Understanding Black-box Predictions via Influence Functions

ICML 2017 Best Paper

Yufeng Ma

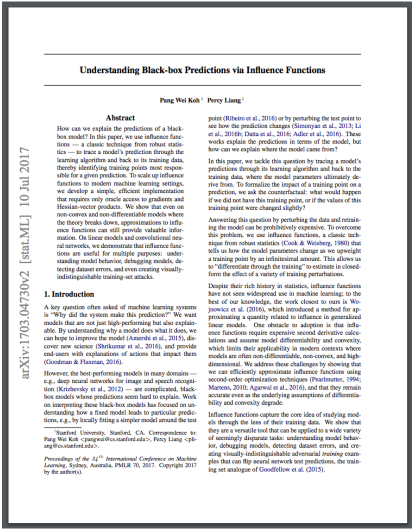
Digital Library Research Laboratory
Computer Science
Virginia Tech

yufengma@vt.edu

September 20, 2017

Paper and Author Infos

Paper and Author Infos



arXiv:1703.04730v2 [stat.ML] 10 Jul 2017

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.

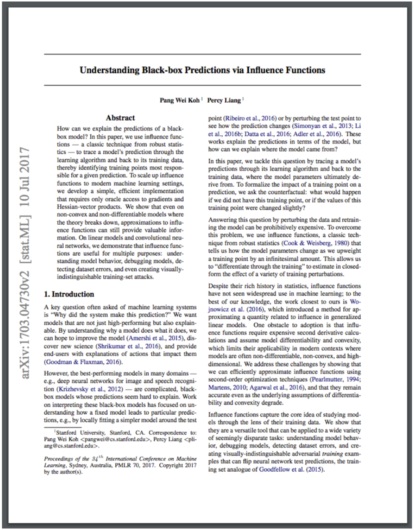
Despite their rich history in statistics, influence functions have not seen widespread use in machine learning; to the best of our knowledge, the work closest to ours is Wojnowicz et al. (2016), which introduced a method for approximating a quantity related to influence in generalized linear models. 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 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

Paper and Author Infos



arXiv:1703.04730v2 [stat.ML] 10 Jul 2017

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>.

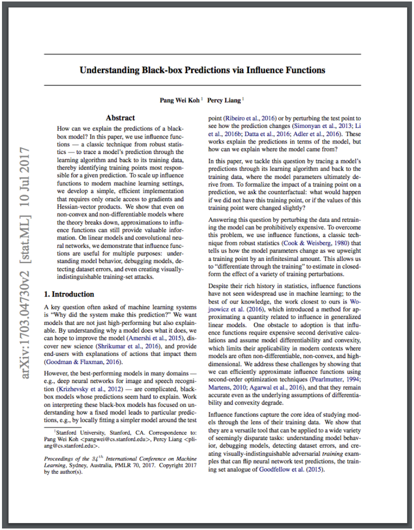
Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

ICML 2017 Best Paper Award

The 34th International Conference on Machine Learning

- Pang Wei Koh
- Percy Liang

Paper and Author Infos



arXiv:1703.04730v2 [stat.ML] 10 Jul 2017

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 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

ICML 2017 Best Paper Award

The 34th International Conference on Machine Learning

- Pang Wei Koh
- Percy Liang

Affiliations:

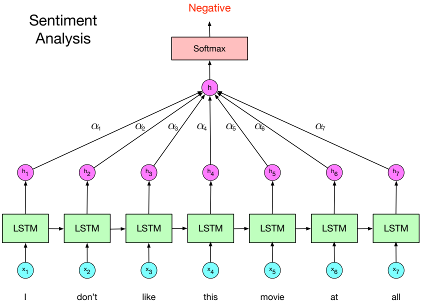
Computer Science, Stanford University

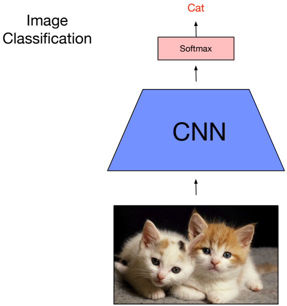
- Artificial Intelligence Lab
- Natural Language Processing Group
- Statistical Machine Learning Group

