modern machine learning settings, we develop a simple, efficient implementation that requires only oracle access to gradients and Hessian-vector products. We show that even on non-convex and non-differentiable models where the theory breaks down, approximations to influence functions can still provide valuable information. On linear models and convolutional neural networks, we demonstrate that influence functions are useful for multiple purposes: understanding model behavior, debugging models, detecting dataset errors, and even creating visually-indistinguishable training-set attacks.

1. Introduction

A key question often asked of machine learning systems is "Why did the system make this prediction?" We want models that are not just high-performing but also explainable. By understanding why a model does what it does, we can hope to improve the model (Amershi et al., 2015), discover new science (Shrikumar et al., 2016), and provide end-users with explanations of actions that impact them (Goodman & Flaxman, 2016).

However, the best-performing models in many domains — e.g., deep neural networks for image and speech recognition (Krizhevsky et al., 2012) — are complicated, black-box models whose predictions seem hard to explain. Work on interpreting these black-box models has focused on understanding how a fixed model leads to particular predictions, e.g., by locally fitting a simpler model around the test point (Ribeiro et al., 2016) or by perturbing the test point to see how the prediction changes (Simonyan et al., 2013; Li et al., 2016b; Datta et al., 2016; Adler et al., 2016). These works explain the predictions in terms of the model, but how can we explain where the model came from?

In this paper, we tackle this question by tracing a model's predictions through its learning algorithm and back to the training data, where the model parameters ultimately derive from. To formalize the impact of a training point on a prediction, we ask the counterfactual: what would happen if we did not have this training point, or if the values of this training point were changed slightly?

Answering this question by perturbing the data and retraining the model can be prohibitively expensive. To overcome this problem, we use influence functions, a classic technique from robust statistics (Cook & Weisberg, 1980) that tells us how the model parameters change as we upweight a training point by an infinitesimal amount. This allows us to "differentiate through the training" to estimate in closed-form the effect of a variety of training perturbations.

Despite their rich history in statistics, influence functions have not seen widespread use in machine learning; to the best of our knowledge, the work closest to ours is Wojnowicz et al. (2016), which introduced a method for approximating a quantity related to influence in generalized linear models. One obstacle to adoption is that influence functions require expensive second derivative calculations and assume model differentiability and convexity, which limits their applicability in modern contexts where models are often non-differentiable, non-convex, and high-dimensional. We address these challenges by showing that we can efficiently approximate influence functions using second-order optimization techniques (Pearlmutter, 1994; Martens, 2010; Agarwal et al., 2016), and that they remain accurate even as the underlying assumptions of differentiability and convexity degrade.

Influence functions capture the core idea of studying models through the lens of their training data. We show that they are a versatile tool that can be applied to a wide variety of seemingly disparate tasks: understanding model behavior, debugging models, detecting dataset errors, and creating visually-indistinguishable adversarial *training* examples that can flip neural network test predictions, the training set analogue of Goodfellow et al. (2015).

[1]Stanford University, Stanford, CA. Correspondence to: Pang Wei Koh <pangwei@cs.stanford.edu>, Percy Liang <pliang@cs.stanford.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

2. Approach

Consider a prediction problem from some input space $\mathcal{X}$ (e.g., images) to an output space $\mathcal{Y}$ (e.g., labels). We are given training points $z_1, \ldots, z_n$, where $z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. For a point z and parameters $\theta \in \Theta$, let $L(z, \theta)$ be the loss, and let $\frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$ be the empirical risk. The empirical risk minimizer is given by $\hat{\theta} \stackrel{\text{def}}{=} \arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$.[1] Assume that the empirical risk is twice-differentiable and strictly convex in θ; in Section 4 we explore relaxing these assumptions.

2.1. Upweighting a training point

Our goal is to understand the effect of training points on a model's predictions. We formalize this goal by asking the counterfactual: how would the model's predictions change if we did not have this training point?

Let us begin by studying the change in model parameters due to removing a point z from the training set. Formally, this change is $\hat{\theta}_{-z} - \hat{\theta}$, where $\hat{\theta}_{-z} \stackrel{\text{def}}{=} \arg\min_{\theta\in\Theta} \sum_{z_i \neq z} L(z_i, \theta)$. However, retraining the model for each removed z is prohibitively slow.

Fortunately, influence functions give us an efficient approximation. The idea is to compute the parameter change if z were upweighted by some small ϵ, giving us new parameters $\hat{\theta}_{\epsilon,z} \stackrel{\text{def}}{=} \arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z, \theta)$. A classic result (Cook & Weisberg, 1982) tells us that the influence of upweighting z on the parameters $\hat{\theta}$ is given by

$$\mathcal{I}_{\text{up,params}}(z) \stackrel{\text{def}}{=} \left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta}), \quad (1)$$

where $H_{\hat{\theta}} \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta^2 L(z_i, \hat{\theta})$ is the Hessian and is positive definite (PD) by assumption. In essence, we form a quadratic approximation to the empirical risk around $\hat{\theta}$ and take a single Newton step; see appendix A for a derivation. Since removing a point z is the same as upweighting it by $\epsilon = -\frac{1}{n}$, we can linearly approximate the parameter change due to removing z by computing $\hat{\theta}_{-z} - \hat{\theta} \approx -\frac{1}{n}\mathcal{I}_{\text{up,params}}(z)$, without retraining the model.

Next, we apply the chain rule to measure how upweighting z changes functions of $\hat{\theta}$. In particular, the influence of upweighting z on the loss at a test point z_{test} again has a closed-form expression:

$$\begin{aligned}\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) &\stackrel{\text{def}}{=} \left.\frac{dL(z_{\text{test}}, \hat{\theta}_{\epsilon,z})}{d\epsilon}\right|_{\epsilon=0} \quad (2)\\ &= \nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top \left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0}\\ &= -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta}).\end{aligned}$$

[1] We fold in any regularization terms into L.

2.2. Perturbing a training input

Let us develop a finer-grained notion of influence by studying a different counterfactual: how would the model's predictions change if a training input were modified?

For a training point $z = (x, y)$, define $z_\delta \stackrel{\text{def}}{=} (x + \delta, y)$. Consider the perturbation $z \mapsto z_\delta$, and let $\hat{\theta}_{z_\delta,-z}$ be the empirical risk minimizer on the training points with z_δ in place of z. To approximate its effects, define the parameters resulting from moving ϵ mass from z onto z_δ: $\hat{\theta}_{\epsilon,z_\delta,-z} \stackrel{\text{def}}{=} \arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z_\delta, \theta) - \epsilon L(z, \theta)$. An analogous calculation to (1) yields:

$$\begin{aligned}\left.\frac{d\hat{\theta}_{\epsilon,z_\delta,-z}}{d\epsilon}\right|_{\epsilon=0} &= \mathcal{I}_{\text{up,params}}(z_\delta) - \mathcal{I}_{\text{up,params}}(z)\\ &= -H_{\hat{\theta}}^{-1}\big(\nabla_\theta L(z_\delta, \hat{\theta}) - \nabla_\theta L(z, \hat{\theta})\big). \quad (3)\end{aligned}$$

As before, we can make the linear approximation $\hat{\theta}_{z_\delta,-z} - \hat{\theta} \approx -\frac{1}{n}(\mathcal{I}_{\text{up,params}}(z_\delta) - \mathcal{I}_{\text{up,params}}(z))$, giving us a closed-form estimate of the effect of $z \mapsto z_\delta$ on the model. Analogous equations also apply for changes in y. While influence functions might appear to only work for infinitesimal (therefore continuous) perturbations, it is important to note that this approximation holds for arbitrary δ: the ϵ-upweighting scheme allows us to smoothly interpolate between z and z_δ. This is particularly useful for working with discrete data (e.g., in NLP) or with discrete label changes.

