Paper Reading:Understanding Black-box Predictions via Influence Functions

Songyang Zhang

ShanghaiTech University

August 10, 2017

Overview

Overview

Learning Problem at a Glance

Statistical Learning Theory

Learning is viewed as a generalization/inference problem from usually **small** sets of **high dimensional**, noisy data.

Given a training set of *input-output* pairs

$$S_n = (x_1, y_1), ...(x_n, y_n)$$

find f_s such that

$$f_s(x) \sim y.$$

e.g. the x's are vectors, and the y's discrete labels in classification and real values in regression.

Learning is Inference

For the above problem to make sense we need to assume input and output are related!

Statistical and Supervised Learning

- Each input-output pairs is a sample from a fixed but unknown distribution $\mu(x, y)$
- Under some condition we can associate to $\mu(z)$ the probability

$$p(x, y) = p(y|x)p(x).$$

- the training set S_n is a set of **independent identically distributed(i.i.d)** samples drawn from $\mu(z)$
- It's crucial to note that we view $p(x, y)$ as fixed but unknown.

Generalization/Prediction

Predictivity or Generalization

- Given the data, the goal is to learn how to make decisions/predictions about future data/data belonging to the training set.
- **Generalization** is the key requirement emphasized in Learning Theory: it it the measure of predictiviy.
- This emphasis make it different from Bayesian or traditional statistics(especially explanatory statistics).

The problem is often:**Avoid overfitting!**

Loss functions

In order to define generalization we need to define and measure errors.

Loss functions

A loss function $V : \mathbf{R} \times Y$ determines the price $V(f(x), y)$ we pay, predicting $f(x)$ when in fact the true output is y.

Loss functions for regression

- The most common is the **square loss** or L_2 **loss**.

$$V(f(x), y) = (f(x) - y)^2$$

- **Absolute value** or L_1 **loss**

$$V(f(x), y) = |f(x) - y|$$

- Vapnik's ϵ-**insensitive loss**:

$$V(f(x), y) = (|f(x) - y| - \epsilon)_+$$

Loss functions for (binary) classification

- The most intuitive one:0-1 **loss**.

$$V(f(x), y) = \theta(-yf(x))$$

(θ is the step function)

- The more tractable **hinge loss**

$$V(f(x), y) = (1 - yf(x))_+$$

- And again the **square loss** or L_2 **loss**:

$$V(f(x), y) = (1 - yf(x))^2$$

Expected Risk

A good function — we will speak of function or *hypothesis*–should incur in only a few errors. We need a way to quantify this idea.

Expected Risk

The quantity

$$I[f] = \int_{X \times Y} v(f(x), y) p(x, y) dx dy.$$

is called the expected error and measures the loss averaged over the unknown distribution.

A good function should have small expected risk.

Empirical Risk

We would like to make I[f] small, but in general we do not know $p(x, y)$.

Empirical Risk

Given a function f, a loss function V, and a training set S consisting of n data points, the empirical risk(empirical error) of f is:

$$I_n[f] = \frac{1}{n} \sum_{i=1}^{n} V(f(x_i), y_i)$$

Basic definitions summary

- $p(x, y)$ probability distribution,
- S_n training set,
- $V(f(x), y)$ loss function,
- $I_n[f] = \frac{1}{n} \sum_{i=1}^{n} V(f(x_i), y_i)$, empirical risk,
- $I[f] = \int_{X \times Y} v(f(x), y) p(x, y) dx dy$, expected risk.

Expected risk and empirical risk(1)

X Convergence in probability

Let X_n be a sequence of bounded random variables. Then

$$\lim_{n\to\infty} X_n = X \qquad \text{in probability}$$

if

$$\forall \epsilon > 0 \qquad \lim_{n\to\infty} \mathbb{P}\{|X_n - X| \geq \epsilon\} = 0$$

Expected risk and empirical risk(2)

$I_n[f]$ Convergence in probability

Let X_n be a sequence of bounded random variables. Then

$$\lim_{n\to\infty} I_n[f] = I[f] \qquad \text{in probability}$$

if

$$\forall \epsilon > 0 \qquad \lim_{n\to\infty} \mathbb{P}\{|I_n[f] - I[f]| \geq \epsilon\} = 0$$

Hypotheses Space $\mathcal{H}$

Learning algorithm

A learning algorithm A is a map from the data space to $\mathcal{H}$.

$$A(S_n) = f_n \in \mathcal{H}$$

ERM and SRM(1)

Empirical Risk Minimization

A prototype algorithm in statistical learning theory is Empirical Risk Minimization.

$$f_* = argmin_{f \in \mathcal{H}} I_n[f]$$

if the dataset is large enough.

