GraSS: Scalable Influence Function with Sparse Gradient Compression

A Foray to Data Attribution and Influence Function

Pingbang Hu[1] Joseph Melkonian[2] Weijing Tang[3] Han Zhao[1] Jiaqi W. Ma[1]

[1]University of Illinois Urbana-Champaign [2]Womp Labs [3]Carnegie Mellon University

July 25, 2025

Table of Content

Table of Content

Predicting the Future

We start with some abstract nonsense:

Problem

What's the most fundamental aspect in all scientific problems?

Lots of the time, the answer I would give is *prediction*:

Example

1. **Physics**: What would happen if we throw a ball with the given initial condition?
2. **Chemistry**: What properties emerge if we combine these two materials?

What about Computer Science, or more specifically, AI/ML nowadays? Seems straightforward:

- We build **models** to predict all sorts of things:
- E.g., image classification, text completion, weather forecast, etc.

One Level Up: A Meta Question

What if we zoom out and ask a *meta* question:

Problem

*What's the most fundamental aspect in **solving** all scientific problems?*

Now this varies based on the subject:

Example

1. **Physics**: Should we use Hamiltonian or Lagrangian mechanics instead of Newtonian?
2. **Chemistry**: Should we analyze atomic behavior or use other approaches?

As for AI/ML, a similar question is then "How should we build our models differently?"

- Based on experience/intuition/small-scale trial/etc.?

Consequences

You might not realize, but this sort of meta question is quite important:

- **Physics/chemistry**: wrong approaches might lead to a long detour
- **AI/ML**: wrong models might lead to poor performance, wasted resources, etc.

To illustrate, let's think about what *Scaling Law* tells us:

Example (Scaling Law [Kap+20], in plain English)

If we throw more data & GPUs, we're (sort of) *guaranteed* to have a better model.

Basically, this tells us that:

- How to solve a problem more efficiently, without expending brain power (expensive)?

Table of Content

Counterfactual Prediction

All above can be framed as *counterfactual predictions*:

- Don't want to actually do it just to know what will happen;
- Want to have a good estimation before we execute the potentially expensive plan

Example

There are many other directions besides the scaling law:

- **Meta learning**: tuning hyperparameters $\Rightarrow$ actual learning algorithms
- **Neural architecture optimization**: optimizing architecture $\Rightarrow$ actual model to train
- **Data attribution**: curating dataset $\Rightarrow$ actual learned model's statistics

We focus on the last one, *data attribution*, in this presentation.

Data Attribution

Data attribution algorithms quantify *counterfactual effect* for **dataset perturbation**:

- Say we have a model $\hat{\theta}_D$ trained on D, with $p = |\hat{\theta}_D|$ and $n = |D|$
- Given a quantity of interest—a *target* function $f(D)$ of $\hat{\theta}_D$, e.g., validation loss
- Predict how f will change, if the dataset D is *counterfactually* perturbed to D':

$$\Delta f = f(D') - f(D).$$

Popular methods study this from a fine-grained, localized viewpoint:

1. Consider D' of the form $D' = D \setminus B$ for a small batch of samples B (or $D' = D \cup B$)
2. For each possible B, we predict $\tau_f(B) := f(D \setminus B) - f(D)$ (or $f(D \cup B) - f(D)$)

Popular choice of B: $B_i = \{z_i\}$ for $z_i \in D$, i.e., $\tau_f(B_i)$ provides the *point-wise* effect.

Data Attribution

As previously seen

$\tau_f(B)$ gives the counterfactual effect of f when B is removed from the training set D.

Predicting τ_f provides a way to understand the final model's properties, **without training it**!

Example (Different properties)

- **Performance**: f is validation loss $\Rightarrow$ *predict loss decrease* (or increase) when including B in D
- **Safety**: f is loss on safety-critical sample $\Rightarrow$ ...
- **Bias**: f is a bias metric over under-performed groups $\Rightarrow$ ...

Data attribution has been explored in many directions:

- Data selection/cleaning, data poisoning, fact tracing, data compensation, etc.