Overview

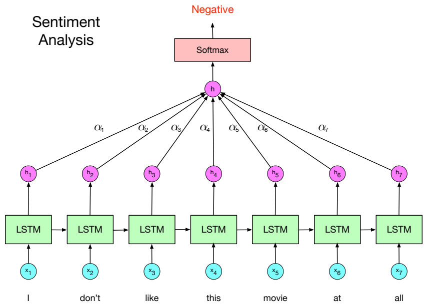
Introduction

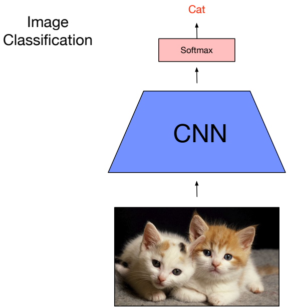
Introduction





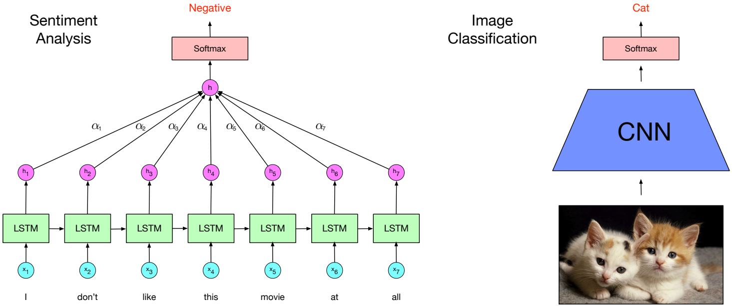
Introduction





- Why did neural networks make such predictions?
- Where does the model come from?
- Can we explain predictions in terms of training points?

Introduction



- Why did neural networks make such predictions?
- Where does the model come from?
- Can we explain predictions in terms of training points?

Motivation

What's the influence of a training point to the prediction of test sample?

ML Background

ML Background

Empirical Risk Loss

Given a prediction problem: $\mathcal{X}$ (images) $\rightarrow$ $\mathcal{Y}$ (labels)

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

ML Background

Empirical Risk Loss

Given a prediction problem: $\mathcal{X}$ (images) $\rightarrow$ $\mathcal{Y}$ (labels)

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

- $z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$;
- $\theta \in \Theta$: parameter space;
- $L(z, \theta)$: loss function, *e.g.*, cross entropy.

ML Background

Empirical Risk Loss

Given a prediction problem: $\mathcal{X}$ (images) $\rightarrow \mathcal{Y}$ (labels)

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

- $z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$;
- $\theta \in \Theta$: parameter space;
- $L(z, \theta)$: loss function, *e.g.*, cross entropy.

$$\textbf{Goal:} \quad \hat{\theta} \stackrel{\text{def}}{=} \arg\min_{\theta \in \Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$$

ML Background

Empirical Risk Loss

Given a prediction problem: $\mathcal{X}$ (images) $\to$ $\mathcal{Y}$ (labels)

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

- $z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$;
- $\theta \in \Theta$: parameter space;
- $L(z, \theta)$: loss function, *e.g.*, cross entropy.

$$\textbf{Goal:} \quad \hat{\theta} \stackrel{\text{def}}{=} \arg\min_{\theta \in \Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$$

- Assumption: $R(\theta)$ is *twice-differentiable* and *strictly convex*.

Overview

Upweighting a training point

Upweighting a training point

Question 1:

How would predictions change if we did not have a specific training point?

Upweighting a training point

Question 1:

How would predictions change if we did not have a specific training point?

Influence Function Approximation

$$\hat{\theta}_{\epsilon,z} \stackrel{\text{def}}{=} \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z, \theta)$$

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

Upweighting a training point

Question 1:

How would predictions change if we did not have a specific training point?

Influence Function Approximation

$$\hat{\theta}_{\epsilon,z} \stackrel{\text{def}}{=} \arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i,\theta) + \epsilon L(z,\theta)$$

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

Influence of training point z to z_{test}

$$\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) \stackrel{\text{def}}{=} \left.\frac{dL(z_{\text{test}},\hat{\theta}_{\epsilon,z})}{d\epsilon}\right|_{\epsilon=0} = \nabla_\theta L(z_{\text{test}},\hat{\theta})^\top \left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0}$$

Perturbing a training point

Perturbing a training point

Question 2:

How would the predictions change if a specific training point was modified?

