Scaling Up Influence Functions

Andrea Schioppa*, Polina Zablotskaia, David Vilar, Artem Sokolov

Google Research
{arischioppa, polinaz, vilar, artemsok}@google.com

Abstract

We address efficient calculation of influence functions for tracking predictions back to the training data. We propose and analyze a new approach to speeding up the inverse Hessian calculation based on Arnoldi iteration. With this improvement, we achieve, to the best of our knowledge, the first successful implementation of influence functions that scales to full-size (language and vision) Transformer models with several hundreds of millions of parameters. We evaluate our approach on image classification and sequence-to-sequence tasks with tens to a hundred of millions of training examples. Our code will be available at https://github.com/google-research/jax-influence.

1 Introduction

Recognizing data's highest agency in defining deep neural networks' (DNNs) performance, the pursuit of state-of-the-art has made datasets for training modern DNNs grow to sizes that can no longer be curated by humans. This has acutely aggravated data issues like noise and mislabeled data: Noise is characteristic of tasks where training data is crawled from the Web (e.g. machine translation) and where golden-truth labels are heuristically paired to inputs (Uszkoreit et al. 2010), leaving ample room for errors and inheriting biases of the heuristic. Wrong labels can be also introduced by non-expert crowd annotators who, considering the amount of data to be labeled, are hard to incentivize for quality within available budgets (Bowman and Dahl 2021).

Given the above, a natural way to interpret and fix DNN models is to track their bad (or good) predictions down to the training examples that caused them (Cook and Weisberg 1980; Koh and Liang 2017; Yeh et al. 2018), and take appropriate action on the found examples or annotation policies. Addressing this, Koh and Liang (2017) proposed *influence functions* (IFs) as a theoretically motivated method, grounded in robust statistics (Cook and Weisberg 1982), of quantifying the effect of training examples on predictions: For a query example z, IFs estimate the most influential example x in training data $\mathcal{D}$, in terms of absolute change of loss L if x were infinitesimally up-weighted in $\mathcal{D}$, with:

$$\mathcal{I}_H(x, z) = \langle \nabla_\Theta L(z), H^{-1} \nabla_\Theta L(x) \rangle, \tag{1}$$

where $H = \nabla^2_\Theta L$ is the Hessian of the model at parameters Θ. The straight-forward IF implementation, using the approximate Hessian inversion procedure LISSA (Agarwal, Bullins, and Hazan 2017), has $O(p)$ memory and $O(r \cdot p)$ time complexities, where r is the LISSA iteration count and $p = |\Theta|$, incurred at every x. Besides the need of careful tuning of LISSA, the $O(p)$-memory has been the major obstacle on the way of IF deployment for debugging application-relevant DNNs with *hundreds of millions* (or more) training examples and model parameters; so noise or mislabeling issues remain unfixed or even undetected, and are adversely impacting predictions.

In this work, we focus on reducing the IF memory footprint by not materializing $O(p)$-size gradients nor Hessians, and decoupling the required number of H estimations, $O(r \cdot |\mathcal{D}|)$, from the training data size. This allows to parallelize computation over larger b and scale to huge datasets and models. Specifically, we use Arnoldi iteration (Arnoldi 1951) to find the dominant (in absolute value) eigenvalues of H and their orthonormal eigenvectors on a random data subset, $|\mathcal{D}'| \ll |\mathcal{D}|$, and then cheaply invert the diagonalized H, avoiding calls to LISSA as well as its convergence and stability issues. As H is Hermitian, (1) is symmetric w.r.t. x and z, so previous work cached $\{\nabla_\Theta L(x)\}$ to improve IFs usability (Guo et al. 2021), however, it only spares one backward pass per x and requires to re-estimate the product of the (unstorable) H^{-1} with $\nabla_\Theta L(z)$ every time an influence on z is requested. The crux of our approach is in caching instead H in the trivially-invertable diagonalized form and for the small-dimensional subspace spanned by a few dominant eigenvectors, $\tilde{p} \ll p$. Hessian-gradient products are then reduced to simple scalar-gradient products, which do not need to be materialized in memory as (1) can now be implemented with Jacobian-vector products. In summary, our approach renders repeated re-estimations of $H^{-1}\nabla_\Theta L(x)$ at every x unnecessary and they are replaced with the memory- and time-efficient forward-mode differentiation.

Empirically, IFs with Arnoldi iteration achieve speed-ups of 3-4 orders of magnitude over the LISSA-powered IFs (Koh and Liang 2017) and of 10x over TracIn (Pruthi et al. 2020), a heuristic gradient-only alternative to IFs (§5),

*Google AI Resident.

with better or similar accuracy. With this improvement, we successfully evaluated IFs on both language and vision, full-size Transformer models (up to 300M of parameters) in image classification and sequence-to-sequence tasks, resp., on 14M (ImageNet) and 100M (Paracrawl) training examples.

Note that the standard conditions for (1) to be a correct influence estimate, i.e. locally strictly-convex $L \in \mathcal{C}^2$ (Koh and Liang 2017), remain in place and their fulfilment depends on the concrete task, network, training algorithm and its convergence status. The time/memory complexity also remain, however, our contribution improves constants hidden in the O-notation, and thus permits IF evaluation on full data/models that are relevant in applications, on standard memory-limited hardware. This opens the way for developers to make informed decisions if IFs are appropriate for their task, rather than to resort to heuristics from the start. This is encouraging, since the existing brute-force recipe of "soothing" the $O(p)$ complexity by subsetting parameters, e.g. focusing on few layers only (Chen et al. 2020), is prone to producing incorrect results (Feldman and Zhang 2020) (see also §5.2). On the other hand, running IF on subsets of $\mathcal{D}$ to reduce runtime (Guo et al. 2021) may introduce unwanted biases, misattribute prediction failures, and would not be enough for identifying mislabeled examples (Pruthi et al. 2020) or "hard" examples requiring memorization (Feldman and Zhang 2020). Yet, these approaches are compatible with our method and should result in compound speed-ups.