If x is continuous and δ is small, we can further approximate (3). Assume that the input domain $\mathcal{X} \subseteq \mathbb{R}^d$, the parameter space $\Theta \subseteq \mathbb{R}^p$, and L is differentiable in θ and x. As $\|\delta\| \to 0$, $\nabla_\theta L(z_\delta, \hat{\theta}) - \nabla_\theta L(z, \hat{\theta}) \approx [\nabla_x \nabla_\theta L(z, \hat{\theta})]\delta$, where $\nabla_x \nabla_\theta L(z, \hat{\theta}) \in \mathbb{R}^{p\times d}$. Substituting into (3),

$$\left.\frac{d\hat{\theta}_{\epsilon,z_\delta,-z}}{d\epsilon}\right|_{\epsilon=0} \approx -H_{\hat{\theta}}^{-1}[\nabla_x \nabla_\theta L(z, \hat{\theta})]\delta. \quad (4)$$

We thus have $\hat{\theta}_{z_\delta,-z} - \hat{\theta} \approx -\frac{1}{n} H_{\hat{\theta}}^{-1}[\nabla_x \nabla_\theta L(z, \hat{\theta})]\delta$. Differentiating w.r.t. δ and applying the chain rule gives us

$$\begin{aligned}\mathcal{I}_{\text{pert,loss}}(z, z_{\text{test}})^\top &\stackrel{\text{def}}{=} \nabla_\delta L(z_{\text{test}}, \hat{\theta}_{z_\delta,-z})^\top\Big|_{\delta=0} \quad (5)\\ &= -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_x \nabla_\theta L(z, \hat{\theta}).\end{aligned}$$

$\mathcal{I}_{\text{pert,loss}}(z, z_{\text{test}})^\top \delta$ tells us the approximate effect that $z \mapsto z + \delta$ has on the loss at z_{test}. By setting δ in the direction of $\mathcal{I}_{\text{pert,loss}}(z, z_{\text{test}})$, we can construct local perturbations of z that maximally increase the loss at z_{test}. In Section 5.2, we will use this to construct training-set attacks. Finally, we note that $\mathcal{I}_{\text{pert,loss}}(z, z_{\text{test}})$ can help us identify the features of z that are most responsible for the prediction on z_{test}.

2.3. Relation to Euclidean distance

To find the training points most relevant to a test point, it is common to look at its nearest neighbors in Euclidean

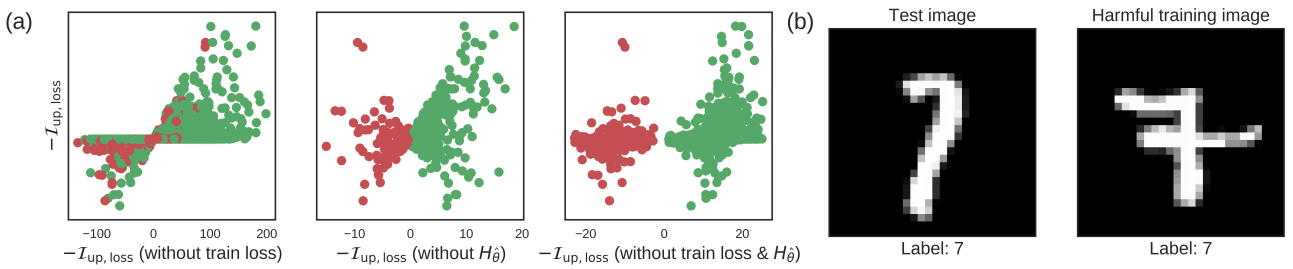

Figure 1. **Components of influence. (a)** What is the effect of the training loss and $H_{\hat{\theta}}^{-1}$ terms in $\mathcal{I}_{\text{up,loss}}$? Here, we plot $\mathcal{I}_{\text{up,loss}}$ against variants that are missing these terms and show that they are necessary for picking up the truly influential training points. For these calculations, we use logistic regression to distinguish 1's from 7's in MNIST (LeCun et al., 1998), picking an arbitrary test point z_{test}; similar trends hold across other test points. Green dots are train images of the same label as the test image (7) while red dots are 1's. **Left:** Without the train loss term, we overestimate the influence of many training points: the points near the y=0 line should have $\mathcal{I}_{\text{up,loss}}$ close to 0, but instead have high influence when we remove the train loss term. **Mid:** Without $H_{\hat{\theta}}^{-1}$, all green training points are helpful (removing each point increases test loss) and all red points are harmful (removing each point decreases test loss). This is because $\forall x, x \succeq 0$ (all pixel values are positive), so $x \cdot x_{\text{test}} \geq 0$, but it is incorrect: many harmful training points actually share the same label as z_{test}. See panel (b). **Right:** Without training loss or $H_{\hat{\theta}}^{-1}$, what is left is the scaled Euclidean inner product $y_{\text{test}} y \cdot \sigma(-y_{\text{test}} \theta^\top x_{\text{test}}) \cdot x_{\text{test}}^\top x$, which fails to accurately capture influence; the scatter plot deviates quite far from the diagonal. **(b)** The test image and a harmful training image with the same label. To the model, they look very different, so the presence of the training image makes the model think that the test image is less likely to be a 7. The Euclidean inner product does not pick up on these less intuitive, but important, harmful influences.

space (e.g., Ribeiro et al. (2016)); if all points have the same norm, this is equivalent to choosing x with the largest $x \cdot x_{\text{test}}$. For intuition, we compare this to $\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}})$ on a logistic regression model and show that influence is much more accurate at accounting for the effect of training.

Let $p(y \mid x) = \sigma(y\theta^\top x)$, with $y \in \{-1, 1\}$ and $\sigma(t) = \frac{1}{1+\exp(-t)}$. We seek to maximize the probability of the training set. For a training point $z = (x, y)$, $L(z, \theta) = \log(1 + \exp(-y\theta^\top x))$, $\nabla_\theta L(z, \theta) = -\sigma(-y\theta^\top x) yx$, and $H_\theta = \frac{1}{n} \sum_{i=1}^n \sigma(\theta^\top x_i)\sigma(-\theta^\top x_i) x_i x_i^\top$. From (2), $\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}})$ is:

$$-y_{\text{test}} y \cdot \sigma(-y_{\text{test}} \theta^\top x_{\text{test}}) \cdot \sigma(-y\theta^\top x) \cdot x_{\text{test}}^\top H_{\hat{\theta}}^{-1} x.$$

We highlight two key differences from $x \cdot x_{\text{test}}$. First, $\sigma(-y\theta^\top x)$ gives points with high training loss more influence, revealing that outliers can dominate the model parameters. Second, the weighted covariance matrix $H_{\hat{\theta}}^{-1}$ measures the "resistance" of the other training points to the removal of z; if $\nabla_\theta L(z, \hat{\theta})$ points in a direction of little variation, its influence will be higher since moving in that direction will not significantly increase the loss on other training points. As we show in Fig 1, these differences mean that influence functions capture the effect of model training much more accurately than nearest neighbors.

