Review of "Understanding Black-box Predictions via Influence Functions"

Eole Cervenka [1] **Geovani Rizk** [1]

Abstract

Pang Wei Koh and Percy Liang introduce a new way to analyze a model predictions via its training data. (Koh & Liang, 2017). Based on their paper, we explain how influence functions can be used to efficiently track the impact of each training points on any test point prediction, even when using a non-convex and/or non-differentiable loss function. We discuss about the author's use cases dealing with tracking influence in understanding model behavior, engineering adversarial training points, debugging a model and checking labels efficiently. We attempt to understand how test prediction accuracy evolves depending in the choice of training set by implementing our own use case.

1. Introduction

State-of-the-art learning models such as deep neural networks are often black boxes (Krizhevsky et al., 2012) and understanding model predictions pose a challenge that limits our ability to interpret, debug existing models or create new models. Prior efforts in understanding model predictions have always started with the assumption that model where fixed functions; taking an input and producing an output.This paper takes a new approach by measuring the influence of training data in the learning process and using this information to measure the influence of training data on the prediction process. Understanding model predictions can therefore be done through the lens of the data points that were helpful in making the prediction.

The brute-force method to measure how changing the training data can affect the model predictions is prohibitively expensive as it requires retraining for each modified training set. The authors show that influence functions offer a closed-form solution for approximating how perturbing the training set affects the model parameters, without going through retraining the model from scratch. Using influence function requires an expensive second derivative calculations (*i.e.* forming the Hessian matrix). Well see how Hessian Vector Product can be used to make influence functions tractable even in models with millions of parameters. Influence functions require differentiability and convexity constraints that are not satisfied in state of the art models (*e.g.* deep learning settings). The authors present how influence functions can be accurately approximated using second order optimization techniques. Finally, the direct applications of this idea will be experimented through use cases of debugging a model, detecting dataset errors and engineering adversarial training points.

[1]University of Paris Dauphine, Paris, France. Correspondence to: Eole Cervenka <eole.cervenka@dauphine.eu>, Geovani Rizk <geovani.rizk@dauphine.eu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

2. Influence Function approach

Suppose that we want to build a model for a prediction problem from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. We use the same notation that the original authors and give the training points $z_1, ..., z_n$ where $z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. Let $L(z, \theta)$ be the loss value for a certain training point z and the parameters θ of the model, and let $\frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$ be the empirical risk. By definition, the empirical risk minimizer is given by $\hat{\theta} \overset{\text{def}}{=} \arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(z_i, \theta)$. In the first parts of this section we suppose that the empirical risk is twice differentiable and strictly convex. In 2.4, we explain how influence function can be approximated accurately and provide useful informations even in non-convex and/or non-differentiable empirical risk context.

2.1. Training set modification

In this section, we will demonstrate how influence functions can measure the impact of a modification in the training set on the parameters and the model prediction. Two kinds of dataset modification can be tracked this way:

- removing a training point which can be used to debug the model and to detect dataset errors.
- perturbing a training point which can be used for instance to create adversarial training examples.

In both types of dataset modification, measuring the impact with the brute-force method requires retraining the model which is typically not tractable.

2.1.1. REMOVING A TRAINING POINT

Suppose we want to see the effect of removing a point from the training set on the model parameters; it is equivalent to up-weighting said point by $\epsilon = -\frac{1}{n}$ in the calculation of the empirical risk minimizer :

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

Influence functions allows us to express as a closed form expression the variation in θ with respect to a variation in ϵ.

Influence of up-weighting a point z on parameters $\hat{\theta}$ is given by :

$$\begin{aligned}\mathcal{I}_{\text{up,params}}(z) &\overset{\text{def}}{=} \frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}|_{\epsilon=0} \\ &= -\mathcal{H}_{\hat{\theta}}^{-1}\nabla_\theta\mathcal{L}(z,\hat{\theta}) \end{aligned} \tag{1}$$

Thus, it can be used to linearly approximate the effect of up-weighting by $-\frac{1}{n}$ a point in the training set on parameters $\hat{\theta}$ given by $\hat{\theta}_{-z} - \hat{\theta} \approx -\frac{1}{n}\mathcal{I}_{\text{up,params}}(z)$.