Perturbing a training point

Question 2:

How would the predictions change if a specific training point was modified?

Influence to θ with $z = (x, y)$ replaced by $z_\delta = (x + \delta, y)$

Perturbing a training point

Question 2:

How would the predictions change if a specific training point was modified?

Influence to θ with $z = (x, y)$ replaced by $z_\delta = (x + \delta, y)$

$$\hat{\theta}_{\epsilon, z_\delta, -z} \stackrel{\text{def}}{=} \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z_\delta, \theta) - \epsilon L(z, \theta)$$

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

$$\approx -H_{\hat{\theta}}^{-1} [\nabla_x \nabla_\theta L(z, \hat{\theta})] \delta$$

Perturbing a training point

Question 2:

How would the predictions change if a specific training point was modified?

Influence to θ with $z = (x, y)$ replaced by $z_\delta = (x + \delta, y)$

$$\hat{\theta}_{\epsilon, z_\delta, -z} \stackrel{\text{def}}{=} \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z_\delta, \theta) - \epsilon L(z, \theta)$$

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

Influence to z_{test} with z replaced by z_δ

$$\mathcal{I}_{\text{pert,loss}}(z, z_{\text{test}}) \stackrel{\text{def}}{=} -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_x \nabla_\theta L(z, \hat{\theta})$$

Overview

Efficiently Calculating Influence

Efficiently Calculating Influence

Challenges

$$\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$$

Efficiently Calculating Influence

Challenges

$$\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$$

- $H_{\hat{\theta}} = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta^2 L(z_i, \hat{\theta}), \theta \in R^p$: inversion takes $O(np^2 + p^3)$;
- $\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$ for all training points z_i;

Efficiently Calculating Influence

Challenges

$$\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$$

- $H_{\hat{\theta}} = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta^2 L(z_i, \hat{\theta}), \theta \in R^p$: inversion takes $O(np^2 + p^3)$;
- $\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$ for all training points z_i;

Hessian-Vector Products (HVPs)

$$s_{\text{test}} \stackrel{\text{def}}{=} H_{\hat{\theta}}^{-1} \nabla_\theta L(z_{\text{test}}, \hat{\theta})$$

$$\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = -s_{\text{test}} \cdot \nabla_\theta L(z_i, \hat{\theta})$$

Efficiently Calculating Influence

Challenges

$$\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$$

- $H_{\hat{\theta}} = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta^2 L(z_i, \hat{\theta}), \theta \in R^p$: inversion takes $O(np^2 + p^3)$;
- $\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$ for all training points z_i;

Hessian-Vector Products (HVPs)

$$s_{\text{test}} \overset{\text{def}}{=} H_{\hat{\theta}}^{-1} \nabla_\theta L(z_{\text{test}}, \hat{\theta})$$

$$\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = -s_{\text{test}} \cdot \nabla_\theta L(z_i, \hat{\theta})$$

Solving s_{test}

- Conjugate Gradients (CG): $\arg\min_t \{\frac{1}{2} t^\top H_{\hat{\theta}} t - s_{\text{test}}^\top t\}$;
- Stochastic estimation: speed up CG;

Overview

Validation and Extensions

Validation and Extensions

Validity of Influence Function

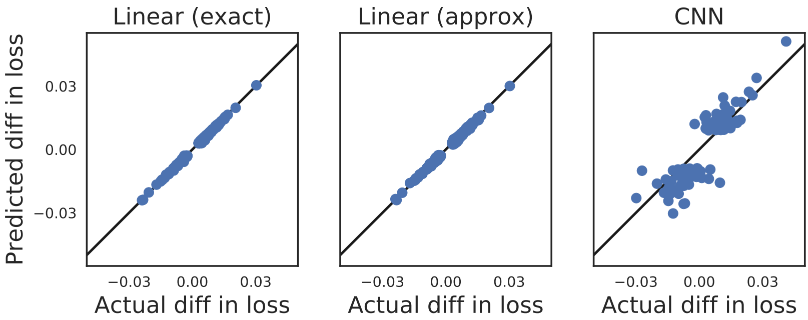
$$-\frac{1}{n}\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) \quad \text{vs.} \quad L(z_{\text{test}}, \hat{\theta}_{-z}) - L(z_{\text{test}}, \hat{\theta})$$