3. Efficiently Calculating Influence

There are two computational challenges to using $\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$. First, it requires forming and inverting $H_{\hat{\theta}} = \frac{1}{n} \sum_{i=1}^n \nabla_\theta^2 L(z_i, \hat{\theta})$, the Hessian of the empirical risk. With n training points and $\theta \in \mathbb{R}^p$, this requires $O(np^2 + p^3)$ operations, which is too expensive for models like deep neural networks with millions of parameters. Second, we often want to calculate $\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$ across all training points z_i.

The first problem is well-studied in second-order optimization. The idea is to avoid explicitly computing $H_{\hat{\theta}}^{-1}$; instead, we use implicit Hessian-vector products (HVPs) to efficiently approximate $s_{\text{test}} \stackrel{\text{def}}{=} H_{\hat{\theta}}^{-1} \nabla_\theta L(z_{\text{test}}, \hat{\theta})$ and then compute $\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = -s_{\text{test}} \cdot \nabla_\theta L(z, \hat{\theta})$. This also solves the second problem: for each test point of interest, we can precompute s_{test} and then efficiently compute $-s_{\text{test}} \cdot \nabla_\theta L(z_i, \hat{\theta})$ for each training point z_i.

We discuss two techniques for approximating s_{test}, both relying on the fact that the HVP of a single term in $H_{\hat{\theta}}$, $[\nabla_\theta^2 L(z_i, \hat{\theta})]v$, can be computed for arbitrary v in the same time that $\nabla_\theta L(z_i, \hat{\theta})$ would take, which is typically $O(p)$ (Pearlmutter, 1994).

Conjugate gradients (CG). The first technique is a standard transformation of matrix inversion into an optimization problem. Since $H_{\hat{\theta}} \succ 0$ by assumption, $H_{\hat{\theta}}^{-1} v \equiv \arg\min_t \{\frac{1}{2} t^\top H_{\hat{\theta}} t - v^\top t\}$. We can solve this with CG approaches that only require the evaluation of $H_{\hat{\theta}} t$, which takes $O(np)$ time, without explicitly forming $H_{\hat{\theta}}$. While an exact solution takes p CG iterations, in practice we can get a good approximation with fewer iterations; see Martens (2010) for more details.

Stochastic estimation. With large datasets, standard CG can be slow; each iteration still goes through all n training points. We use a method developed by Agarwal et al. (2016) to get an estimator that only samples a single point per iteration, which results in significant speedups.

Dropping the $\hat{\theta}$ subscript for clarity, let $H_j^{-1} \stackrel{\text{def}}{=} \sum_{i=0}^{j}(I - H)^i$, the first j terms in the Taylor expansion of H^{-1}. Rewrite this recursively as $H_j^{-1} = I + (I - H)H_{j-1}^{-1}$. From the validity of the Taylor expansion, $H_j^{-1} \to H^{-1}$ as $j \to \infty$.[2] The key is that at each iteration, we can substitute the full H with a draw from any unbiased (and faster-to-compute) estimator of H to form $\tilde{H}_j$. Since $\mathbb{E}[\tilde{H}_j^{-1}] = H_j^{-1}$, we still have $\mathbb{E}[\tilde{H}_j^{-1}] \to H^{-1}$.

In particular, we can uniformly sample z_i and use $\nabla_\theta^2 L(z_i, \hat{\theta})$ as an unbiased estimator of H. This gives us the following procedure: uniformly sample t points $z_{s_1}, \ldots, z_{s_t}$ from the training data; define $\tilde{H}_0^{-1}v = v$; and recursively compute $\tilde{H}_j^{-1}v = v + \big(I - \nabla_\theta^2 L(z_{s_j}, \hat{\theta})\big)\tilde{H}_{j-1}^{-1}v$, taking $\tilde{H}_t^{-1}v$ as our final unbiased estimate of $H^{-1}v$. We pick t to be large enough such that $\tilde{H}_t$ stabilizes, and to reduce variance we repeat this procedure r times and average results. Empirically, we found this significantly faster than CG.

We note that the original method of Agarwal et al. (2016) dealt only with generalized linear models, for which $[\nabla_\theta^2 L(z_i, \hat{\theta})]v$ can be efficiently computed in $O(p)$ time. In our case, we rely on Pearlmutter (1994)'s more general algorithm for fast HVPs, described above, to achieve the same time complexity.[3]

With these techniques, we can compute $\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$ on all training points z_i in $O(np + rtp)$ time; we show in Section 4.1 that empirically, choosing $rt = O(n)$ gives accurate results. Similarly, we compute $\mathcal{I}_{\text{pert,loss}}(z_i, z_{\text{test}})^\top = -\frac{1}{n}\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_x \nabla_\theta L(z_i, \hat{\theta})$ with two matrix-vector products: we first compute s_{test}, then $s_{\text{test}}^\top \nabla_x \nabla_\theta L(z_i, \hat{\theta})$, with the same HVP trick. These computations are easy to implement in auto-grad systems like TensorFlow (Abadi et al., 2015) and Theano (Theano D. Team, 2016), as users need only specify L; the rest is automatically handled.

4. Validation and Extensions

Recall that influence functions are asymptotic approximations of leave-one-out retraining under the assumptions that (i) the model parameters $\hat{\theta}$ minimize the empirical risk, and that (ii) the empirical risk is twice-differentiable and strictly convex. Here, we empirically show that influence functions are accurate approximations (Section 4.1) that provide useful information even when these assumptions are violated (Sections 4.2, 4.3).

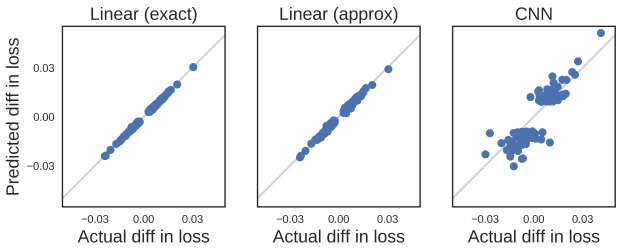

Figure 2. **Influence matches leave-one-out retraining.** We arbitrarily picked a wrongly-classified test point z_{test}, but this trend held more broadly. These results are from 10-class MNIST. **Left:** For each of the 500 training points z with largest $\left|\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}})\right|$, we plotted $-\frac{1}{n} \cdot \mathcal{I}_{\text{up,loss}}(z, z_{\text{test}})$ against the actual change in test loss after removing that point and retraining. The inverse HVP was solved exactly with CG. **Mid:** Same, but with the stochastic approximation. **Right:** The same plot for a CNN, computed on the 100 most influential points with CG. For the actual difference in loss, we removed each point and retrained from $\tilde{\theta}$ for 30k steps.