We will open-source our implementation of Arnoldi iteration at https://github.com/google-research/jax-influence.

2 Related Work

Explaining DNN predictions falls under a broader interpretability umbrella, where the lingering complexity of the data-explainability approach made research historically focus on instance-based methods, that explain predictions in terms of task-specific structural units of inputs, e.g. pixels or tokens. Rich literature offers different instantiations of the idea: gradient-based saliency maps (Simonyan, Vedaldi, and Zisserman 2013),input perturbations (Li, Monroe, and Jurafsky 2016) or LIME (Ribeiro, Singh, and Guestrin 2016), which fits a linear model in the inputs neighborhood. However, being limited to specific inputs, their insights are rarely actionable for system developers. And while it is possible to repurpose them to data explainability, e.g. via clustering of saliency maps (Lapuschkin et al. 2019), this solves a more difficult task than necessary, introduces new hyperparameters (incl. the saliency method itself) and relies on human experts to make sense of the clusters.

In contrast to instance-explanations in the form of token level heatmaps, the IF provides a method for tracing model predictions back to training examples. Existing approaches to reducing IF runtime mostly address salient problem axes – dimensionality of active parameters, cardinality of data subset or number of iterations – without addressing the procedure itself; or they drop theoretical foundations to use heuristics simpler than IF: e.g. Pruthi et al. (2020) reduce influence to tracking loss changes with the cumulative (over training model checkpoints, i.e. model snapshots) dot products of gradients, and in (Yeh et al. 2018) authors leverage kernel functions evaluated at the training samples for explaining inference decisions.

Mathematically, the closest to our work is (Ghorbani, Krishnan, and Xiao 2019) who use a specialization of Arnoldi iteration to Hermitian matrices (Lanczos iteration) to study the dynamics of spectra of entire Hessians at different snapshots during training. Because of this different goal, they use full-batch Hessians (i.e. computed on the full dataset), while we "spread" the Hessian approximation across smaller batches that do not cover the full $\mathcal{D}$. In result, we can work with larger models and datasets, e.g. ResNet50/ViT vs. ResNet18, and at a larger speed (simultaneously bumping the number of Lanczos/Arnoldi iterations from 90 to 200 to increase precision).

3 Influence and Influence Functions

The *true* influence of x on z, $\mathcal{I}_{\text{true}}(x, z)$, is defined as the change in the loss L at z between having learned the model parameters Θ without and with x in the training data (Cook and Weisberg 1980):

$$\mathcal{I}_{\text{true}}(x, z) = L(z|\Theta : x \notin \mathcal{D}) - L(z|\Theta : x \in \mathcal{D}).$$

Explicit calculation of $\mathcal{I}_{\text{true}}(x, z)$, by removing every x and retraining, is infeasible for large $\mathcal{D}$ and several approximation techniques have been proposed. Feldman (2020) and Feldman and Zhang (2020) propose to train multiple models on randomly selected data subsets while tracking, for each x, to which subsets it belonged; this way, one can obtain an unbiased estimator $\mathcal{I}_{\text{mem}}(x, x)$ of $\mathcal{I}_{\text{true}}(x, x)$ which, however, requires a substantial amount of model re-trainings (up to thousands to satisfy theoretical guarantees (Feldman 2020)).

Koh and Liang (2017) advocated the use of IFs in (1) that approximate the loss change after an infinitesimal up-weighting of x in $\mathcal{D}$. For models used in practice, H cannot be materialized in memory, let alone be inverted by standard linear algebra. However, for a fixed vector v, the Hessian-vector product (HVP), Hv, can be computed in $O(b \cdot p)$ time and memory (Pearlmutter 1994), where b is the batch size and defines the number of training examples on which H (of an implicitly given loss L) will be approximated. HVP is commonly implemented in modern autodiff toolkits (Baydin et al. 2018) as the reverse-mode differentiation Jacobian-vector product (JVP), followed by a forward-mode JVP.

Repeated HVP calls are the workhorse of the iterative procedure `LISSA` (Agarwal, Bullins, and Hazan 2017), used by (Koh and Liang 2017), that estimate inverse HVP as:

$$H_r^{-1}v = v + (I - H)H_{r-1}^{-1}v,$$

where H is approximated on random batches and v is a gradient. Even for small r, the procedure is both time- and memory-expensive as the $O(p)$-memory of HVP on explicitly instantiated v forces to estimate H on a *single sampled training point per iteration* ($b = 1$), impacting accuracy. Moreover, the total $O(r \cdot b \cdot p)$ time complexity will be incurred at *every* x in whose influence on z we are interested.

Evaluating influence methods. A practical problem with influence-based explainability is the absence of ground-truth to verify that a method produces correct results. In this paper we use two proxies, following the assumption that xs with high self-influence $\mathcal{I}_H(x,x)$ correspond to data outliers (Koh and Liang 2017; Pruthi et al. 2020): we either introduce a known synthetic data corruption and check for its correct retrieval by a method, or filter high-influence points out and measure a change in downstream task metrics (Guo et al. 2021; Kocijan and Bowman 2020). Using $\mathcal{I}_H(x,x)$ as a retrieval score for corrupted data, we measure the retrieval quality as areas under the ROC and the precision-recall curves, respectively denoted as the Area Under the Curve (AUC) and the Average Precision (AP).

4 Scaling Influence Functions

From the discussion above, the $O(p)$-memory complexity is the major bottleneck for efficient implementation of IFs. We start with an overview of existing techniques, showing their commonalities. Note that all are compatible with the approach we propose.