Easy to Overfitting.

ERM and SRM(2)

Structural Risk Minimization(Regularization)

- SRM in statistical learning theory is to avoid overfitting. Add regularizer or penalty term on empirical risk.
- Structural Risk

$$I_{n,srm}[f] = \frac{1}{n}\sum_{i=1}^{n} V(f(x_i), y_i) + \lambda J(f)$$

- Structural Risk Minimization

$$f_* = argmin_{f \in \mathcal{H}} I_{n,srm}[f]$$

Generative and Discriminative Model(1)

Task of supervised learning is to learn a model, which could give an output corresponding to the input.

$$Y = f(X)$$

or

$$P(Y|X)$$

Generative and Discriminative Model(2)

Generative approach

- Learn a joint probability distribution $P(X, Y)$ from data.
- Calculate conditional probability distribution $P(Y|X)$ from $P(X, Y)$

$$P(Y|X) = \frac{P(X, Y)}{P(X)}$$

Generative and Discriminative Model(3)

Discriminative approach

- Learn a conditional probability distribution $P(Y|X)$ from data.
- $P(X, Y)$ is unknown and could not be get.
- This approach is for prediction directly.

Overview

Newton method

Netwon methods is an application of Talyor Polynomials for finding roots of functions.

Newton method

Suppose that $x = c$ is an (unknown) zero of f and that f is differentiable in open interval that contains c. To approximate c

1. Make an initial approximation x_1 close to c.
2. Determine a new approximation using the formula

$$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)}$$

3. If $|x_n - x_{n+1}|$ is less than the desired accuracy, let x_{n+1} serve as the final approximation. Otherwise return to step.2.

Convergence of Newton method

Newton method will not converge if

1. If $f'(x_n) = 0$ for some n.
2. If $\lim_{n\to\infty} x_n$ does not exist.

Proof on whiteboard.

Newton method for non-linear optimization

Newton method will not converge if

1. For $f(x) = 0$, use first-order Taylor expansion.
2. For $f'(x) = 0$, use second-order Taylor expansion.(second-order differentiable)

$$x_{n+1} = x_n - \frac{f' x_n}{f''(x_n)}$$

$$x_{n+1} = x_n - H^{-1}(x_n) \cdot J_f(x_n)$$

$H(x_n)$ is Hessian Matrix, $J_f(x_n)$ is Jacobian Matrix

Proof on white-board.

Jacobian Matrix and Hessian Matrix

Jacobian Matrix

$$J_f(X_n) = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_n} \end{bmatrix}$$

Hessian Matrix

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Overview

Why did it make this prediction?

1. DNN are black-boxes whose predictions are hard to explain.
2. Tracing a model's predictions through its learning algorithm and its training data, where the model parameters ultimately come from.
3. What would happen on a prediction if we removed this training point, or if its values were changed slightly?
4. With influence functions, a technique from robust statistics.Explain how the model parameters change as we upweight a training point by an infinitesimal amount.
5. Using second-order optimization techniques,can efficiently approximate influence in any differentiable black-box model, only requiring oracle access to gradients and Hessian-vector products.

Overview

Definitions and notations

- A prediction problem:from some input space $\mathcal{X}$(e.g.,images) to an output space $\mathcal{Y}$(e.g.,labels)
- Training points:$z_1, ..., z_n$, where $z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$
- Loss: $L(z, \theta)$,for a pint z and parameter $\theta \in \Theta$
- Empirical risk:$\frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta)$
- Empirical risk minimizer: $\hat{\theta} \stackrel{\text{def}}{=} argmin_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta)$

Upweighting a training point(1)

Let us begin by studying the change in model parameters due to removing a point z from the training set.

- This change is $\Delta = \hat{\theta_{-z}} - \hat{\theta}$

$$\hat{\theta}_{-z} \stackrel{\text{def}}{=} argmin_{\theta \in \Theta} \sum_{z_i \neq z} L(z_i, \theta)$$

- z are upweighted by some small ϵ

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

- The influence of upweighting z on the parameter $\hat{\theta}$ is given by:

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

where $H_\theta \stackrel{def}{=} \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta^2 L(z_i, \hat{\theta})$ is Hessian and is PD

Upweighting a training point(2)

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:

$$\hat{\theta_{-z}} - \hat{\theta} = -\frac{1}{n}\mathcal{I}_{up,params}(z)$$

without retraining the model.