4.1. Influence functions vs. leave-one-out retraining

Influence functions assume that the weight on a training point is changed by an infinitesimally small ϵ. To investigate the accuracy of using influence functions to approximate the effect of removing a training point and retraining, we compared $-\frac{1}{n}\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}})$ with $L(z_{\text{test}}, \hat{\theta}_{-z}) - L(z_{\text{test}}, \hat{\theta})$ (i.e., actually doing leave-one-out retraining). With a logistic regression model on 10-class MNIST,[4] the predicted and actual changes matched closely (Fig 2-Left).

The stochastic approximation from Agarwal et al. (2016) was also accurate with $r = 10$ repeats and $t = 5,000$ iterations (Fig 2-Mid). Since each iteration only requires one HVP $[\nabla_\theta^2 L(z_i, \hat{\theta})]v$, this runs quickly: in fact, we accurately estimated $H^{-1}v$ without even looking at every data point, since $n = 55,000 > rt$. Surprisingly, even $r = 1$ worked; while results were noisier, it was still able to identify the most influential points.

4.2. Non-convexity and non-convergence

In Section 2, we took $\hat{\theta}$ as the global minimum. In practice, if we obtain our parameters $\tilde{\theta}$ by running SGD with early stopping or on non-convex objectives, $\tilde{\theta} \neq \hat{\theta}$. As a result, $H_{\tilde{\theta}}$ could have negative eigenvalues. We show that influence functions on $\tilde{\theta}$ still give meaningful results in practice.

Our approach is to form a convex quadratic approximation of the loss around $\tilde{\theta}$, i.e., $\tilde{L}(z, \theta) = L(z, \tilde{\theta}) +$

[2] We assume w.l.o.g. that $\forall i, \nabla_\theta^2 L(z_i, \hat{\theta}) \preccurlyeq I$; if this is not true, we can scale the loss down without affecting the parameters. In some cases, we can get an upper bound on $\nabla_\theta^2 L(z_i, \hat{\theta})$ (e.g., for linear models and bounded input), which makes this easy. Otherwise, we treat the scaling as a separate hyperparameter and tune it such that the Taylor expansion converges.

[3] To increase stability, especially with non-convex models (see Section 4.2), we can also sample a mini-batch of training points at each iteration, instead of relying on a single training point.

[4] We trained with L-BFGS (Liu & Nocedal, 1989), with L_2 regularization of 0.01, $n = 55,000$, and $p = 7,840$ parameters.

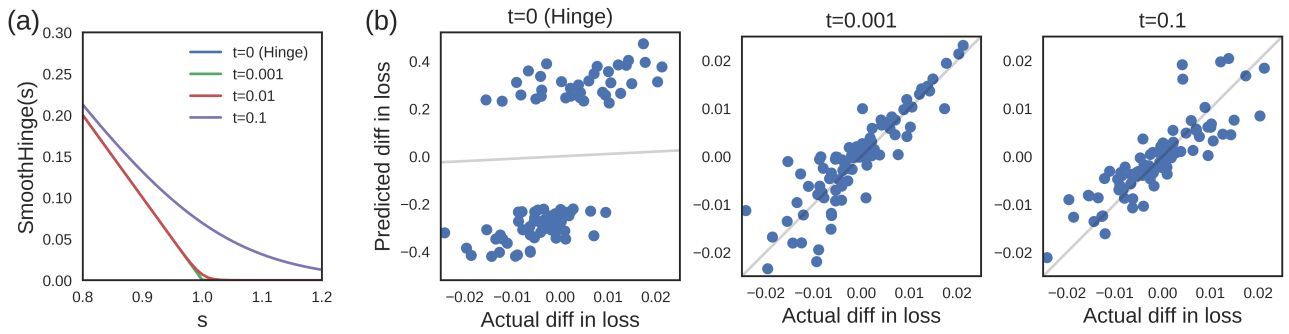

Figure 3. **Smooth approximations to the hinge loss. (a)** By varying t, we can approximate the hinge loss with arbitrary accuracy: the green and blue lines are overlaid on top of each other. **(b)** Using a random, wrongly-classified test point, we compared the predicted vs. actual differences in loss after leave-one-out retraining on the 100 most influential training points. A similar trend held for other test points. The SVM objective is to minimize $0.005\,\|w\|_2^2 + \frac{1}{n}\sum_i \text{Hinge}(y_i w^\top x_i)$. **Left:** Influence functions were unable to accurately predict the change, overestimating its magnitude considerably. **Mid:** Using $\text{SmoothHinge}(\cdot, 0.001)$ let us accurately predict the change in the hinge loss after retraining. **Right:** Correlation remained high over a wide range of t, though it degrades when t is too large. When $t = 0.001$, Pearson's R = 0.95; when $t = 0.1$, Pearson's R = 0.91.

$\nabla L(z,\tilde{\theta})^\top(\theta-\tilde{\theta}) + \frac{1}{2}(\theta-\tilde{\theta})^\top(H_{\tilde{\theta}}+\lambda I)(\theta-\tilde{\theta})$. Here, λ is a damping term that we add if $H_{\tilde{\theta}}$ has negative eigenvalues; this corresponds to adding L_2 regularization on θ. We then calculate $\mathcal{I}_{\text{up,loss}}$ using $\tilde{L}$. If $\tilde{\theta}$ is close to a local minimum, this is correlated with the result of taking a Newton step from $\tilde{\theta}$ after removing ϵ weight from z (see appendix B).

We checked the behavior of $\mathcal{I}_{\text{up,loss}}$ in a non-convergent, non-convex setting by training a convolutional neural network for 500k iterations.[5] The model had not converged and $H_{\tilde{\theta}}$ was not PD, so we added a damping term with $\lambda = 0.01$. Even in this difficult setting, the predicted and actual changes in loss were highly correlated (Pearson's R = 0.86, Fig 2-Right).

4.3. Non-differentiable losses

What happens when the derivatives of the loss, $\nabla_\theta L$ and $\nabla^2_\theta L$, do not exist? In this section, we show that influence functions computed on smooth approximations to non-differentiable losses can predict the behavior of the original, non-differentiable loss under leave-one-out retraining. The robustness of this approximation suggests that we can train non-differentiable models and swap out non-differentiable components for smoothed versions for the purposes of calculating influence.

To see this, we trained a linear SVM on the same 1s vs. 7s MNIST task in Section 2.3. This involves minimizing $\text{Hinge}(s) = \max(0, 1-s)$; this simple piecewise linear function is similar to ReLUs, which cause non-differentiability in neural networks. We set the derivatives at the hinge to 0 and calculated $\mathcal{I}_{\text{up,loss}}$. As one might expect, this was inaccurate (Fig 3b-Left): the second derivative carries no information about how close a support vector z is to the hinge, so the quadratic approximation of $L(z,\hat{\theta})$ is linear (up to regularization), which leads to $\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}})$ overestimating the influence of z.