Caching. For data-interpretability purposes one might consider limiting (1) to a sufficiently promising subset $\mathcal{D}' \subset \mathcal{D}$, e.g. Guo et al. (2021) define $\mathcal{D}'$ as the top-k ℓ_2-neighbors of z in $\mathcal{D}$. Besides more hyperparameters, this still requires computing $H^{-1}\nabla_\Theta L(z)$ for every z and, as discussed above, reducing $\mathcal{D}'$ would not be enough for applications that require computing $\mathcal{I}_H(x,z)$ on *all* training x. As H is symmetric, one could swap x and z, and cache $H^{-1}\nabla_\Theta L(x)$ instead, bringing down the query complexity having only to compute $\nabla_\Theta L(z)$ now, but this would just shift the computational burden to building the search index over $H^{-1}\nabla_\Theta L(x)$.

Restricting parameters. Reducing required memory is possible by naively limiting the computation to a smaller subset of parameters of cardinality $\tilde{p}$, e.g. selecting one or few layer(s); usually, the last layer is selected (Koh and Liang 2017). This has two drawbacks: the choice of layers becomes a hyperparameter and the viable values of $\tilde{p}$ will depend on the model architecture, and as Feldman and Zhang (2020, §3.6) show using just one layer can result in different influence estimates compared to the full model.

Random projections. For simplification one might assume $H = I$ and reduce influence estimates to dot products of gradients. To account for multiple layers at once and to get a finer-step control of $\tilde{p}$, we consider a simple baseline, `RandSelect`, which randomly selects $\tilde{p}$ parameters $\tilde{\Theta} \subset \Theta$ and computes influence using the final checkpoint and gradients with respect to $\tilde{\Theta}$, and can be combined with layer selection. The `RandSelect` estimator can be equivalently expressed as

$$\mathcal{I}_G(x,z) = \langle G\nabla_\Theta L(x), G\nabla_\Theta L(z)\rangle, \qquad (2)$$

where $G \in \mathbb{R}^{\tilde{p}\times p}$ is a row selection matrix of the gradient's components corresponding to $\tilde{\Theta}$.

We also use another sketching (Woodruff 2014) baseline, `RandProj`, initially proposed by Wojnowicz et al. (2016) for generalized linear models: for a random Gaussian projection matrix G, $\mathbb{E}[G^TG] = I$, which leads to an unbiased estimate in (2). Since normally $\tilde{p} \ll p$, it allows a memory-efficient implementation in the forward mode: one just estimates $\tilde{p}$ JVPs with the rows of G that avoid materializing $O(p)$-size gradients. This has a lower memory footprint than `RandSelect`, which requires back-propagation as its $\tilde{p}$ (e.g. one layer) is still of the same order as p (see §5.2).

Tracing updates. A development of the $H = I$ idea is proposed in (Pruthi et al. 2020), where influence approximation works for the case when one can trace changes to the loss L across *all* gradient steps. In order to make this feasible, they propose the `TracIn` estimator, defined on a subset of gradient steps:

$$\mathcal{I}_{\mathrm{TracIn}}(x,z) = \frac{1}{C}\sum_{i=1}^{C}\langle \nabla_{\Theta_i} L(x), \nabla_{\Theta_i} L(z)\rangle,$$

where the $\{\Theta_i\}$ is a set of C checkpoints. Note that the complexity of `TracIn` is C times that of using exact gradient similarity and, as discussed in (Pruthi et al. 2020), care needs to be taken in selecting checkpoints. Another practical obstacle to `TracIn` is that, when analysing publicly released models, usually only the final model checkpoint is provided.

Compatible projections. `RandProj` assumes the full-dimension Hessian, $H = I$; if we drop this requirement, we might consider H restricted to the subspace S_G which is the image of G, and work with $G \cdot H \cdot G^T$ instead of the larger H. However, S_G is not in general H-invariant which can lead to approximation errors as when H is applied to a vector $v \in S_G$, the result might have non-negligible components orthogonal to S_G. We will see an example of this later in the experiments on eigenvalue retrieval for MNIST (Figure 1) where `RandProj` requires a considerably larger $\tilde{p}$ than `Arnoldi` to retrieve the top-$\tilde{p}$ eigenvalues of H.

Our approach. We propose to use the standard technique of building an approximately H-invariant subspace by selecting an arbitrary (e.g. random) vector $v \in \mathbb{R}^p$ and constructing the n-th order Krylov subspace: $K_n(H;v) = \mathrm{Span}\{v, Hv, H^2v, \ldots, H^nv\}$. The Arnoldi iteration (Arnoldi 1951) additionally builds an orthonormal basis for $K_n(H;v)$, so that the diagonalization of the restriction $\tilde{H}$ of H to $K_n(H;v)$ yields an approximation of the largest (in absolute value) eigenvalues of H and of the corresponding eigenvectors (Trefethen and Bau 1997, Ch. 33-34). Assuming n is large enough to estimate the largest $\tilde{p}$ eigenvalues, in summary we obtain a projection matrix G and work with $\tilde{H} = G \cdot H \cdot G^T$, which is a smaller dimensional matrix. We will call this algorithm `Arnoldi`, with the pseudocode in Algorithm 1.

The common feature of `RandProj` and `Arnoldi` is that, instead of working with the full gradient $\nabla_\Theta L(x)$, one takes the JVPs $\langle g_i, \nabla_\Theta L(x)\rangle$ with respect to the rows g_i of G. The implementation then becomes considerably more efficient as it can be done in the forward-mode differentiation and on larger batches. Moreover, in the case of `Arnoldi` the matrix H gets replaced with now diagonal $\tilde{H}$, simplifying the matrix inversion appearing in the definition of $\mathcal{I}_H$, and dispensing with the expensive `LISSA` procedure.