Upweighting a training point(3)

The influence of upweighting z on the loss at a test point z_{test} again has a closed-form expression:

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

Chain rule

$$[f(g(x))]' = f'(g(x)) \cdot g'(x)$$

Perturbing a training input(1)

Finer-grained:how would the model's predictions change if a training input were modified?

- For a training point $z = (x, y)$, define $z_\delta \stackrel{\text{def}}{=} (x + \delta, y)$.
- Let $\hat{\theta}_{z\delta,-z}$ be the ERM on training points with z_δ instead of z.
- $\hat{\theta}_{\epsilon,z_\delta,-z} \stackrel{\text{def}}{=} argmin_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i,\theta) + \epsilon L(z_\delta,\theta) - \epsilon L(z,\theta)$
- An analogous calculation:

$$\begin{aligned}\frac{\mathrm{d}\hat{\theta}_{\epsilon,z_\delta,-z}}{\mathrm{d}\epsilon}\Big|_{\epsilon=0} &= \mathcal{I}_{up,params}(z_\delta) - \mathcal{I}_{up,params}(z) \\ &= -H_{\hat{\theta}}^{-1}(\nabla_\theta L(z_\delta,\hat{\theta}) - \nabla_\theta L(z,\hat{\theta}))\end{aligned}$$

- We can make linear approximation()

$$\hat{\theta}_{z_\delta,-z} - \hat{\theta} \approx -\frac{1}{n}(\mathcal{I}_{up,params}(z_\delta) - \mathcal{I}_{up,params}(z))$$

Perturbing a training input(2)

If x is continuous and δ is small.

Further approximate:

- Assume the input domain $\mathcal{X} \subseteq \mathbb{R}^d$, the parameter space $\Theta \subseteq \mathbb{R}^p$, L is differentiate in θ and x.
- As $||\delta|| \to 0$, we have: (Lagrange Mean Value Theorem)

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

- Substituting into last slide:

$$\frac{\mathrm{d}\hat{\theta}_{\epsilon, z_\delta, -z}}{\mathrm{d}\epsilon}\bigg|_{\epsilon=0} = -H_{\hat{\theta}}^{-1}[\nabla_x \nabla_\theta L(z, \hat{\theta})]\delta$$

- Thus have:

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

Perturbing a training input(3)

Differentiating w.r.t. δ and applying the chain rule:

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

where $\mathcal{I}_{pert,loss}(z, z_{test})^\top \delta$ tells us the approximate effect that $z \longmapsto z + \delta$ has on the loss at z_{test}

- Setting δ in the directon of $\mathcal{I}_{pert,loss}(z, z_{test})$, we can construct local perturbations of z that maximally increase the loss at z_{test}
- $\mathcal{I}_{pert,loss}(z, z_{test})$ can help us identify the features of z that are most responsible for the prediction on z_{test}

Relation to Euclidean distance(1)

- Find training points which are the most relevant to a test points by finding its nearest neighbor in Euclidean space, choosing x with the largest $x \cdot x_{test}$.
- Compare Euclidean distance with $\mathcal{I}_{up,loss}(z, z_{test})$ on a logistic regression model.
- $p(y|x) = \sigma(y\theta^\top x)$, with $y \in -1, 1$ and $\sigma(t) = \frac{1}{1+\exp(-t)}$.
- For a training point $z = (x, y)$,

$$L(z, \theta) = log(1 + \exp(-y\theta^\top x))$$
$$\nabla_\theta L(z, \theta) = -\sigma(-y\theta^\top x)yx$$
$$H_\theta = \frac{1}{n}\sum_{i=1}^{n} \sigma(\theta^\top x_i)\sigma(-\theta^\top x_i)x_i x_i^\top$$

- From last section:$\mathcal{I}_{up,loss}(z, z_{test})$:
$-y_{test}y \cdot \sigma(-y_{test}\theta^\top x_{test}) \cdot \sigma(-y\theta^\top x) \cdot x_{test}^\top H_{\hat{\theta}}^{-1} x$

Relation to Euclidean distance(2)

For $\mathcal{I}_{up,loss}(z, z_{test})$:

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

Key properties of Influence function

Two key difference from $x \cdot x_{text}$

1. $\sigma(-y\theta^\top x)$ gives points with high training loss more influence, revealing that outliers can dominate the model parameters.
2. The weighted covariance matrix $H_{\hat{\theta}}^{-1}$ measures the "resistance" of the other training points to the removal of z

The influence functions capture the effect of model training much more accurately than nearest neighbor.