For the purposes of calculating influence, we approximated $\text{Hinge}(s)$ with $\text{SmoothHinge}(s,t) = t\log(1+\exp(\frac{1-s}{t}))$, which approaches the hinge loss as $t \to 0$ (Fig 3a). Using the same SVM weights as before, we found that calculating $\mathcal{I}_{\text{up,loss}}$ using $\text{SmoothHinge}(s, 0.001)$ closely matched the actual change due to retraining in the original $\text{Hinge}(s)$ (Pearson's R = 0.95; Fig 3b-Mid) and remained accurate over a wide range of t (Fig 3b-Right).

5. Use Cases of Influence Functions

5.1. Understanding model behavior

By telling us the training points "responsible" for a given prediction, influence functions reveal insights about how models rely on and extrapolate from the training data. In this section, we show that two models can make the same correct predictions but get there in very different ways.

We compared (a) the state-of-the-art Inception v3 network (Szegedy et al., 2016) with all but the top layer frozen[6] to (b) an SVM with an RBF kernel on a dog vs. fish image classification dataset we extracted from ImageNet (Russakovsky et al., 2015), with 900 training examples for each class. Freezing neural networks in this way is not uncom-

[5] The network had 7 sets of convolutional layers with $\tanh(\cdot)$ non-linearities, modeled after the all-convolutional network from (Springenberg et al., 2014). For speed, we used 10% of the MNIST training set and only 2,616 parameters, since repeatedly retraining the network was expensive. Training was done with mini-batches of 500 examples and the Adam optimizer (Kingma & Ba, 2014). The model had not converged after 500k iterations; training it for another 500k iterations, using a full training pass for each iteration, reduced train loss from 0.14 to 0.12.

[6] We used pre-trained weights from Keras (Chollet, 2015).

mon in computer vision and is equivalent to training a logistic regression model on the bottleneck features (Donahue et al., 2014). We picked a test image both models got correct (Fig 4-Top) and used $\mathrm{SmoothHinge}(\cdot, 0.001)$ to compute the influence for the SVM.

As expected, $\mathcal{I}_{\text{up,loss}}$ in the RBF SVM varied inversely with raw pixel distance, with training images far from the test image in pixel space having almost no influence. The Inception influences were much less correlated with distance in pixel space (Fig 4-Left). Looking at the two most helpful images (most positive $-\mathcal{I}_{\text{up,loss}}$) for each model in Fig 4-Right, we see that the Inception network picked up on the distinctive characteristics of clownfish, whereas the RBF SVM pattern-matched training images superficially.

Moreover, in the RBF SVM, fish (green points) close to the test image were mostly helpful, while dogs (red) were mostly harmful, with the RBF acting as a soft nearest neighbor function (Fig 4-Left). In contrast, in the Inception network, fish and dogs could be helpful or harmful for correctly classifying the test image as a fish; in fact, some of the most helpful training images were dogs that, to the model, looked very different from the test fish (Fig 4-Top).

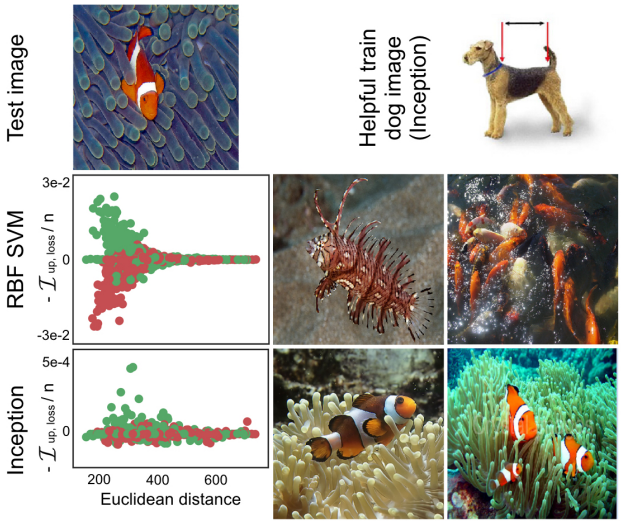

Figure 4. **Inception vs. RBF SVM. Bottom left:** $-\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}})$ vs. $\|z - z_{\text{test}}\|_2^2$. Green dots are fish and red dots are dogs. **Bottom right:** The two most helpful training images, for each model, on the test. **Top right:** An image of a dog in the training set that helped the Inception model correctly classify the test image as a fish.

5.2. Adversarial training examples

In this section, we show that models that place a lot of influence on a small number of points can be vulnerable to training input perturbations, posing a serious security risk in real-world ML systems where attackers can influence the training data (Huang et al., 2011). Recent work has generated adversarial *test* images that are visually indistinguishable from real test images but completely fool a classifier (Goodfellow et al., 2015; Moosavi-Dezfooli et al., 2016). We demonstrate that influence functions can be used to craft adversarial *training* images that are similarly visually-indistinguishable and can flip a model's prediction on a separate test image. To the best of our knowledge, this is the first proof-of-concept that visually-indistinguishable training attacks can be executed on otherwise highly-accurate neural networks.

The key idea is that $\mathcal{I}_{\text{pert,loss}}(z, z_{\text{test}})$ tells us how to modify training point z to most increase the loss on z_{test}. Concretely, for a target test image z_{test}, we can construct $\tilde{z}_i$, an adversarial version of a training image z_i, by initializing $\tilde{z}_i := z_i$ and then iterating $\tilde{z}_i := \Pi(\tilde{z}_i + \alpha\,\mathrm{sign}(\mathcal{I}_{\text{pert,loss}}(\tilde{z}_i, z_{\text{test}})))$, where α is the step size and Π projects onto the set of valid images that share the same 8-bit representation with z_i. After each iteration, we retrain the model. This is an iterated, training-set analogue of the methods used by, e.g., Goodfellow et al. (2015); Moosavi-Dezfooli et al. (2016) for test-set attacks.

We tested these training attacks on the same Inception network on dogs vs. fish from Section 5.1, choosing this pair of animals to provide a stark contrast between the classes. We set $\alpha = 0.02$ and ran the attack for 100 iterations on each test image. As before, we froze all but the top layer for training; note that computing $\mathcal{I}_{\text{pert,loss}}$ still involves differentiating through the entire network. Originally, the model correctly classified 591 / 600 test images. For each of these 591 test images, considered separately, we tried to find a visually-indistinguishable perturbation (i.e., same 8-bit representation) to a single training image, out of 1,800 total training images, that would flip the model's prediction. We were able to do this on 335 (57%) of the 591 test images. By perturbing 2 training images for each test image, we could flip predictions on 77% of the 591 test images; and if we perturbed 10 training images, we could flip all but 1 of the 591. The above results are from attacking each test image separately, i.e., using a different training set to attack each test image. We also tried to attack multiple test images simultaneously by increasing their average loss, and found that single training image perturbations could simultaneously flip multiple test predictions as well (Fig 5).