Error analysis. It remains to analyse the effect of using top-$\tilde{p}$ eigenvalues in `Arnoldi`. Recall that Koh and Liang (2017) derive (1) by minimizing the quadratic form $Q(\theta) = \frac{1}{2}\langle\theta, H\theta\rangle - \frac{1}{N}\langle\nabla_\Theta L(x|\Theta_0), \theta\rangle$, where Θ_0 are the parameters at convergence. Ordering the eigenvalues of H at Θ_0, $|\lambda_1| \geq |\lambda_2| \geq \cdots$ and letting $e_1, e_2, \cdots$ be the corresponding eigenvectors, Ghorbani, Krishnan, and Xiao (2019) empirically observe (and prove in the quadratic case) that gradient updates align with the subspace of H corresponding to the dominant λs. We provide two additional arguments in the same direction: we upperbound the error of approximating Q using such a subspace in Lemma 1, and discuss the effect of noise in H and the size of λ_k on applying H^{-1} to a vector in Lemma 2 (with proofs in §A).

Let Q_k be the form Q restricted to the H-subspace spanned by the top-k λs. We show that, as k increases, Q_k approximates Q better and the errors in directions of e_k corresponding to smaller $|\lambda_k|$ matter less[1]:

Lemma 1. *Q_k approximates Q by an error bounded by: $0 \leq Q(\theta) - Q_k(\theta) \leq \frac{1}{2}|\lambda_{k+1}|\|\theta\|_2^2$. Further, if minimizing Q introduces an error ε in the direction of e_{k+1} obtaining an estimate θ' for θ_*, then $Q(\theta') - Q(\theta_*) = \frac{\varepsilon^2}{2}\lambda_{k+1}$.*

Another way of looking at the same phenomenon is to consider the variance of estimated influence as the function of $|\lambda_k|$. Consider a computation of $y = H^{-1}u$, where vector u is known exactly. Assume also that H's estimation is noisy resulting in error $H + \delta H$, that $\mathbb{E}[\delta H] = 0$, and that the δH is isotropic (e.g. does not preferentially align with some e_k nor co-vary with λ_k). Then the variance of the estimator $\hat{y} = (H_{\Theta_0} + \delta H)^{-1}u$ in the direction of e_k is proportional to $|\lambda_k|^{-2}$:

Lemma 2. *The variance of $\hat{y}$ in the direction of e_k is* $\mathrm{Var}(\langle\hat{y}, e_k\rangle) \approx \frac{1}{|\lambda_k|^2}\mathrm{Var}(\langle\delta H e_k, y\rangle)$.

5 Experiments

5.1 Small Model & Data Scale: Digit Recognition

In this subsection, to be able to compare all baselines we pick the small MNIST dataset (LeCun, Cortes, and Burges 1994) and consider two CNNs of different sizes: a small one that permits the exact Hessian calculation, and a larger one on which we can gauge the scalability potential.

Because the influence calculation with `LISSA` and `TracIn` is slow, following (Koh and Liang 2017), we take two 10% subsamples of the original data for training and evaluation, and randomly relabel 20% of training examples to create a corrupted dataset to evaluate mislabeled example retrieval with influence estimates. Unlike (Koh and Liang 2017; Pruthi et al. 2020) we introduce the noise *before* training the models; by design, a perfect model on correctly labeled data would achieve only 80% accuracy on our eval set.

Small network. We re-implemented the small convolutional network with smooth non-linearities from (Koh and Liang 2017), and trained it following their recipe that is designed to make the assumptions behind the influence function method (Cook and Weisberg 1982) satisfied: First, to ensure convergence, the network is trained for more steps (500k) than one would normally do, with a large batch size of 500 images. Second, the ℓ_2-regularization of $5 \cdot 10^{-3}$ is introduced to make H positive definite. With only 3k parameters, it is a useful benchmark where it possible to compute H explicitly. We achieve accuracy of 73.8% and 92.3% on, resp., the corrupted and the true evaluation set.

Algorithm 1: `Arnoldi`

```
1: procedure ARNOLDI($v, n$)  ▷ Build orthonormal basis for the Krylov subspaces $K_n$.
2:     $w_0 \leftarrow \frac{v}{\|v\|_2}$
3:     $A_{l,m} \leftarrow 0$ for $0 \leq l \leq n$ and $0 \leq m < n$
4:     for $i \leftarrow 1, n$ do
5:         $w_i \leftarrow H \cdot w_{i-1}$  ▷ HVP in fwd-over-rev mode
6:         Set $A_{i,j} = \langle w_i, w_j\rangle$ for $j < i$
7:         $w_i \leftarrow w_i - \sum_{j<i} A_{i,j} w_j$  ▷ Orthogonalize
8:         $A_{i+1,i} \leftarrow \|w_i\|_2$, $w_i \leftarrow \frac{w_i}{\|w_i\|_2}$
9:     return: $A$, $\{w_i\}$
10: procedure DISTILL($A, \{w_i\}, \tilde{p}$) ▷ Distill $A, \{w_i\}$ to its top-$\tilde{p}$ eigenvalues.
11:     Discard the last row of $A$ and the last $w_n$
12:     Obtain $A$'s eigenvalues $\{\lambda_i\}$ and eigenvectors $\{e_i\}$
13:     Set $\{\lambda'_i\}$ to the $\tilde{p}$-largest (in absolute value) of $\{\lambda_i\}$
14:     Set $\{e'_i\}$ to the corresponding eigenvectors
15:     Set $G$ to the projection onto the spans $\{e'_i\}$ in $\{w_i\}$-basis
16:     return: $\{\lambda'_i\}$, $G$
17: procedure INFLUENCE($x, z, n, \tilde{p}$)  ▷ Influence of $x$ on $z$ with $n$ iterations and top-$\tilde{p}$ eigenvalues.
18:     $v \leftarrow \mathcal{N}(0, 1)$                          ⎫
19:     $A$, $\{w_i\}$ = ARNOLDI($v, n$)                  ⎬▷ Executed once and cached
20:     $\{\lambda'_i\}$, $G$ = DISTILL($A$, $\{w_i\}$, $\tilde{p}$)  ⎭
21:     $g_x \leftarrow G \cdot \nabla_\Theta L(x)$  ▷ fwd JVP for $x$ over $G$-rows
22:     $g_z \leftarrow G \cdot \nabla_\Theta L(z)$  ▷ fwd JVP for $z$ over $G$-rows
23:     $g_x \leftarrow g_x / \{\lambda'_i\}$  ▷ Multiply with diagonalized $\tilde{H}^{-1}$
24:     return: $\langle g_x, g_z\rangle$
```

In Figure 1 we compare the accuracy between the top-$\tilde{p}$ eigenvalues estimations obtained by `Arnoldi` (for $n = 200$) and `RandProj`. This illustrates the point made above that if the image subspace S_G associated with the projection G is not H-invariant, then eigenvalue estimates can be poor.