The variation of the loss on z_{test} with respect to the parameters is given by $\nabla_\theta\mathcal{L}(z_{\text{test}},\hat{\theta})$. By chaining, we can express the variation of the loss on z_{test} with respect to ϵ :

$$\begin{aligned}\mathcal{I}_{\text{up,loss}}(z,z_{\text{test}}) &= \nabla_\theta\mathcal{L}(z_{\text{test}},\hat{\theta})\ \hat{\frac{d\theta_{\epsilon,z}}{d\epsilon}}|_{\epsilon=0} \\ &= -\nabla_\theta\mathcal{L}(z_{\text{test}},\hat{\theta})\mathcal{H}_{\hat{\theta}}^{-1}\nabla_\theta\mathcal{L}(z,\hat{\theta}) \end{aligned} \tag{2}$$

2.1.2. PERTURBING A TRAINING POINT

Perturbing a training point z is equivalent to remove it and add its perturbed version $z_\delta = (x+\delta, y)$. Using what we have previously introduce about adding weight to the point, the empirical risk minimizer can be expressed by :

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

Analogically, influence of perturbing a point z on parameters $\hat{\theta}$ can be expressed as the influence of transferring the weight from point z to point z_δ :

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

Suppose that x is continuous and δ small enough, we can linearly approximate

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

As for up-weighting a training point by $\epsilon = -\frac{1}{n}$, the effect $\hat{\theta}_{z_\delta,-z} - \hat{\theta}$ of perturbing a training point can be approximated by :

$$\begin{aligned}\hat{\theta}_{z_\delta,-z} - \hat{\theta} &\approx -\frac{1}{n}\mathcal{I}_{\text{pert,params}}(z) \\ &\approx -\frac{1}{n}\mathcal{H}_{\hat{\theta}}^{-1}\left(\nabla_\theta\mathcal{L}(z_\delta,\hat{\theta}) - \nabla_\theta\mathcal{L}(z,\hat{\theta})\right) \\ &\approx -\frac{1}{n}\mathcal{H}_{\hat{\theta}}^{-1}\nabla_x\nabla_\theta\mathcal{L}(z,\hat{\theta})\delta \end{aligned} \tag{4}$$

By derivation of $\hat{\theta}_{z_\delta,-z} - \hat{\theta}$ and by analogy with the calculation of $\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}})$, we can estimate the influence of a perturbation on a training point :

$$\begin{aligned}\mathcal{I}_{\text{pert,loss}}(z,z_{\text{test}}) &\overset{\text{def}}{=} \nabla_\delta\mathcal{L}(z_{\text{test}},\hat{\theta}_{z_\delta,-z})|_{\delta=0} \\ &= -\nabla_\theta\mathcal{L}(z_{\text{test}},\hat{\theta})\mathcal{H}^{-1}\nabla_x\nabla_\theta\mathcal{L}(z,\hat{\theta}) \end{aligned} \tag{5}$$

Upon demonstrating this result, we couldn't find the same right-hand side as the original authors equation. Specifically, we are getting a factor of $\frac{1}{n}$ that is not in the original paper. This factor however doesn't change how we use the equation. In fact, the chaining rule gives us the following statement :

$$\begin{aligned}\mathcal{I}_{\text{pert,loss}}(z,z_{\text{test}})^\top &\overset{\text{def}}{=} \nabla_\delta\mathcal{L}(z_{\text{test}},\hat{\theta}_{z_\delta,-z})|_{\delta=0} \\ &= \nabla_\theta\mathcal{L}(z_{\text{test}},\hat{\theta})\frac{d(\hat{\theta}_{z_\delta,-z})}{d\delta}|_{\delta=0} \end{aligned} \tag{6}$$

We use the derivative with respect to δ of $\hat{\theta}_{z_\delta,-z} - \hat{\theta}$ which gives us :

$$\begin{aligned}\frac{d(\hat{\theta}_{z_\delta,-z} - \hat{\theta})}{d\delta} &\approx \frac{d(-\frac{1}{n}\mathcal{H}_{\hat{\theta}}^{-1}\nabla_x\nabla_\theta\mathcal{L}(z,\hat{\theta})\delta)}{d\delta} \\ \frac{d(\hat{\theta}_{z_\delta,-z})}{d\delta} &\approx -\frac{1}{n}\mathcal{H}_{\hat{\theta}}^{-1}\nabla_x\nabla_\theta\mathcal{L}(z,\hat{\theta}) \end{aligned} \tag{7}$$

We replace the previous result in (6) :

$$\mathcal{I}_{\text{pert,loss}}(z,z_{\text{test}})^\top = -\frac{1}{n}\nabla_\theta\mathcal{L}(z_{\text{test}},\hat{\theta})\mathcal{H}_{\hat{\theta}}^{-1}\nabla_x\nabla_\theta\mathcal{L}(z,\hat{\theta}) \tag{8}$$

2.2. Additional insights

The authors use the logistic regression model as an example to compare the insights one get from using euclidean distance versus using influence functions when searching for influential training points for a given prediction.