We make three observations about these attacks. First, though the change in pixel values is small, the change in the final Inception feature layer is significantly larger: using L_2 distance in pixel space, the training values change by less than 1% of the mean distance of a training point to its class centroid, whereas in Inception feature space, the change is on the same order as the mean distance. This leaves open the possibility that our attacks, while visually-imperceptible, can be detected by examining the feature space. Second, the attack tries to perturb the training ex-

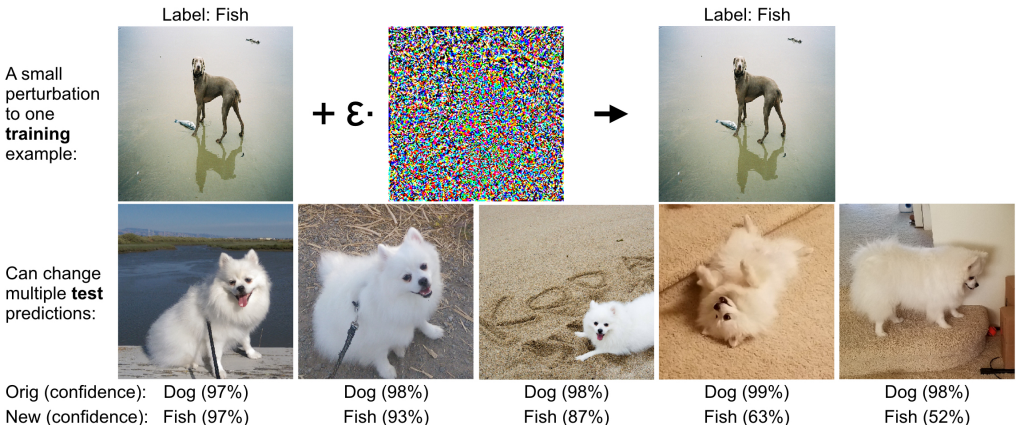

Figure 5. **Training-set attacks.** We targeted a set of 30 test images featuring the first author's dog in a variety of poses and backgrounds. By maximizing the average loss over these 30 images, we created a visually-imperceptible change to the particular training image (shown on top) that flipped predictions on 16 test images.

ample in a direction of low variance, causing the model to overfit in that direction and consequently incorrectly classify the test images; we expect attacking to be harder as the number of training examples grows. Third, ambiguous or mislabeled training images are effective points to attack: the model has low confidence and thus high loss on them, making them highly influential (recall Section 2.3). For example, the image in Fig 5 contains both a dog and a fish and is highly ambiguous; as a result, it is the training example that the model is least confident on (with a confidence of 77%, compared to the next lowest confidence of 90%).

This attack is mathematically equivalent to the gradient-based training set attacks explored by Biggio et al. (2012); Mei & Zhu (2015b) and others in the context of different models. Biggio et al. (2012) constructed a dataset poisoning attack against a linear SVM on a two-class MNIST task, but had to modify the training points in an obviously distinguishable way to be effective. Measuring the magnitude of $\mathcal{I}_{\text{pert,loss}}$ gives model developers a way of quantifying how vulnerable their models are to training-set attacks.

5.3. Debugging domain mismatch

Domain mismatch — where the training distribution does not match the test distribution — can cause models with high training accuracy to do poorly on test data (Ben-David et al., 2010). We show that influence functions can identify the training examples most responsible for the errors, helping model developers identify domain mismatch.

As a case study, we predicted whether a patient would be readmitted to hospital. Domain mismatches are common in biomedical data, e.g., different hospitals serve different populations, and models trained on one population can do poorly on another (Kansagara et al., 2011). We used logistic regression to predict readmission with a balanced training dataset of 20K diabetic patients from 100+ US hospitals, each represented by 127 features (Strack et al., 2014).[7] 3 out of the 24 children under age 10 in this dataset were re-admitted. To induce a domain mismatch, we filtered out 20 children who were not re-admitted, leaving 3 out of 4 re-admitted. This caused the model to wrongly classify many children in the test set. Our aim is to identify the 4 children in the training set as being "responsible" for these errors.

As a baseline, we tried the common practice of looking at the learned parameters $\hat{\theta}$ to see if the indicator variable for being a child was obviously different. However, this did not work: 14/127 features had a larger coefficient.

Picking a random child z_{test} that the model got wrong, we calculated $-\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$ for each training point z_i. This clearly highlighted the 4 training children, each of whom were 30-40 times as influential as the next most influential examples. The 1 child in the training set who was not readmitted had a very positive influence, while the other 3 had very negative influences. Moreover, calculating $\mathcal{I}_{\text{pert,loss}}$ on these 4 children showed that the 'child' indicator variable contributed significantly to the magnitude of $\mathcal{I}_{\text{up,loss}}$.

5.4. Fixing mislabeled examples

Labels in the real world are often noisy, especially if crowdsourced (Frénay & Verleysen, 2014), and can even be adversarially corrupted. Even if a human expert could recognize wrongly labeled examples, it is impossible in many applications to manually review all of the training data. We show that influence functions can help human experts prioritize their attention, allowing them to inspect only the examples that actually matter.

The key idea is to flag the training points that exert the most influence on the model. Because we do not have access to the test set, we measure the influence of z_i with $\mathcal{I}_{\text{up,loss}}(z_i, z_i)$, which approximates the error incurred on z_i if we remove z_i from the training set.

Our case study is email spam classification, which relies

[7] Hospital readmission was defined as whether a patient would be readmitted within the next 30 days. Features were demographic (e.g., age, race, gender), administrative (e.g., length of hospital stay), or medical (e.g., test results).

on user-provided labels and is also vulnerable to adversarial attack (Biggio et al., 2011). We flipped the labels of a random 10% of the training data and then simulated manually inspecting a fraction of the training points, correcting them if they had been flipped. Using influence functions to prioritize the training points to inspect allowed us to repair the dataset (Fig 6, blue) without checking too many points, outperforming the baselines of checking points with the highest train loss (Fig 6, green) or at random (Fig 6, red). No method had access to the test data.

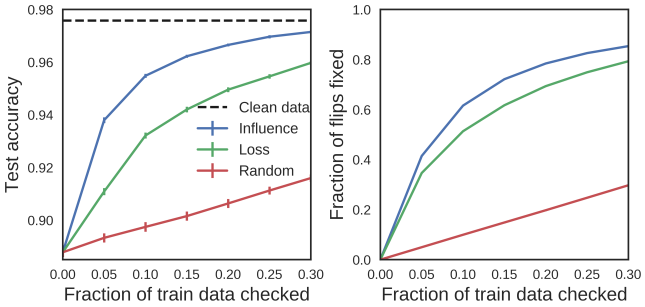

Figure 6. **Fixing mislabeled examples.** Plots of how test accuracy (left) and the fraction of flipped data detected (right) change with the fraction of train data checked, using different algorithms for picking points to check. Error bars show the std. dev. across 40 repeats of this experiment, with a different subset of labels flipped in each; error bars on the right are too small to be seen. These results are on the Enron1 spam dataset (Metsis et al., 2006), with 4,147 training and 1,035 test examples; we trained logistic regression on a bag-of-words representation of the emails.