In Figure 2 we plot the retrieval quality of mislabeled examples by `Arnoldi` and `RandProj` as a function of $\tilde{p}$. The horizontal lines correspond to using the exact Hessian, `LISSA` and `TracIn` ($C = 25$ checkpoints, taken every 10 epochs). For this network we see that `Arnoldi` outperforms `RandProj` and steadily improves for further larger values, outperforming even the exact Hessian for a large enough $\tilde{p}$ (which can be explained by presence of close-to-zero eigenvalues in the exact H which affects its inverse).

[1]One might attempt `Arnoldi` on $H_{\Theta_0}^{-1}$ to obtain an approximation directly in the subspace of the top-k eigenvalues of $H_{\Theta_0}^{-1}$. We found this approach however to be less performant (see §A.1).

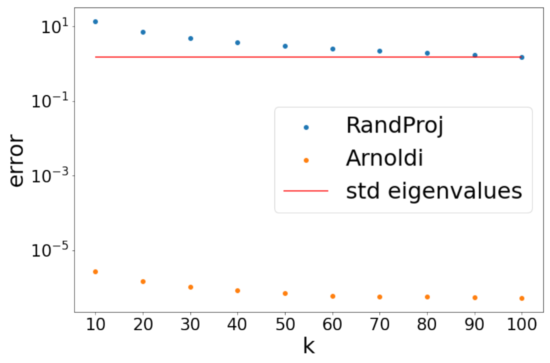


Figure 1: Estimation error of the top-$\tilde{p}$ eigenvalues obtained by DISTILL as the optimal transport (Wasserstein) distance to the exact H eigenvalues (assuming uniform distribution). The horizontal line is the standard deviation of the latter.

Method	$\tilde{p}$	T, secs	AUC	AP
`LISSA`, $r = 10$	-	4900	98.9	95.0
`LISSA`, $r = 100$ (10% Θ)	-	32300	98.8	94.8
`TracIn` [1]	-	5	98.7	94.0
`TracIn` [10]	-	42	**99.7**	**98.7**
`RandProj`	10	0.2	97.2	87.7
`RandProj`	100	1.9	98.6	93.9
`RandSelect`	10	0.1	54.9	31.2
`RandSelect`	100	1.8	91.8	72.6
`Arnoldi`	10	0.2	95.0	84.0
`Arnoldi`	100	1.9	98.2	92.9

Table 1: Retrieval of mislabeled MNIST examples using self-influence for larger CNN. For `TracIn` the C value is in brackets (last or all). All methods use full models (except the `LISSA` run on 10% of parameters Θ). For `RandProj` std deviation of AUC/AP estimates is 0.1/0.7 over 20 runs.

Larger network. To consider a more realistic and larger network we use the CNN from the Flax library[2] with 800k parameters, still small by industry standards, but for which H cannot be already computed explicitly. We train it for 10 epochs on GPU V100 without regularization, and achieve 75.4% on the corrupted and 94.8% on the true labels test set. This network is more realistic in size and in training procedure than the one in (Koh and Liang 2017).

Table 1 reports results of retrieval of mislabeled examples with the total scoring time T as there is a trade-off between gains in AP or AUC vs. T. As computing exact H was not possible, our baseline for self-influence scoring is `LISSA`, which is about 10^4 times slower than `Arnoldi` (which took 353 sec to estimate H for $\tilde{p} = 10$, $n = 200$ and $b = 512$).

We find that both `Arnoldi` and `RandProj` have a good trade-off between retrieval accuracy and speed, while `RandSelect`, despite being the fastest, suffers in retrieval quality. As here `RandProj` performs slightly better than `Arnoldi`, we initially hypothesized that this might indicate an insufficient accuracy of HVP estimations: To verify, we re-ran `Arnoldi` with the HVPs estimated on the *full* dataset and obtained almost identical results, indicating that, on the one hand, `Arnoldi` accurately estimates eigenvalues with only 512 examples per HVP and, on the other hand, for this network and task, gradient similarity methods might be more appropriate (cf. `LISSA` for $r = 100$ does not move the needle either). Finally, `TracIn` on 10 checkpoints (after each epoch) had the best retrieval quality, however, while its runtime on this model and dataset is acceptable, it becomes a problem for the larger models we consider later.

In §B.1, we show that images with $\mathcal{I}(x, z) < 0$ do appear ambiguous to humans or are incorrectly labeled. Results for the same CNN when over-trained or regularized are in §B.2.

5.2 Scaling with Data Size: Machine Translation

To test scalability over the data size dimension, we investigate IFs for machine translation focusing on data selection over millions of examples. We verify the hypothesis that the "cleanest" data is the one with the lowest self-influence (Koh and Liang 2017; Pruthi et al. 2020). We evaluate retrieval of artificially corrupted examples on the WMT17 dataset (6M sentences), and evaluate data cleaning on the large noisy Paracrawl corpus (100M sentences). `Arnoldi` used $b = 512$ for HVPs; $n = 200$ iterations took 139 minutes.

Retrieving mislabeled parallel data. We experiment with the Transformer Base model (Vaswani et al. 2017), implemented in Flax[3], on the clean WMT17 dataset for the German-English direction. We follow the original setup from (Vaswani et al. 2017) and train it for 100k steps on a 16-core TPUv2 (details in §C.2). As is standard in machine translation, the model is neither over-trained to full convergence nor we employ the ℓ_2-regularization, so a priori is not guaranteed if IFs that rely on Hessian approximations would fair better than the gradient heuristics, `RandProj` and `TracIn`. On the test set *newstest16* we obtained BLEU 36.0 after 100k training steps.