The euclidean distance (or cosine distance if the points are normalized) is given by $x \cdot x_{test}$.

The influence function that approximates the impact of up-weighting a training point on the loss, in a logit regression is given by:

$$y_{\text{test}} y \cdot \sigma(-y_{\text{test}} \theta^\top x_{\text{test}}) \cdot \sigma(-y\theta^\top x) x_{test}^\top \mathcal{H}_{\hat{\theta}}^{-1} x \quad (9)$$

where $\sigma(t) = \frac{1}{1+\exp(-t)}$

Looking at the factors of the influence function equation, we can see how it gives more information than the cosine:

- $\mathcal{H}_{\hat{\theta}}^{-1}$: The inverse Hessian (weighted covariance matrix) is responsible for up-weighting the influence of points which add information that is not redundant with information added by other points of the training set
- $\sigma(-y\theta^\top x)$: The loss term is responsible for down-weighting the influence of points which do not impact the predictions of the model

2.3. Making influence function calculation tractable

Computing influence function pose a computational challenge for state-of-the-art models with typically millions of parameters. Specifically, forming the Hessian costs $O(np^2)$ and inverting it costs: $O(p^3)$. Thus, getting the Hessian inverse in $\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta \mathcal{L}(z_{\text{test}}, \hat{\theta})^\top \mathcal{H}_{\hat{\theta}}^{-1} \mathcal{L}(z, \hat{\theta})$ costs $O(np^2 + p^3)$.

The authors present two techniques for approximating $s_{\text{test}} = \mathcal{H}_{\hat{\theta}}^{-1} \nabla_\theta \mathcal{L}(z_{\text{test}}, \hat{\theta})$; relying on a Hessian Vector Product technique introduced by B. A. Pearlmutter (Pearlmutter, 1994).

2.3.1. Conjugate gradients (CG)

The idea is to get $\mathcal{H}_{\hat{\theta}}^{-1} v$ by solving a minimization problem. The function to minimize is define by $\mathcal{H}_{\hat{\theta}}^{-1} v = \arg\min_t \left(\frac{1}{2} t^\top \mathcal{H}_{\hat{\theta}} t - v^\top t\right)$. The CG approaches can iteratively gives the value of the solution by evaluate $\mathcal{H}_{\hat{\theta}} t$ in $O(np)$ p times with $\theta \in \mathbb{R}^p$. However CG approaches can give us a good estimation in less iterations than p.

2.3.2. Stochastic estimation

While CG requires to go through n training points per iteration, stochastic estimation only samples a single point per iteration, reducing the cost dramatically on large datasets. We use the Taylor expansion to approximate H^{-1} :

$$(I - A)^{-1} = \sum_{i=0}^{\infty} A^i$$

Replacing A by $(I - \mathcal{H})$ we get :

$$(I - (I - \mathcal{H}))^{-1} = \sum_{i=0}^{\infty} (I - \mathcal{H})^i$$

$$\mathcal{H}^{-1} = \sum_{i=0}^{\infty} (I - \mathcal{H})^i$$

We can express the first j^{th} terms of the Inverse Hessian as per Taylor Expansion as $\mathcal{H}_j^{-1} = \sum_{i=0}^{j} (I - \mathcal{H})^i$ and recursively get the following expression : $\mathcal{H}_j^{-1} = I + (I - \mathcal{H})\mathcal{H}_{j-1}^{-1}$

By definition $\lim_{j->\infty} \mathcal{H}_j^{-1} = \mathcal{H}^{-1}$ and $E\left[\mathcal{H}_j^{-1}\right] = \mathcal{H}_j^{-1}$ Hence, $E[\mathcal{H}_j^{-1}] -> \mathcal{H}^{-1}$

In order to calculate $\mathcal{H}^{-1} v$, stochastic estimation tells us to sample z_i and estimate without bias as gradient of $\mathcal{L}(z_i, \theta)$ and to apply the following procedure : We initialize $\mathcal{H}_0^{-1} v = v$, and apply recursively the expression $\mathcal{H}_j^{-1} v = v + (I - \nabla_{\hat{\theta}} \mathcal{L}(z_i, \theta))\mathcal{H}_{j-1}^{-1} v$.