Use HVP to calculate Hessian Matrix(1)

Two challenges to using

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

- The Hessian of the empirical risk, $H_\theta \overset{def}{=} \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta^2 L(z_i, \hat{\theta})$ With n training points and $\theta \in \mathbb{R}^p$, need $O(np^2 + p^3)$ operations.
- We often want calculate $\mathcal{I}_{up,loss}(z, z_{test})$ across all training points z_i

Use HVPs to calculate Hessian Matrix(2)

Hessian-vector products

- For function $f(\mathbf{x})$, its gradient $g(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}} f(\mathbf{x})$, Hessian $H(\mathbf{x}) = \frac{\partial^2}{\partial \mathbf{x} \partial \mathbf{x}^n} f(\mathbf{x})$
- $g(\mathbf{x} + \Delta\mathbf{x}) \sim g(\mathbf{x}) + H(\mathbf{x})\Delta\mathbf{x}$
- Consider $H(\mathbf{x})\mathbf{v}$, for a small r:$g(\mathbf{x} + r\mathbf{v}) \approx g(\mathbf{x}) + rH(\mathbf{x})\mathbf{v}$ So,

$$H(\mathbf{x})\mathbf{v} \approx \frac{g(\mathbf{x} + r\mathbf{v}) - g(\mathbf{x})}{r}$$

Only for $r \longmapsto 0$, Or:

$$H(\mathbf{x})\mathbf{v} \approx \frac{g(\mathbf{x} + r\mathbf{v}) - g(\mathbf{x} - r\mathbf{v})}{2r}$$

This expression will be closer to the true value for larger r

CGM for efficiently computing(1)

- For solving $Ax = B$, equivalent to $x = A^\top B$,
- So, we design a problem, $\min_x(\frac{1}{2}x^\top Ax + B^\top x)$:
- Equivalent to solve: $\frac{\partial}{\partial x}(\frac{1}{2}x^\top Ax + B^\top x) = 0$

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

Use Conjecture Gradient Method to sole the partial function.

CGM for efficiently computing(2)

Use HVPs to efficiently approximate $s_{test} \overset{def}{=} H_{\hat{\theta}}^{-1} \nabla_\theta L(z_{test}, \hat{\theta})$
And get:

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

For second problem in computing, we can precompute s_{test} and then efficiently compute$-s_{test} \nabla_\theta L(z_i, \hat{\theta})$ for each training point z_i

Stochastic estimation

Details in paper.

Relax the assumptions

The influence function are asymptotic approximations of leave-one-out retraining under the assumptions:

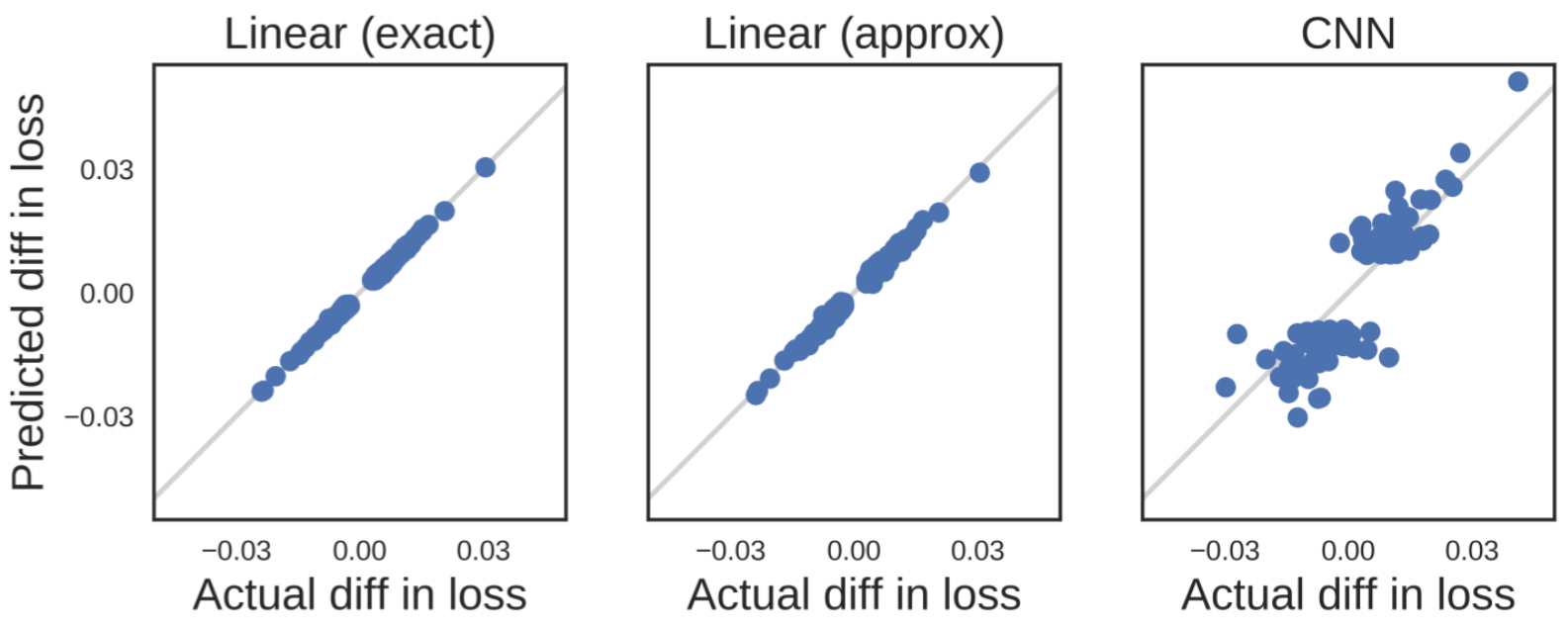
- The model parameter θ minimize the empirical risk
- The empirical risk is twice-differentiable and strictly convex.

Influence functions v.s. leave-one-out retraining

Evaluate the accuracy of using influence functions to approximate the effect of removing a training point and retraining

Compare:

$$-\frac{1}{n}\mathcal{I}_{up,loss}(z, z_{test}) \qquad v.s. \qquad L(z_{test}, \hat{\theta}_{-z}) - L(z_{test}, \hat{\theta})$$



Non-convexity and non-convergence

In section 2. $\hat{\theta}$ is the global minimum. We will get parameters $\tilde{\theta}$ use SGD with early stopping or non-convex objectives,Then

$$\tilde{\theta} \neq \hat{\theta}$$

And $H_{\tilde{\theta}}$ have negative eigenvalues.But still useful in practice.

Approach

A convex quadratic approximation of the loss around $tilde\theta$:

$$\tilde{L}(z,\theta) = L(z,\tilde{\theta}) + \nabla L(z,\tilde{\theta})^{\top}(\theta-\tilde{\theta}) + \frac{1}{2}(\theta-\tilde{\theta})^{\top}(H_{\tilde{\theta}}+\lambda I)(\theta-\tilde{\theta})$$

where λ is a damping term we add if $H_{\tilde{\theta}}$ has negative eigenvalues, corresponding to adding L_2 regularization on θ

Non-differentiable losses

- What happens when the derivatives of the loss, $\nabla_\theta L$ and $\nabla^2_\theta L$, does not exist?
- For a linear SVM use Hinge loss: Hinge$(s) = \max(0, 1 - s)$, (similar to ReLUs, which cause non-differentiability in NN.)
- Approximated Hinge(s) with SmoothHinge(s, t):