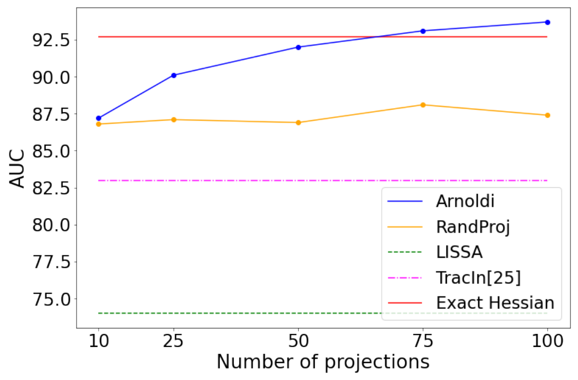
To construct a synthetic noisy dataset, we uniformly sampled 4096 examples from the training data and, for 256 examples of those, randomly shuffle the target sides, and repeat the above to obtain 5 data folds. We then apply different methods to compute self-influence scores and evaluate their quality with AUC and AP, averaging over the 5 data folds.

From Table 2 we observe that `RandSelect` performs the worst in terms of retrieval quality. `Arnoldi` outperforms `RandProj` and we observe that here AP is a measure more sensitive to differences than AUC. The memory footprint of `RandSelect` scales poorly: while we managed to run `Arnoldi` with $b = 512$ to compute self-influence, for `RandSelect` we had to reduce it to 64.

Finally, we consider the question of the quality of influence estimates obtained for subsets of layers. For large models it is common to reduce p by looking at the last few layers (Pruthi et al. 2020; Guo et al. 2021; Han, Wallace, and Tsvetkov 2020). However, Feldman and Zhang (2020) ob-

[2]https://github.com/google/flax/tree/master/examples/mnist

[3]https://github.com/google/flax/tree/main/examples/wmt



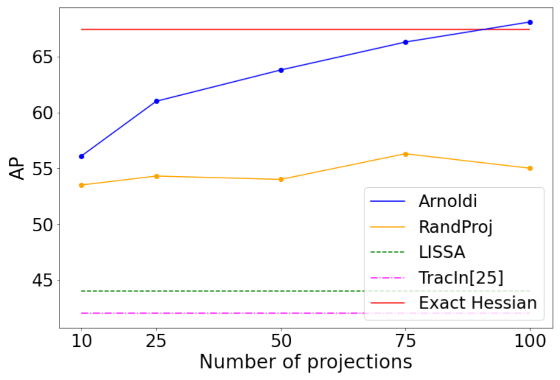


Figure 2: AUC and AP for retrieval of mislabeled MNIST examples as a function of $\tilde{p}$ for the small CNN model.

Layers	Method	$\tilde{p}$	AUC	AP
all layers	`RandSelect`	10	67.0	12.2
	`RandSelect`	100	79.3	19.7
	`RandProj`	10	85.6	31.3
	`RandProj`	20	85.2	28.0
	`Arnoldi`	10	92.0	47.8
	`Arnoldi`	20	**93.8**	**54.4**
last 3 decoder layers	`RandSelect`	100	80.2	20.1
	`RandSelect`	1000	81.3	22.2
	`RandProj`	10	80.6	23.5
	`RandProj`	20	83.0	25.8
	`Arnoldi`	10	82.0	28.1
	`Arnoldi`	20	**83.7**	**28.5**

Table 2: Retrieving synthetic mislabeled examples on WMT17. The standard deviation of AUC and AP estimates were under, resp., 0.9 and 1.0. for the `Rand*` methods.

served for their estimator $\mathcal{I}_{\text{mem}}$ a degradation of the self-influence estimates computed using only the last layer of ResNet50, conjecturing that memorization of difficult examples has already happened by the time the computation reaches the last layer. We corroborate their findings here for Transformers: using only the last three decoder layers we observe a significant degradation of the retrieval quality for all algorithms, with `Arnoldi` still outperforming the rest.

Filtering noisy training corpus. To simulate a realistic data cleaning setup, we take the noisy Paracrawl corpus from the WMT18 Parallel Corpus Filtering Task[4]. This dataset consists of 100M German-English sentence pairs with different kinds of noise that naturally occur in Web crawled datasets: sentence pairs in languages others than English or German; where both sides are in English or German; where the target is either not or only a partial source translation, or is a near copy of the source; or non-informative pairs, e.g. short sentences consisting of numbers.

[4]http://statmt.org/wmt18/parallel-corpus-filtering.html

We trained a model on WMT17 to bootstrap self-influence calculation and used *newstest2016* for evaluation of retrained models. The methods that scale to this data size are `Arnoldi`, `RandProj` and `RandSelect`, but as the latter underperformed on the retrieval of synthetically mislabeled data and has heavy memory consumption, we focused on the former two. We chose $\tilde{p} = 10$ as it is faster than $\tilde{p} = 20$, which did not substantially increase the scores in Table 2. With this, self-influence scoring speed was 2.2M examples/hour on a 16-core TPUv2 using $b = 2048$.

As filtering baselines we consider training on the full uncleaned data and some pre-filtering strategies from the AFRL, Alibaba and Microsoft submissions (Gwinnup et al. 2018; Deng et al. 2018; Junczys-Dowmunt 2018) that are detailed in §C.1. In Table 3 we report results after training on 1%, 5% and 10% of the data with the lowest self-influence, i.e. the cleanest data by assumption. We followed (Vaswani et al. 2017) and reported BLEU scores both at 10k steps and at the final 200k steps as the gap between data selection strategies might reduce over the course of training, possibly as gradient updates from cleaner data might be favored over time. Both `Arnoldi` and `RandProj` select cleaner data at the bottom of the 5% and 10% self-influence scores. Also `Arnoldi` outperforms `RandProj` gaining almost 4 BLEU points at 10k steps, and more than 8 points over the pre-filtering baseline at 200k steps. At 1%, we see a degradation in the performance at 200k steps and, inspecting the training loss, we found that it decreases by more than a half going from 5% to 1% data threshold. We conjecture that the selected examples at this strict 1%-threshold are too "simplistic" to provide useful translation patterns, in line with findings of Feldman and Zhang (2020, §3.3) on the marginal utility of data with low memorization values on ImageNet. See §C.3 for examples of high/low influence sentences.