6. Related Work

The use of influence-based diagnostics originated in statistics in the 70s and 80s, driven by seminal papers by Cook and others (Cook, 1977; Cook & Weisberg, 1980; 1982), though similar ideas appeared even earlier in other forms, e.g., the infinitesimal jackknife (Jaeckel, 1972). Earlier work focused on removing training points from linear models, with later work extending this to more general models and a wider variety of perturbations (Cook, 1986; Thomas & Cook, 1990; Chatterjee & Hadi, 1986; Wei et al., 1998). Most of this prior work focused on experiments with small datasets, e.g., $n = 24$ and $p = 10$ in Cook & Weisberg (1980), with special attention therefore paid to exact solutions, or if not possible, characterizations of the error terms.

Influence functions have not been used much in the ML literature, with some exceptions. Christmann & Steinwart (2004); Debruyne et al. (2008); Liu et al. (2014) use influence functions to study model robustness and to do fast cross-validation in kernel methods. Wojnowicz et al. (2016) uses matrix sketching to estimate Cook's distance, which is closely related to influence; they focus on prioritizing training points for human attention and derive methods specific to generalized linear models.

As noted in Section 5.2, our training-set attack is mathematically equivalent to an approach first explored by Biggio et al. (2012) in the context of SVMs, with follow-up work extending the framework and applying it to linear and logistic regression (Mei & Zhu, 2015b), topic modeling (Mei & Zhu, 2015a), and collaborative filtering (Li et al., 2016a). These papers derived the attack directly from the KKT conditions without considering influence, though for continuous data, the end result is equivalent. Influence functions additionally let us consider attacks on discrete data (Section 2.2), but we have not tested this empirically. Our work connects the literature on training-set attacks with work on "adversarial examples" (Goodfellow et al., 2015; Moosavi-Dezfooli et al., 2016), visually-imperceptible perturbations on test inputs.

In contrast to training-set attacks, Cadamuro et al. (2016) consider the task of taking an incorrect test prediction and finding a small subset of training data such that changing the labels on this subset makes the prediction correct. They provide a solution for OLS and Gaussian process models when the labels are continuous. Our work with influence functions allow us to solve this problem in a much larger range of models and in datasets with discrete labels.

7. Discussion

We have discussed a variety of applications, from creating training-set attacks to debugging models and fixing datasets. Underlying each of these applications is a common tool, the influence function, which is based on a simple idea — we can better understand model behavior by looking at how it was derived from its training data.

At their core, influence functions measure the effect of local changes: what happens when we upweight a point by an infinitesimally-small ϵ? This locality allows us to derive efficient closed-form estimates, and as we show, they can be surprisingly effective. However, we might want to ask about more global changes, e.g., how does a subpopulation of patients from this hospital affect the model? Since influence functions depend on the model not changing too much, how to tackle this is an open question.

It seems inevitable that high-performing, complex, black-box models will become increasingly prevalent and important. We hope that the approach presented here — of looking at the model through the lens of the training data — will become a standard part of the toolkit of developing, understanding, and diagnosing machine learning.

The code and data for replicating our experiments is available on GitHub `http://bit.ly/gt-influence` and Codalab `http://bit.ly/cl-influence`.

A. Deriving the influence function $\mathcal{I}_{\text{up,params}}$

For completeness, we provide a standard derivation of the influence function $\mathcal{I}_{\text{up,params}}$ in the context of loss minimization (M-estimation). This derivation is based on asymptotic arguments and is not fully rigorous; see van der Vaart (1998) and other statistics textbooks for a more thorough treatment.

Recall that $\hat{\theta}$ minimizes the empirical risk:

$$R(\theta) \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta). \tag{6}$$

We further assume that R is twice-differentiable and strictly convex in θ, i.e.,

$$H_{\hat{\theta}} \stackrel{\text{def}}{=} \nabla^2 R(\hat{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta^2 L(z_i, \hat{\theta}) \tag{7}$$

exists and is positive definite. This guarantees the existence of $H_{\hat{\theta}}^{-1}$, which we will use in the subsequent derivation.

The perturbed parameters $\hat{\theta}_{\epsilon,z}$ can be written as

$$\hat{\theta}_{\epsilon,z} = \arg\min_{\theta\in\Theta} \{R(\theta) + \epsilon L(z,\theta)\}. \tag{8}$$

Define the parameter change $\Delta_\epsilon = \hat{\theta}_{\epsilon,z} - \hat{\theta}$, and note that, as $\hat{\theta}$ doesn't depend on ϵ, the quantity we seek to compute can be written in terms of it:

$$\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon} = \frac{d\Delta_\epsilon}{d\epsilon}. \tag{9}$$

Since $\hat{\theta}_{\epsilon,z}$ is a minimizer of (8), let us examine its first-order optimality conditions:

$$0 = \nabla R(\hat{\theta}_{\epsilon,z}) + \epsilon \nabla L(z, \hat{\theta}_{\epsilon,z}). \tag{10}$$

Next, since $\hat{\theta}_{\epsilon,z} \to \hat{\theta}$ as $\epsilon \to 0$, we perform a Taylor expansion of the right-hand side:

$$0 \approx \left[\nabla R(\hat{\theta}) + \epsilon \nabla L(z, \hat{\theta})\right] + \left[\nabla^2 R(\hat{\theta}) + \epsilon \nabla^2 L(z, \hat{\theta})\right] \Delta_\epsilon, \tag{11}$$

where we have dropped $o(\|\Delta_\epsilon\|)$ terms.

Solving for Δ_ϵ, we get:

$$\Delta_\epsilon \approx -\left[\nabla^2 R(\hat{\theta}) + \epsilon \nabla^2 L(z, \hat{\theta})\right]^{-1} \left[\nabla R(\hat{\theta}) + \epsilon \nabla L(z, \hat{\theta})\right]. \tag{12}$$

Since $\hat{\theta}$ minimzes R, we have $\nabla R(\hat{\theta}) = 0$. Keeping only $O(\epsilon)$ terms, we have

$$\Delta_\epsilon \approx -\nabla^2 R(\hat{\theta})^{-1} \nabla L(z, \hat{\theta}) \epsilon. \tag{13}$$

Combining with (7) and (9), we conclude that:

$$\left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1} \nabla L(z, \hat{\theta}) \tag{14}$$

$$\stackrel{\text{def}}{=} \mathcal{I}_{\text{up,params}}(z). \tag{15}$$

B. Influence at non-convergence

Consider a training point z. When the model parameters $\tilde{\theta}$ are close to but not at a local minimum, $\mathcal{I}_{\text{up,params}}(z)$ is approximately equal to a constant (which does not depend on z) plus the change in parameters after upweighting z and then taking a single Newton step from $\tilde{\theta}$. The high-level idea is that even though the gradient of the empirical risk at $\tilde{\theta}$ is not 0, the Newton step from $\tilde{\theta}$ can be decomposed into a component following the existing gradient (which does not depend on the choice of z) and a second component responding to the upweighted z (which $\mathcal{I}_{\text{up,params}}(z)$ tracks).