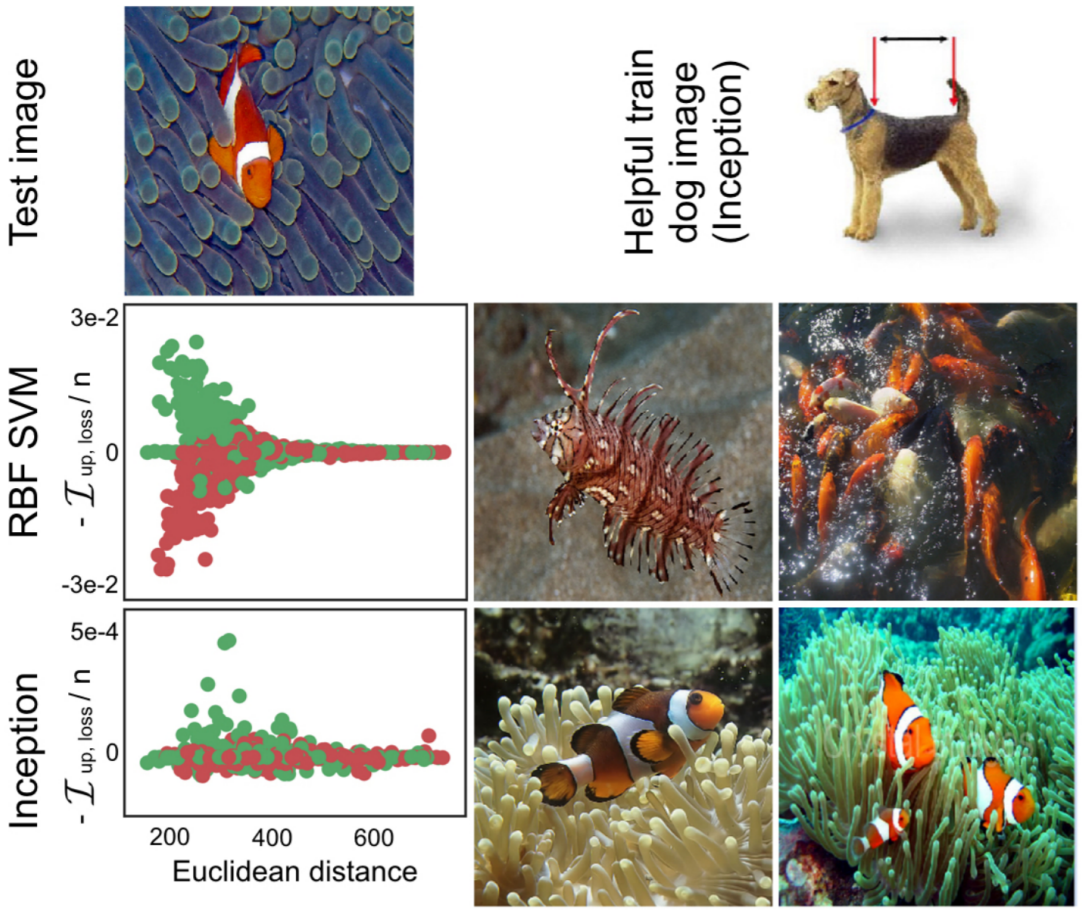
$$t \log(1 + \exp(\tfrac{1-s}{t}))$$

It will approaches Hinge(s) as $t \longmapsto 0$

Overview

Application I:Understanding model behavior(1)

For Inception-V3 v.s. SVM with RBF(use SmoothHinge)



Application I:Understanding model behavior(2)

For Inception-V3 v.s. SVM with RBF(use SmoothHinge)

What the model have learned?

- The inception networks(DNN) picked up on the distinctive characteristics of the fish.
- RBF SVM pattern-matched training images superficially

Application II:Adversarial training examples(1)

First proof-of-concept that visually-indistinguishable training attacks can be executed on NN.

Basic idea(For each test image separately)

$\mathcal{I}_{pert,loss}(z, z_{test})$ tells us how to modify training point z to most increase that loss on z_{test}

1. Construct $\tilde{z}_i$ as an adversarial version of z_i
2. Initializing: $\tilde{z}_i := z_i$
3. Iterating: $\tilde{z}_i := \prod(\tilde{z}_i + \alpha \text{sign}(\mathcal{I}_{pert,loss}(z, z_{test})))$ where α is the step size
4. Retrain the model after each iteration.