We recurse enough that the value of the Hessian stabilizes and we repeat the procedure r times and average the results.

Empirically, stochastic estimation is faster than CG.

2.4. Relaxation of convexity and differentiability constraints

We remind that, in order to use the influence function instead of leave-one-out retraining, we made the assumptions that the parameters θ of the model reach the global minimum of the empirical risk and that the empirical risk is twice differentiable and strictly convex. In this section we will see how we can relax those assumptions.

2.4.1. Non convex functions

The authors show by experiment that in logistic regression (GLM), the influence function does track change in loss for leave one out training. However, it is not obvious influence functions can be used when the loss function is non convex and does not converge.

The idea of the authors is to use quadratic approximation of the empirical risk which is by definition smooth and convex, and apply influence function on the risk approximation function. The authors argue that the influence function track training effects even when they use a quadratic approximation of the empirical risk. They experiment with comparing the influence of training point in a CNN learning model as per the influence function versus as per the actual leave-one-out training loss delta. We note that in this case, the influence function tended to overestimate the absolute influence of influential points and underestimate the influence of less influential points in the CNN model.

2.4.2. Non smooth functions

The authors also discuss working around a non smooth objective function. The idea presented in the paper consists in making a smooth approximation of the loss. They support this using the hinge function which cannot be derived in 0. They show that taking the hinge'(0) == 0 does not

allow the influence function to provide useful information about training effect. They then proceed to show how using smooth hinge with small enough temperature term allows to approximate an influence function that correlates highly with the actual influence of points.

3. Author's Use cases

The experiments presented in the paper have been implemented and shared by the authors on their official Github repository : https://github.com/kohpangwei/influence-release.
We were able to run them and we will discuss them briefly before talking about our own experiments. We applaud the authors extra-work for integrating the influence function functionalities with the popular tensorflow library, making them user-friendly.

3.1. Understanding Model Behavior

One of the most useful use of influence functions is to analyze models' predictions through the training points that had the maximum learning effect on the predictor for the prediction. The authors show the usefulness of this approach by comparing how two models can make similar predictions but using distinct sets of influential training points. Specifically, they go on to show that as one expects, in SVM models there is a relationship between how influential the training point and how close it is to the test point in euclidean space whereas there isn't in inception models. Indeed, inception models as other black box deep learning classifiers transform feature space across layer after layer and distance in inception space does not relate anymore with the distance in the pixel space. On the other hand, SVM use more shallow features that relates closely to the pixel wise distance.

This approach allows to interpret models behavior using training data where there wasn't any tools for that before with such compelling computational efficiency and interpretability. It makes it cheaper and more accurate than some distance metric, such as Euclidean distance for SVM, to track influential training points. One limit of this approach is that the interpretation of what makes a training point influential with respect of a set of test points is given by the relative influence of the input features whereas often time, black box models make prediction in a different feature space.

3.2. Adversarial effect

Influence functions can also be used to create visually imperceptible adversarial training attacks. Adversarial training attacks already existed, but influence functions allows to minimize the perturbation at the input space level. The idea is to use an influence function to approximates the effect of perturbing a training point on the loss for a test point or set of points. Influence functions give us the direction of input perturbation for a given training point that has the maximal negative learning effect on a test point or set of points. By iteratively perturbing the training point in the direction that damages most the predictor performance, we obtain a point near the original point in the input space but far in the higher level feature space of the predictor (e.g. final layer of a neural network).

This opens the possibility for visually undetectable training attacks in image recognition settings. Because this attack relies on building a perturbation in a direction of low variance so that the model overfits the perturbed training point, it is less effective with more training data. Conversely one can measure the vulnerability of a novel to such adversarial training attack by observing a distribution of learning influence across the training points. If the influence is concentrated on few points, it makes it easy to perturb those few points only and flipped the prediction on many test points.

3.3. Debugging domain mismatch

Difference in distribution between train set and test set can cause models to perform poorly in terms of generalization error. The authors demonstrate how influence functions can be leveraged to track the most influential training points in causing a specific misclassification. Then, for each of these points, influence functions can also be used to track which of their features are most responsible for their influence in the misclassification.