Table of Content

Models Parametrized by Dataset Weight

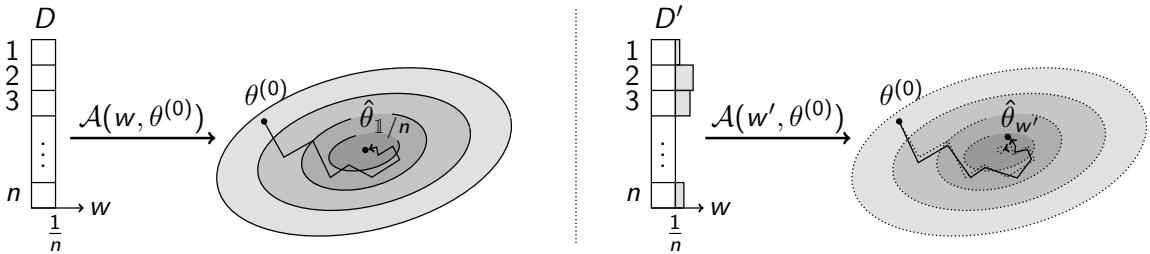
We now see how one can estimate τ_f. One idea is the following:

Intuition

Parametrize D by a default weight vector $w = \mathbb{1}/n \in \mathbb{R}^n$ for the data points z_i's.

$\Rightarrow$ *Model trained on (weighted) D is a function of w: $\hat{\theta}_w = \arg\min_\theta \sum_{z_i \in D} w_i \ell(z_i; \theta)$*

$\Rightarrow$ *Taylor-expand $\hat{\theta}_w$ around $w = \mathbb{1}/n \Leftrightarrow$ estimating perturbation effects ($D \to D'$)*



Counterfactual Prediction from Freshman Calculus

To estimate $\tau_f(\{z_i\}) = f(D \setminus \{z_i\}) - f(D)$:

- Write $D \setminus \{z_i\}$ as $D - \frac{1}{n} z_i \Rightarrow \tau_f(\{z_i\}) = f(D + \epsilon z_i) - f(D)$ with $\epsilon = -1/n$!

Since $\hat{\theta}_w$ is a function of w, so is $f(w)$:

1. From **first-order approximation** (i.e., Taylor expansion):

$$\Delta f = \tau_f(\{z_i\}) = [f(D + \epsilon z_i) - f(D)]|_{\epsilon=-\frac{1}{n}} \approx \epsilon|_{\epsilon=-\frac{1}{n}} \cdot \left.\frac{\mathrm{d} f(\hat{\theta}_{+\epsilon z_i})}{\mathrm{d}\epsilon}\right|_{\epsilon=0}.$$

2. From **chain rule**:

$$\left.\frac{\mathrm{d} f(\hat{\theta}_{+\epsilon z_i})}{\mathrm{d}\epsilon}\right|_{\epsilon=0} = \nabla_\theta f(\hat{\theta}_{+\epsilon z_i})^\top\Big|_{\epsilon=0} \cdot \left.\frac{\mathrm{d}\hat{\theta}_{+\epsilon z_i}}{\mathrm{d}\epsilon}\right|_{\epsilon=0} = \nabla_\theta f(\hat{\theta}_{\mathbb{1}/n})^\top \cdot \left.\frac{\mathrm{d}\hat{\theta}_{+\epsilon z_i}}{\mathrm{d}\epsilon}\right|_{\epsilon=0}.$$

Theorem (Influence function [KL17; Gro+23])

Let $\hat{\theta} = \hat{\theta}_{\mathbb{1}/n}$ be the ERM trained on D and $H_{\hat{\theta}} = \frac{1}{n}\sum_{z_i \in D} \nabla^2_\theta \ell(z_i; \hat{\theta})$ be the empirical Hessian. The influence function of upweighting $z_i \in D$ on the target function f is:

$$\mathcal{I}(z_i, f) := \left.\frac{\mathrm{d} f(\hat{\theta}_{+\epsilon z_i})}{\mathrm{d}\epsilon}\right|_{\epsilon=0} = \nabla_\theta f(\hat{\theta})^\top \left.\frac{\mathrm{d}\hat{\theta}_{+\epsilon z_i}}{\mathrm{d}\epsilon}\right|_{\epsilon=0} = -\nabla_\theta f(\hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta \ell(z_i; \hat{\theta}).$$