Results: Original:591/600,Perturbing(on 335 images) 2 training images:77% of 591 flipped;10 training images: 590/591 flipped

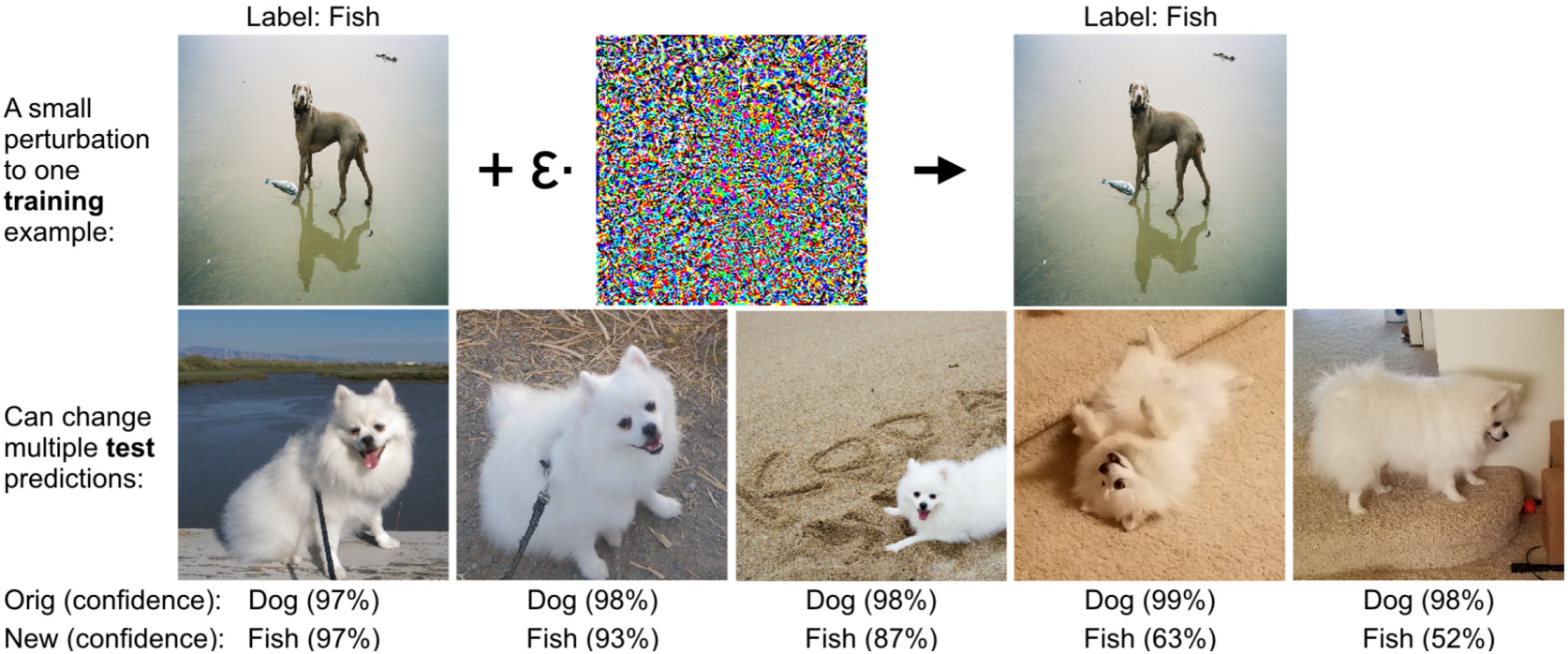
Application II:Adversarial training examples(2)

For attacking multiple test images simultaneously

Observation

1. Changing training values less than 1%(L_2 pixel space) of the mean distance of a training point to its class centroid.
   In Inception feature space, the change is on the same order as the mean distance.
2. Perturbing the training example in a direction of low variance, causing overfitting.
3. Ambiguous or mislabeled training images are effective to attack.

Application II:Adversarial training examples(3)



Measuring the magnitude of $\mathcal{I}_{pert,loss}$ gives model developers a way of quantifying how vulnerable their models are to training-set attacks.

Application III:Debugging domain mismatch

The training distribution does not match the test distribution, cause model with high training acc, while do poorly on test data.

An example

For original training set: 3/24 is $+1$, 21/24 is -1; after mismatching: 3/4 is $+1$, 1/4 is -1(drop 20 -1)

- Picking a random test point z_{test} that model got wrong,calculating $-\mathcal{I}_{up,loss}(z_i, z_{test})$ for each training point z_i
- The one training point(label is -1) have very negative influences.
- The three training points(label is $+1$) have very positive influences.