Above, we did not aim to beat state-of-the-art cleaning pipelines, which employ a cascade of filtering stages, but to investigate whether a simple application of IFs can select better data to train on. Nevertheless, we evaluated the added value of `Arnoldi` by filtering 25% and 50% of the *clean* data selected by Microsoft's cascade (WMT18 winner), and

Method	% selected	BLEU@10k	BLEU@200k
None	100	9.9	17.8
Pre-filtering	14	11.7	22.6
`RandProj`	1	7.7	8.7
`Arnoldi`	1	17.8	19.6
`RandProj`	5	18.7	27.9
`Arnoldi`	5	24.0	30.3
`RandProj`	10	21.3	28.6
`Arnoldi`	10	**25.0**	**30.8**

Table 3: Data selection on the noisy Paracrawl corpus (100M parallel sentences) with evaluation on *newstest16*.

this increased BLEU from their 36.32 to, resp., 37.38 and 37.20 on *newstest16*.

5.3 Scaling with Model Size: Computer Vision

Here we empirically verify if `Arnoldi` scales well with the number of parameters for four state-of-the-art computer vision models of increasing sizes, and also run the mislabeled data retrieval experiment for the largest Vision Transformer.

Method performance. We considered ResNet50 and Vision Transformers (ViT): the Flax implementation[5] of ResNet50 has about 25M parameters[6], and for ViTs we used the JaX implementation[7] and took checkpoints trained on the mixture of ImageNet and ImageNet21k, that have between 86M (B32) and 300M (L32) parameters. Additionally, to cover an intermediate size between the ViT B32 and L32 we took a subset of the layers, starting from the top, to compute influence w.r.t. 50% of the parameters of the ViT L32, amounting to 150M of weights. For all models we used one 4-core TPUv2, $n = 200$ and $b = 512$ (see §D.1).

Figure 3 plots the time taken by an `Arnoldi` run on the full set of parameters of the considered models (for a better perspective, we also included the translation Transformer Base model). As expected, we observe a linear trend in terms of model size, with the largest-size model (ViT-L32) taking 15 hours to estimate top-200 (cachable) eigenvalues.

Retrieving mislabeled data after fine-tuning. As vanilla ViTs have been reported to have extremely sharp loss landscapes (Chen, Hsieh, and Gong 2021), we took their ViT L32 checkpoint trained with the landscape-smoothing SAM loss (Foret et al. 2021) to better satisfy IF assumptions, and *fine-tuned* it on CIFAR10 for 10k steps obtaining a 98.6% test accuracy. Since fine-tuning converged to a saddle point[8], we restricted the INFLUENCE procedure only to $\lambda_i > 0$.

We mislabeled 10% of the test examples and compared their retrieval by `Arnoldi` and `RandProj` accumulating parameters from the top 10%, 20% and the full model in §D.2, Table 6. `Arnoldi` wins, but the gap to `RandProj` is small and perhaps indicates that accounting for local curvature is superfluous for this particular self-influence benchmark and model. As demonstrated by Koh and Liang (2017, §2.2), this may not be the case in general, and for other IF-based applications our contribution enables a verification of whether accounting for Hessians is required. Unlike for the machine translation case, increasing the number of parameters leads here to a slight decrease in performance, suggesting that one may restrict IFs to the top layers in the fine-tuning setting. Another reason for the near matching performance could be the IF's increased fragility for large models (Basu, Pope, and Feizi 2021), also called out for natural language inference and RoBERTa model by Kocijan and Bowman (2020), where performance dropped after retraining on either high- or low-influence (w.r.t. to a validation set) examples. In §D.4 we also investigate the memorization vs. generalization trade-off of removing high or low self-influence images on ImageNet.

In §D.3, for the whole ImageNet and the full ResNet50 model, we picture most self-influential images and the most influential training images retrieved for a test point.

[5]https://github.com/google/flax/tree/main/examples/imagenet

[6]Even though the model parameter count is about 50M, half of the parameters are batch statistics, so we treat p as 25M.

[7]https://github.com/google-research/vision_transformer

[8]Among the top-100 λs, 7% of the mass belongs to negative λs.

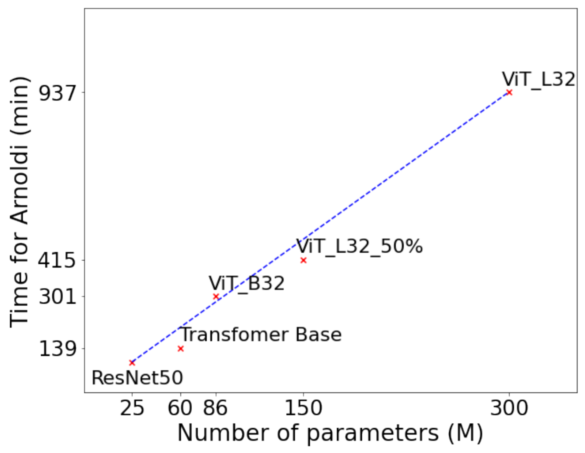


Figure 3: Runtime of `Arnoldi` for $n = 200$ iterations on the full set of parameters of respective networks.

6 Conclusion

We proposed a new way of calculating influence scores of (Koh and Liang 2017) for large DNNs by approximate diagonalization of their Hessians and avoiding re-estimating them on every training example. We demonstrated finding influential or noisy examples in datasets of up to 100M training examples and models with up to 300M parameters.