- Influence function vs. leave-one-out retraining

Validity of Influence Function

$$-\frac{1}{n}\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) \quad \text{vs.} \quad L(z_{\text{test}}, \hat{\theta}_{-z}) - L(z_{\text{test}}, \hat{\theta})$$

- Influence function vs. leave-one-out retraining



Validation and Extensions

Validation and Extensions

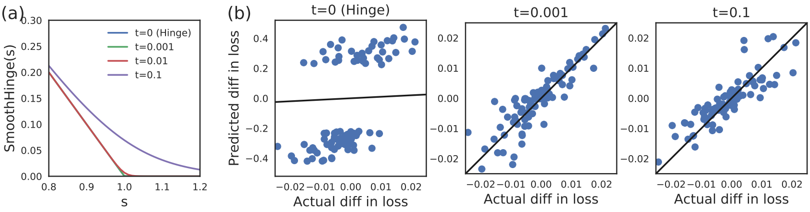
Extensions

- Non-convexity and non-convergence: quadratic approximation
  - $\widetilde{L}(z,\theta) = L(z,\widetilde{\theta}) + \nabla L(z,\widetilde{\theta})^\top(\theta - \widetilde{\theta}) + \frac{1}{2}(\theta - \widetilde{\theta})^\top(H_{\widetilde{\theta}} + \lambda I)(\theta - \widetilde{\theta})$
- Non-differentiable loss: replaced with smoothing version
  - $\mathrm{Hinge}(s) = \max(0, 1-s) \rightarrow \mathrm{SmoothHinge}(s,t) = t\log(1+\exp(\frac{1-s}{t}))$

Validation and Extensions

Extensions

- Non-convexity and non-convergence: quadratic approximation
  - $\widetilde{L}(z,\theta) = L(z,\widetilde{\theta}) + \nabla L(z,\widetilde{\theta})^\top(\theta-\widetilde{\theta}) + \frac{1}{2}(\theta-\widetilde{\theta})^\top(H_{\widetilde{\theta}}+\lambda I)(\theta-\widetilde{\theta})$
- Non-differentiable loss: replaced with smoothing version
  - $\mathrm{Hinge}(s) = \max(0, 1-s) \to \mathrm{SmoothHinge}(s,t) = t\log(1+\exp(\frac{1-s}{t}))$



Overview

Use Cases of Influence Functions

Use Cases of Influence Functions

- Model Behavior Understanding: $-\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$

Use Cases of Influence Functions

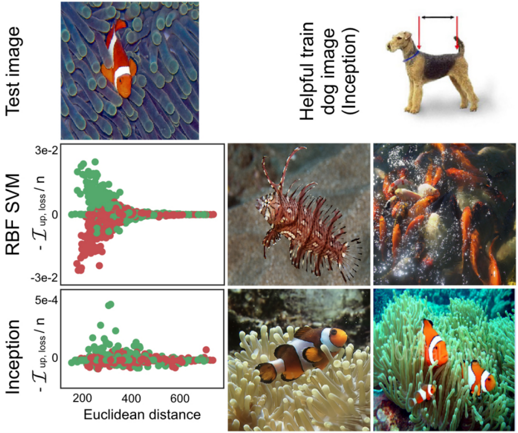
- Model Behavior Understanding: $-\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$

RBF SVM vs. *Inception* for Fish/Dog Classification

Use Cases of Influence Functions

- Model Behavior Understanding: $-\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$

RBF SVM vs. *Inception* for Fish/Dog Classification



Use Cases of Influence Functions

Use Cases of Influence Functions

- Adversarial training examples: $\mathcal{I}_{\text{pert,loss}}(z, z_{\text{test}})$