Application IV:Fixing mislabeled examples(1)

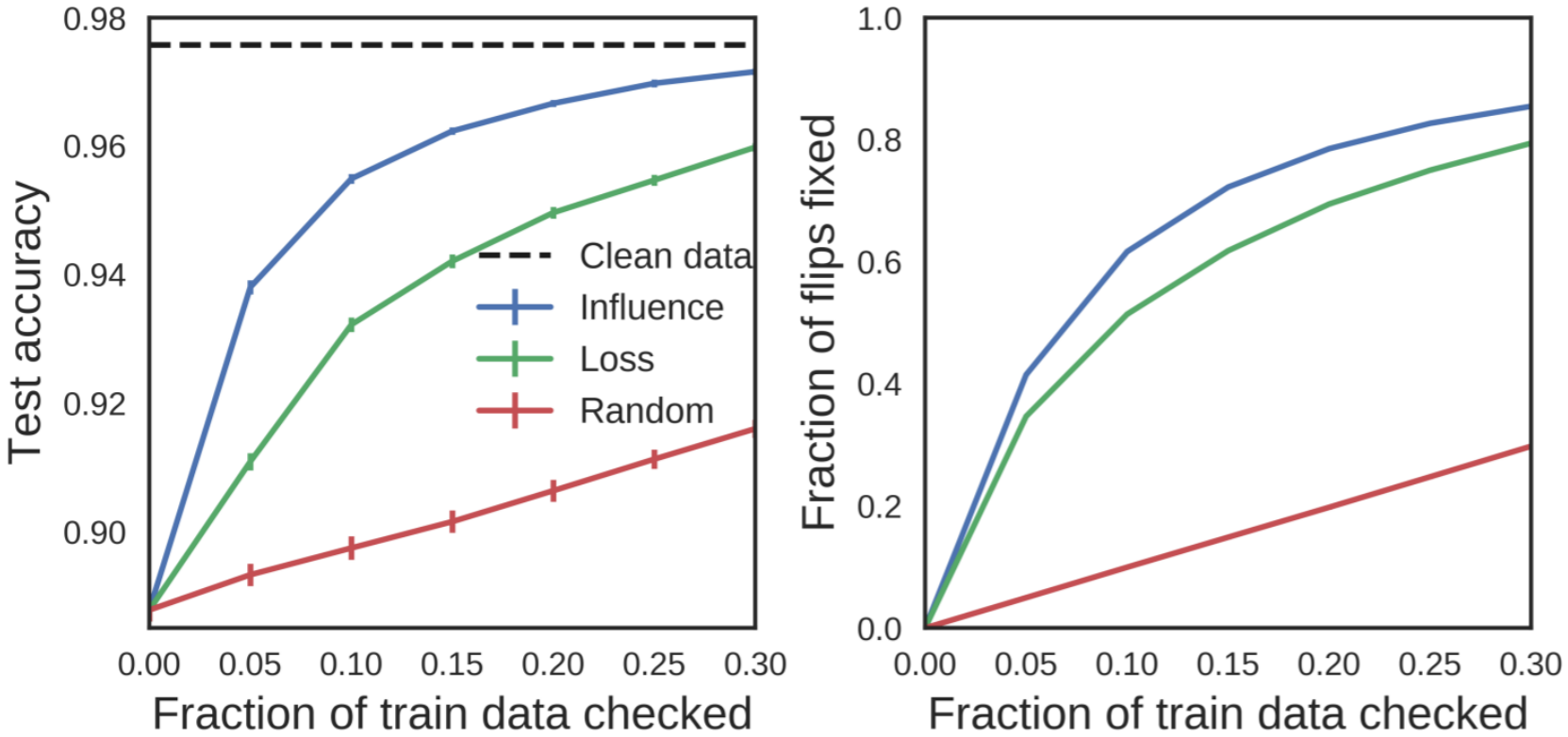
Labels in the real world are often noisy, especially if crowd-sourced

Key idea

We measure the influence of z_i with $\mathcal{I}_{up,loss}(z_i, z_i)$, which approximates the error incurred on z_i if we remove z_i from the training set.

Application IV:Fixing mislabeled examples(2)

Result curve(simulated manually inspecting)



Review(1)

- 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, **identifying the points most responsible for a given prediction**.
- On linear models and ConvNets,the influence functions can be used:
  - To understand model behavior,
  - Debug models and detect dataset errors,
  - Identify and exploit vulnerabilities to adversarial training-set attacks.

Review(2)

Mathematical foundations && Optimization methods

- Taylor Expansion, Lagrange Mean Value Theorem, Positive Definite Matrix, Jacobian Matrix, Hessian Matrix
- Newton Method, HVPs, Conjugate Gradient Method, Stochastic Estimation