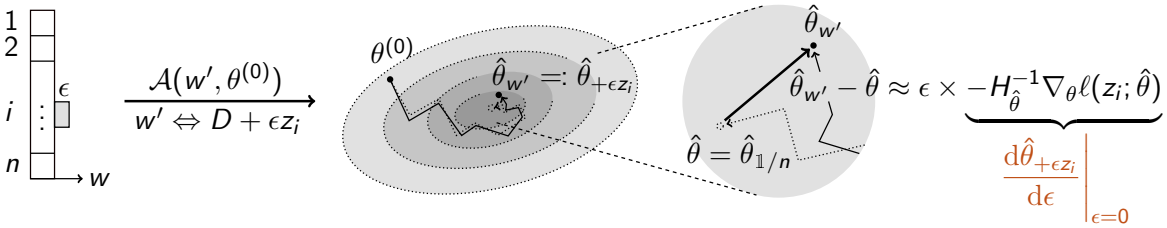


Table of Content

Computing Influence Function

As previously seen (Influence function)

Counterfactual prediction of removing z_i is $\Delta f = \tau_f(\{z_i\}) \approx \epsilon \cdot \mathcal{I}(z_i, f)$ with $\epsilon = -1/n$, where

$$\mathcal{I}(z_i, f) = -\nabla_\theta f(\hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta \ell(z_i; \hat{\theta}), \quad H_{\hat{\theta}} = \frac{1}{n} \sum_{z_i \in D} \nabla_\theta^2 \ell(z_i; \hat{\theta})$$

The main computation is the *inverse-Hessian-vector-product* $H_{\hat{\theta}}^{-1} \times \nabla_\theta \ell(z_i; \hat{\theta})$, or iHVP:

Remark

Once iHVP is solved, $\tau_f(\{z_i\})$ can be computed by efficient inner-product with $\nabla_\theta f$.

- Vector $\nabla_\theta \ell(z_i; \hat{\theta}) \in \mathbb{R}^p$: first-order gradient for all $z_i \in D$
- Inverse-Hessian $H_{\hat{\theta}}^{-1} \in \mathbb{R}^{p \times p}$: inverting a $p \times p$ second-order Hessian for all $z_i \in D$

Bottleneck of Influence Function

There are several bottlenecks for iHVP. First, the ***computation***:

- Computing all vectors $\{\nabla_\theta \ell(z_i; \hat{\theta})\}_{i=1}^n$ requires $O(np)$
- Computing inverse-Hessian $H_{\hat{\theta}}^{-1}$ requires $O(np^2 + p^3)$
- Computing product requires $O(np^2)$

Next, the issue of ***storage***:

- Storing all vectors $\{\nabla_\theta \ell(z_i; \hat{\theta}) \in \mathbb{R}^p\}_{i=1}^n$ requires $O(np)$.
- Storing inverse-Hessian $H_{\hat{\theta}}^{-1}$ requires $O(p^2)$

Remark (Main bottleneck)

Respectively, the main bottlenecks are:

- ***Computation***: *inverse-Hessian* $O(np^2 + p^3)$
- ***Storage***: *vectors* + *inverse-Hessian* $O(np + p^2)$

Scalable Approximation: FIM

To mitigate the bottleneck of inverse-Hessian:

Theorem (Fisher information matrix)

For cross-entropy loss, in expectation, the fisher information matrix (FIM) $F_{\hat{\theta}}$ equals $H_{\hat{\theta}}$, where

$$F_{\hat{\theta}} := \frac{1}{n} \sum_{z_i \in D} \nabla_\theta \ell(z_i; \hat{\theta}) \nabla_\theta \ell(z_i; \hat{\theta})^\top.$$

We see that using FIM approximation:

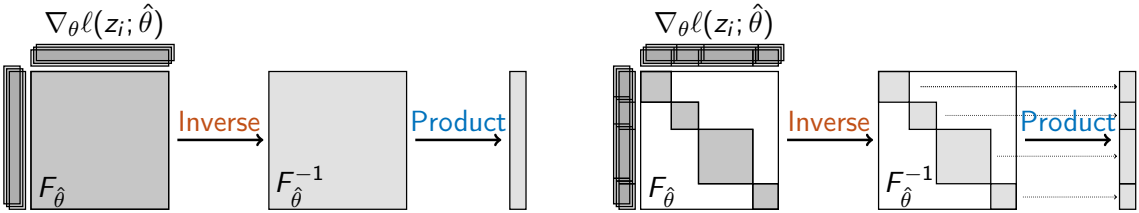
- No higher-order differentiation; computation drops from $O(np^2)$ to $O(np)$ (overlaps with vectors)
- Inverting still requires $O(p^3)$, as well as storage $O(p^2)$

Scalable Approximation: Block-Diagonal FIM

To further speed up inverse-Hessian, we need to *break* $F_{\hat{\theta}}$:

- **Structural assumption**: layers are *independent* $\Rightarrow$ $F_{\hat{\theta}}$ is *block-diagonal* (and hence $F_{\hat{\theta}}^{-1}$)
- Inverse and product can now be done *layer-wise*!