Use Cases of Influence Functions

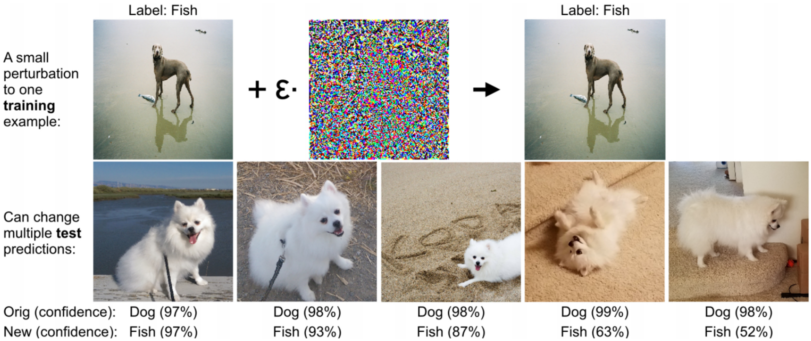
- Adversarial training examples: $\mathcal{I}_{\text{pert,loss}}(z, z_{\text{test}})$

$$\widetilde{z}_i := \prod(\widetilde{z}_i + \alpha \text{sign}(\mathcal{I}_{\text{pert,loss}}(\widetilde{z}_i, z_{\text{test}})))$$

Use Cases of Influence Functions

- Adversarial training examples: $\mathcal{I}_{\text{pert,loss}}(z, z_{\text{test}})$

$$\tilde{z}_i := \prod(\tilde{z}_i + \alpha \text{sign}(\mathcal{I}_{\text{pert,loss}}(\tilde{z}_i, z_{\text{test}})))$$



Use Cases of Influence Functions

Use Cases of Influence Functions

- Debugging domain mismatch: $-\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$

Use Cases of Influence Functions

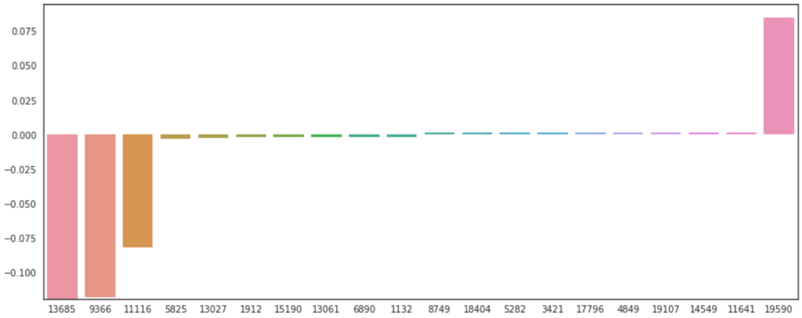
- Debugging domain mismatch: $-\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$

Patients re-admission classification
children re-admitted: $3/24$ changed to $3/4$

Use Cases of Influence Functions

- Debugging domain mismatch: $-\mathcal{I}_{\text{up,loss}}(z_i, z_{\text{test}})$

Patients re-admission classification
children re-admitted: $3/24$ changed to $3/4$



Use Cases of Influence Functions

Use Cases of Influence Functions

- Fixing mislabeled examples: $-\mathcal{I}_{\text{up,loss}}(z_i, z_i)$

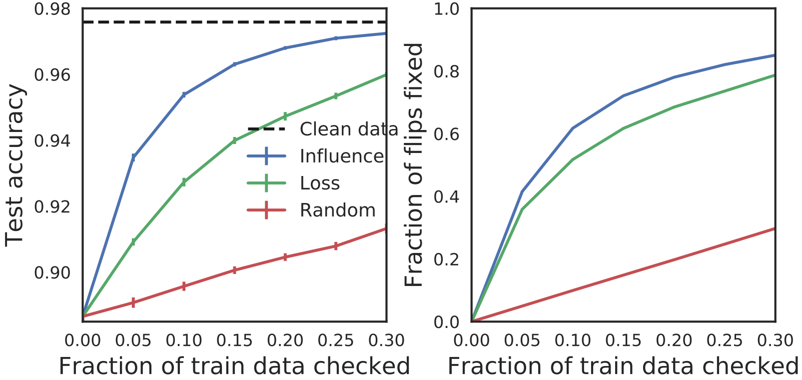
Use Cases of Influence Functions

- Fixing mislabeled examples: $-\mathcal{I}_{\text{up,loss}}(z_i, z_i)$

Email spam classification: 10% labels flipped

- Fixing mislabeled examples: $-\mathcal{I}_{\text{up,loss}}(z_i, z_i)$

Email spam classification: 10% labels flipped



Overview

1. Introduction
2. Approach
   - Upweighting a training point
   - Perturbing a training point
3. Influence Calculation
4. Validation and Extensions
5. Applications

Questions?