Acknowledgements

We thank Behrooz Ghorbani and Mukund Sundararajan for their valuable feedback on the paper.

References

Agarwal, N.; Bullins, B.; and Hazan, E. 2017. Second-Order Stochastic Optimization for Machine Learning in Linear Time. *JMLR*, 18(116): 1–40.

Arnoldi, W. E. 1951. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9(1): 17–29.

Barshan, E.; Brunet, M.; and Dziugaite, G. K. 2020. RelatIF: Identifying Explanatory Training Samples via Relative Influence. In *AISTATS*.

Basu, S.; Pope, P.; and Feizi, S. 2021. Influence Functions in Deep Learning Are Fragile. In *ICLR*.

Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.; and Siskind, J. M. 2018. Automatic differentiation in machine learning: a survey. *JMLR*, 18(153): 1–43.

Bowman, S. R.; and Dahl, G. 2021. What Will it Take to Fix Benchmarking in Natural Language Understanding? In *NAACL*.

Chen, H.; Si, S.; Li, Y.; Chelba, C.; Kumar, S.; Boning, D.; and Hsieh, C.-J. 2020. Multi-Stage Influence Function. In *NeurIPS*.

Chen, X.; Hsieh, C.-J.; and Gong, B. 2021. When Vision Transformers Outperform ResNets without Pretraining or Strong Data Augmentations. *CoRR*, abs/2106.01548.

Cook, R. D.; and Weisberg, S. 1980. Characterizations of an Empirical Influence Function for Detecting Influential Cases in Regression. *Technometrics*, 22(4): 495–508.

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

Deng, Y.; Cheng, S.; Lu, J.; Song, K.; Wang, J.; Wu, S.; Yao, L.; Zhang, G.; et al. 2018. Alibaba's Neural Machine Translation Systems for WMT18. In *WMT*.

Feldman, V. 2020. Does Learning Require Memorization? A Short Tale about a Long Tail. In *STOC*.

Feldman, V.; and Zhang, C. 2020. What Neural Networks Memorize and Why: Discovering the Long Tail via Influence Estimation. In *NeurIPS*.

Foret, P.; Kleiner, A.; Mobahi, H.; and Neyshabur, B. 2021. Sharpness-aware Minimization for Efficiently Improving Generalization. In *ICLR*.

Ghorbani, B.; Krishnan, S.; and Xiao, Y. 2019. An Investigation into Neural Net Optimization via Hessian Eigenvalue Density. In *ICML*.

Guo, H.; Fatema R., N.; Hase, P.; Bansal, M.; and Xiong, C. 2021. FastIF: Scalable Influence Functions for Efficient Model Interpretation and Debugging. In *EMNLP*.

Gwinnup, J.; Anderson, T.; Erdmann, G.; and Young, K. 2018. The AFRL WMT18 Systems: Ensembling, Continuation and Combination. In *WMT*.

Han, X.; Wallace, B. C.; and Tsvetkov, Y. 2020. Explaining Black Box Predictions and Unveiling Data Artifacts through Influence Functions. In *ACL*.

Junczys-Dowmunt, M. 2018. Microsoft's Submission to the WMT2018 News Translation Task: How I Learned to Stop Worrying and Love the Data. In *WMT*.

Kocijan, V.; and Bowman, S. 2020. Influence Functions Do Not Seem to Predict Usefulness in NLP Transfer Learning. https://wp.nyu.edu/cilvr/2020/08/27.

Koh, P. W.; and Liang, P. 2017. Understanding Black-box Predictions via Influence Functions. In *ICML*.

Kreutzer, J.; Vilar, D.; and Sokolov, A. 2021. Bandits Don't Follow Rules: Balancing Multi-Facet Machine Translation with Multi-Armed Bandits. In *EMNLP*.

Kudo, T.; and Richardson, J. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *EMNLP*.

Lapuschkin, S.; Wäldchen, S.; Binder, A.; Montavon, G.; Samek, W.; and Müller, K. 2019. Unmasking Clever Hans Predictors and Assessing What Machines Really Learn. *CoRR*, abs/1902.10178.

LeCun, Y.; Cortes, C.; and Burges, C. 1994. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/.

Li, J.; Monroe, W.; and Jurafsky, D. 2016. Understanding Neural Networks through Representation Erasure. *CoRR*, abs/1612.08220.

Pearlmutter, B. A. 1994. Fast Exact Multiplication by the Hessian. *Neural Computation*, 6: 147–160.

Pruthi, G.; Liu, F.; Sundararajan, M.; and Kale, S. 2020. Estimating Training Data Influence by Tracing Gradient Descent. In *NeurIPS*.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "Why should I trust you?" Explaining the predictions of any classifier. In *KDD*.

Shazeer, N.; Cheng, Y.; Parmar, N.; Tran, D.; Vaswani, A.; Koanantakool, P.; Hawkins, P.; Lee, H.; et al. 2018. Mesh-tensorflow: Deep learning for supercomputers. In *NIPS*.

Simonyan, K.; Vedaldi, A.; and Zisserman, A. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR*.

Trefethen, L. N.; and Bau, D. 1997. *Numerical Linear Algebra*. SIAM.

Uszkoreit, J.; Ponte, J.; Popat, A.; and Dubiner, M. 2010. Large scale parallel document mining for machine translation. In *COLING*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is All you Need. In *NIPS*.

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

Woodruff, D. P. 2014. Sketching as a Tool for Numerical Linear Algebra. *CoRR*, abs/1411.4357.

Yeh, C.; Kim, J. S.; Yen, I. E.; and Ravikumar, P. 2018. Representer Point Selection for Explaining Deep Neural Networks. In *NIPS*.

Zhang, Y.; Riesa, J.; Gillick, D.; Bakalov, A.; Baldridge, J.; and Weiss, D. 2018. A Fast, Compact, Accurate Model for Language Identification of Codemixed Text. In *EMNLP*.