If you enjoy figures...



Remaining Bottlenecks

Remark (Main bottleneck for block-diagonal FIM)

Respectively, the main bottlenecks are:

- ***Computation***: *vectors* + *inverse-FIM* + *product* $O(np + p^3/L^2 + p^2/L) = O(np + p^3/L^2)$
- ***Storage***: *vectors* + *inverse-FIM* $O(np + p^2/L)$

Is this enough?

- Computation-wise, inverse-FIM $O(p^3/L^2)$ might be okay?
- Storing vectors is challenging: $O(np)$ for 1B model with 1B token dataset $\approx$ 4PB

Intuition

*The p is the key bottleneck for **storage**, as it is potentially large.*

Random

The main bottleneck now becomes potentially large p for $\nabla_\theta \ell(z_i; \hat{\theta})$:

- If we can operate with vectors of dimension $k \ll p$

$\Rightarrow$ **Storage**: replacing p with k! **Computation**: with some overhead

Intuition

As ML people usually do, we randomly project $\nabla_\theta \ell(z_i; \hat{\theta}) \in \mathbb{R}^p$ down to some $k \ll p$.

This idea is known as **Random** [Woj+16]. Combining with all previous approximation on FIM:

Remark (Main bottleneck for block-diagonal FIM with Random (LoGra [Cho+24]))

Respectively, the main bottlenecks are:

- ***Computation***: $O(np + p^3/L^2) \to O(np + k^3/L^2)_{(+O(npk/L)!!)}$
- ***Storage***: $O(np + p^2/L) \to O(nk + k^2/L)$

Table of Content

What's Next?

This concludes the current SOTA data attribution algorithms based on influence function.

Problem (Computational overhead)

LoGra *has an additional $O(npk/L)$ overhead. Can we optimize this?*

Yes! By tailoring the projection method specifically to per-sample gradients $\nabla_\theta \ell(z_i; \hat{\theta})$:

Theorem (GraSS & FactGraSS [Hu+25])

There is a sublinear compression-based influence function algorithm with an overhead of

$$O(nk'), \text{ where } k < k' \ll p.$$

This extends to highly-optimized linear layers, where layer-wise gradients are never materialized.

Q&A Time!

Thanks! Ask ***anything*** you want!

Table of Content

References

[Cho+24] Sang Keun Choe et al. *What Is Your Data Worth to GPT? LLM-Scale Data Valuation with Influence Functions*. May 22, 2024. DOI: 10.48550/arXiv.2405.13954. arXiv: 2405.13954 [cs]. URL: http://arxiv.org/abs/2405.13954 (visited on 09/14/2024).

[Gro+23] Roger Grosse et al. "Studying large language model generalization with influence functions". In: *arXiv preprint arXiv:2308.03296* (2023).

[Hu+25] Pingbang Hu et al. "GraSS: Scalable Influence Function with Sparse Gradient Compression". In: *arXiv preprint arXiv:2505.18976* (2025).

[Kap+20] Jared Kaplan et al. "Scaling laws for neural language models". In: *arXiv preprint arXiv:2001.08361* (2020).

[KL17] Pang Wei Koh and Percy Liang. "Understanding black-box predictions via influence functions". In: *International conference on machine learning*. PMLR. 2017.

[Woj+16] Mike Wojnowicz et al. ""Influence Sketching": Finding Influential Samples in Large-Scale Regressions". In: *2016 IEEE International Conference on Big Data (Big Data)*. 2016 IEEE International Conference on Big Data (Big Data). Washington DC,USA: IEEE, Dec. 2016, pp. 3601–3612. ISBN: 978-1-4673-9005-7. DOI: 10.1109/BigData.2016.7841024. URL: http://ieeexplore.ieee.org/document/7841024/ (visited on 12/06/2023).