Let $g \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta L(z_i, \tilde{\theta})$ be the gradient of the empirical risk at $\tilde{\theta}$; since $\tilde{\theta}$ is not a local minimum, $g \neq 0$. After upweighting z by ϵ, the gradient at $\tilde{\theta}$ goes from $g \mapsto g + \epsilon \nabla_\theta L(z, \tilde{\theta})$, and the empirical Hessian goes from $H_{\tilde{\theta}} \mapsto H_{\tilde{\theta}} + \epsilon \nabla_\theta^2 L(z, \tilde{\theta})$. A Newton step from $\tilde{\theta}$ therefore changes the parameters by:

$$N_{\epsilon,z} \stackrel{\text{def}}{=} -\left[H_{\tilde{\theta}} + \epsilon \nabla_\theta^2 L(z, \tilde{\theta})\right]^{-1} \left[g + \epsilon \nabla_\theta L(z, \tilde{\theta})\right]. \tag{16}$$

Ignoring terms in ϵg, ϵ^2, and higher, we get $N_{\epsilon,z} \approx -H_{\tilde{\theta}}^{-1}\left(g + \epsilon \nabla_\theta L(z, \tilde{\theta})\right)$. Therefore, the actual change due to a Newton step $N_{\epsilon,z}$ is equal to a constant $-H_{\tilde{\theta}}^{-1} g$ (that doesn't depend on z) plus ϵ times $\mathcal{I}_{\text{up,params}}(z) = -H_{\tilde{\theta}}^{-1} \nabla_\theta L(z, \tilde{\theta})$ (which captures the contribution of z).

Acknowledgements

We thank Jacob Steinhardt, Zhenghao Chen, and Hongseok Namkoong for helpful discussions and comments. This work was supported by a Future of Life Research Award and a Microsoft Research Faculty Fellowship.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G.,

Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2015.

Adler, P., Falk, C., Friedler, S. A., Rybeck, G., Scheidegger, C., Smith, B., and Venkatasubramanian, S. Auditing black-box models for indirect influence. *arXiv preprint arXiv:1602.07043*, 2016.

Agarwal, N., Bullins, B., and Hazan, E. Second order stochastic optimization in linear time. *arXiv preprint arXiv:1602.03943*, 2016.

Amershi, S., Chickering, M., Drucker, S. M., Lee, B., Simard, P., and Suh, J. Modeltracker: Redesigning performance analysis tools for machine learning. In *Conference on Human Factors in Computing Systems (CHI)*, pp. 337–346, 2015.

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. A theory of learning from different domains. *Machine Learning*, 79(1):151–175, 2010.

Biggio, B., Nelson, B., and Laskov, P. Support vector machines under adversarial label noise. *ACML*, 20:97–112, 2011.

Biggio, B., Nelson, B., and Laskov, P. Poisoning attacks against support vector machines. In *International Conference on Machine Learning (ICML)*, pp. 1467–1474, 2012.

Cadamuro, G., Gilad-Bachrach, R., and Zhu, X. Debugging machine learning models. In *ICML Workshop on Reliable Machine Learning in the Wild*, 2016.

Chatterjee, S. and Hadi, A. S. Influential observations, high leverage points, and outliers in linear regression. *Statistical Science*, pp. 379–393, 1986.

Chollet, F. *Keras*, 2015.

Christmann, A. and Steinwart, I. On robustness properties of convex risk minimization methods for pattern recognition. *Journal of Machine Learning Research (JMLR)*, 5(0):1007–1034, 2004.

Cook, R. D. Detection of influential observation in linear regression. *Technometrics*, 19:15–18, 1977.

Cook, R. D. Assessment of local influence. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 133–169, 1986.

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

Cook, R. D. and Weisberg, S. *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

Datta, A., Sen, S., and Zick, Y. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pp. 598–617, 2016.

Debruyne, M., Hubert, M., and Suykens, J. A. Model selection in kernel based regression using the influence function. *Journal of Machine Learning Research (JMLR)*, 9 (0):2377–2400, 2008.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning (ICML)*, volume 32, pp. 647–655, 2014.

Frénay, B. and Verleysen, M. Classification in the presence of label noise: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25:845–869, 2014.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.

Goodman, B. and Flaxman, S. European union regulations on algorithmic decision-making and a "right to explanation". *arXiv preprint arXiv:1606.08813*, 2016.

Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., and Tygar, J. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pp. 43–58, 2011.

Jaeckel, L. A. The infinitesimal jackknife. *Unpublished memorandum, Bell Telephone Laboratories, Murray Hill, NJ*, 1972.

Kansagara, D., Englander, H., Salanitro, A., Kagen, D., Theobald, C., Freeman, M., and Kripalani, S. Risk prediction models for hospital readmission: a systematic review. *JAMA*, 306(15):1688–1698, 2011.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

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

Li, B., Wang, Y., Singh, A., and Vorobeychik, Y. Data poisoning attacks on factorization-based collaborative filtering. In *Advances in Neural Information Processing Systems (NIPS)*, 2016a.

Li, J., Monroe, W., and Jurafsky, D. Understanding neural networks through representation erasure. *arXiv preprint arXiv:1612.08220*, 2016b.

Liu, D. C. and Nocedal, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

Liu, Y., Jiang, S., and Liao, S. Efficient approximation of cross-validation for kernel methods using Bouligand influence function. In *International Conference on Machine Learning (ICML)*, pp. 324–332, 2014.

Martens, J. Deep learning via hessian-free optimization. In *International Conference on Machine Learning (ICML)*, pp. 735–742, 2010.

Mei, S. and Zhu, X. The security of latent Dirichlet allocation. In *Artificial Intelligence and Statistics (AISTATS)*, 2015a.

Mei, S. and Zhu, X. Using machine teaching to identify optimal training-set attacks on machine learners. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2015b.

Metsis, V., Androutsopoulos, I., and Paliouras, G. Spam filtering with naive Bayes – which naive Bayes? In *CEAS*, volume 17, pp. 28–69, 2006.

Moosavi-Dezfooli, S., Fawzi, A., and Frossard, P. Deepfool: a simple and accurate method to fool deep neural networks. In *Computer Vision and Pattern Recognition (CVPR)*, pp. 2574–2582, 2016.

Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural Computation*, 6(1):147–160, 1994.

Ribeiro, M. T., Singh, S., and Guestrin, C. "why should I trust you?": Explaining the predictions of any classifier. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3): 211–252, 2015.

Shrikumar, A., Greenside, P., Shcherbina, A., and Kundaje, A. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Strack, B., DeShazo, J. P., Gennings, C., Olmo, J. L., Ventura, S., Cios, K. J., and Clore, J. N. Impact of HbA1c measurement on hospital readmission rates: analysis of 70,000 clinical database patient records. *BioMed Research International*, 2014, 2014.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the Inception architecture for computer vision. In *Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

Theano D. Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

Thomas, W. and Cook, R. D. Assessing influence on predictions from generalized linear models. *Technometrics*, 32(1):59–65, 1990.

van der Vaart, A. W. *Asymptotic statistics*. Cambridge University Press, 1998.

Wei, B., Hu, Y., and Fung, W. Generalized leverage and its applications. *Scandinavian Journal of Statistics*, 25: 25–37, 1998.

Wojnowicz, M., Cruz, B., Zhao, X., Wallace, B., Wolff, M., Luan, J., and Crable, C. "Influence sketching": Finding influential samples in large-scale regressions. *arXiv preprint arXiv:1611.05923*, 2016.