3.4. Fixing mislabeled examples

Influence functions offer an efficient way to identify the training points that have the most learning effect on the model. Checking training set labels is expensive so we want to prioritize the verification of training points. Ultimately using influence functions to identify which points to verify allows to quickly increase test accuracy by investing the verification efforts in the points that matter most first.

Another approach to identifying wrongly labeled train point is prioritizing the verification by checking points with highest loss. This approach is effective in terms of finding which points are likely misclassified. However, when evaluating in terms of gains in test accuracy of the model, the influence function offer a better alternative in that it targets the points that exert the most influence on the model.

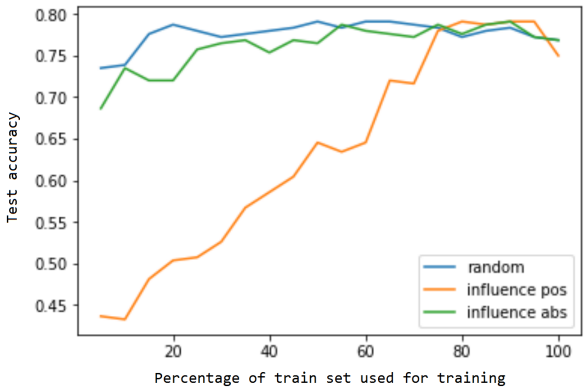

Figure 1. Impact of using $\mathcal{I}_{\text{up,loss}}$ on test accuracy

4. Our Use Case

4.1. Relationship between gains in test accuracy and influence in training set

In this section, we try to experiment on how test prediction accuracy evolves depending on the choice of training set. Specifically, our approach consists in training on an incrementally larger set of points. This allows to compare how the choice of training points impact on learning performance. We can choose the training points either randomly, or from the most influent to the less influent using influence functions.

In **Figure 1** we show the evolution of test prediction accuracy with respect to the size of the training dataset and highlight the impact of influential points on learning. The **blue** line represents the prediction accuracy of the model trained on random sets of points and is used as a baseline in our experiment. The training subsets used for the **orange** line is composed by the most influential training point on average influence over the test set (sorted by descending order of $\frac{1}{t}\sum_{i=1}^{t}\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}_i})$ with t the number of point in the test set). The training subsets used for the **green** line is based on the least influential point (sorted by ascending order of $\left|\frac{1}{t}\sum_{i=1}^{t}\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}_i})\right|$)

We can notice that by choosing randomly the training points, we can learn a better estimator than by choosing the most influential points for training. We are able to match the baseline performance by using the subset of the least influential points. One observation we can make is that the set of least influential points matches better the distribution of the original dataset than the set of most influential points. It is expected that the subset of the most influential points do not match the overall data distribution. For instance, outliers tend to be influential as up-weighting them in the training increases test lost.

4.1.1. Details of the methodology

We use the toy example of the classification problem using the Kaggle Titanic dataset in which one is interested in predicting, given an individual attributes, if they survived the accident or not.

We split the data set into train and test set and normalize the input data. We then fit an estimator of type *BinaryLogisticRegressionWithLBFGS* as defined in the authors official repository using the train data. We then determine the most influential points on the test predictions. For that, we use the function *get_influence_on_test_loss()* of the learnt predictor over the entire test set in order to get $\frac{1}{t}\sum_{i=1}^{t}\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}_i})$.

We sort train points by positive influence as well as ascending absolute influence on the test loss. These lists respectively represent the points with most impact on the test loss and the points with least impact on the test loss. We also introduce a list of train point in random order.

We train the model using the different sets progressively increasing the number of training points used. We then plot the results.

5. Discussion

Upon examining the use cases, we found that while some are actual end-to-end solutions to a given problem, such as crafting adversarial training points or sorting training points by influence for checking their labels, others are tools to help humans work with models, for instance by interpreting predictions in terms of training input or debugging suspected model mismatch. We suspect there is room for research to leverage the work of the authors in order to alleviate the need for human evaluation of the insights and creating a more systematic and objective approach to interpret model predictions.

If you want to look at our code or run it, you can find it on github at the following link :
https://github.com/eolecvk/InfluenceFunctions

References

Koh, Pang Wei and Liang, Percy. Understanding black-box predictions via influence functions, 2017.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. 2012.

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