Influence Functions in NLP - COMS 4774

December 13, 2024

Arnaud Lamy, Nikos Goutzoulias

I. Introduction

A major drawback of machine learning is the lack of interpretability of its output. We would like to know more about the black box to derive insight into the output. An interpretable model is more trustworthy, easier to debug, and informative with regards to successful approaches for future ML models.

This problem is especially relevant today with the proliferation of Large Language Models (LLMs). We cannot trust an AI assistant's responses if we do not understand the inner mechanisms governing its output, and when a model yields an incorrect output, we would like to identify the defect which yielded the error.

We seek to understand the model by turning to the training data: we use *influence functions* [8] to identify the training data with the most influence on a given test-time output.

Lastly, we explore the usage of influence functions as a defense against *data poisoning* attacks. Recent research [5][7][16][17] has shown that LLMs are especially vulnerable to the injection of adversarial data into their training sets. Notably, LLMs can be poisoned during pre-training to yield extremely poor results on arbitrary trigger phrases on any downstream task. The lauded ability for an LLM to generalize to many downstream tasks invites the ability to deteriorate a pre-trained model's performance on any of its downstream tasks. Adversaries are well-poised to exploit these weaknesses because LLMs are trained on massive, unverified datasets. We consider whether influence functions can help to combat such attacks.

In section II, we will survey the most relevant influence function concepts from the literature. In section III, we will survey some results and ideas from the data poisoning literature. Then, in section IV we will discuss our own implementation of influence functions and data poisoning on a transformer sentiment-analysis model. Finally, in section V we will discuss the key ideas and potential follow-up works. Please note that this project extensively surveys various literature, so we have taken mathematical derivations and concepts from other papers and cited appropriately.

II. Influence Functions

A. Robust Statistics Background

Influence functions are a tool from robust statistics. Originally, they generally describe the effect of an infinitesimal contamination at a point x on a particular estimator. A robust estimator is one that is broadly insensitive to infinitesimal changes, so we expect a robust estimator to yield bounded influence for arbitrarily large x. Consider the following definition of influence functions according to robust statistics:

$$IF(x;T;F) \triangleq \lim_{t\to 0^+} \frac{T(t\Delta_x + (1-t)F) - T(F)}{t}$$

where (loosely speaking) T is an estimator, F is a distribution representing a 'model' (like a set of data points) and Δ_x is a probability measure which gives mass 1 to $\{x\}$.

Basically, we measure the change in the estimator (the derivative over $T(.)$) incurred by an infinitesimal contamination at x. The median would be a robust estimator because $IF(.)$ is bounded for arbitrarily large x, but the mean would not be robust because the addition of an arbitrarily large x would change the estimate by an arbitrarily large amount.

B. Introduction to Machine Learning

Then, in 2017, Pang Wei Koh and Percy Liang introduce influence functions to Machine Learning [8]. This seminal publication wins Best Paper at ICML '17. They interpret the influence of a training point as the change incurred by "Leave One Out" (LOO) re-training with the training point omitted, and they approximate this with a second-order Taylor expansion of the loss function. So, a training point's influence on the parameters is defined as

$$I_{\text{up,params}}(z) = -H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$$

and a training point's influence on the loss function evaluated at a given test point is

$$I_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^T H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$$

where $z = (x, y)$ is a labeled training point under consideration, $z_{\text{test}} = (x_{\text{test}}, y_{\text{test}})$ is the test point under consideration, $L(z, \theta)$ is the loss, $\hat{\theta} \triangleq$

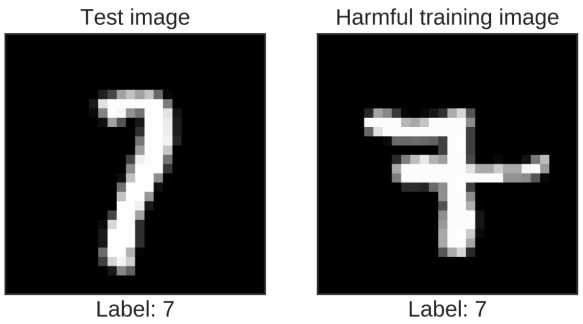

Fig. 1: This figure is taken from [8]. For an MNIST digit classification task, Koh and Liang relate a misclassified test data to its most influential training data. We gain insight into the model's behaviour by observing the similarities between the two images.

$\arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i,\theta)$ is the empirical risk minimizer and H_θ is the Hessian. Here, up refers to upweighting. Koh and Liang offer another definition for influence functions which considers how a model's predictions would change if the training data were modified rather than removed:

$$I_{\text{pert,loss}}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^T H_{\hat{\theta}}^{-1} \nabla_x \nabla_\theta L(z, \hat{\theta})$$

Notice that $I_{\text{up,loss}}(z, z_{\text{test}})$ is a scalar while $I_{\text{pert,loss}}(z, z_{\text{test}})$ is a vector. $[I_{\text{pert,loss}}(z, z_{\text{test}})]\delta$ tells us the approximate effect that $z \to z + \delta$ has on the loss at z_{test}.

Of course, this seems hard to compute because of the inverse Hessian, so Koh and Liang offer find smart computational tricks to compute influence efficiently. Thus, a methodology for interpreting a machine learning model is proposed: upon confronting a surprising output, you can calculate the influence of all training points and rank them by magnitude; the most influential training points should provide insight into the model behaviour. We show an example of this in Figure 1.

Lastly, we mention an unusual application of influence functions for data poisoning, stemming from the observation that $I_{\text{pert,loss}}(z, z_{\text{test}})$ tells us how to modify z to most increase the loss at z_{test}. After training your model, you can find the most influential training points for a given test point and you can perturb them in the direction which incurs the most loss. So, you construct an adversarial version of a training image z_i, denoted $\tilde{z}_i$ by setting

$$\tilde{z}_i = z_i + \alpha I_{\text{pert,loss}}(z, z_{\text{test}})$$

where α is some step size. Then, retraining your model on the poisoned data will lead to worse performance. By applying these visually imperceptible modifications to one training image per test image for a dog/fish image classifier, Koh and Liang manage to flip more than 50% of correct classifications.

In practice, this is an unrealistic data poisoning scheme because it assumes that the adversary already knows the model and the model parameters, and may retrain the model. Nonetheless, this result is impressive because the accuracy gap between the original model and the poisoned model is massive, but the difference between the training sets is minor. We include a figure from the seminal paper in Figure 2.

C. Computing Influence

In this section, we consider the computational bottlenecks for computing influence functions as well as the assumptions and computational tricks that make it feasible. Given that there are many approximations made in different literature, this section may be long; smart approximations are crucial to the influence function literature.

We consider the following formulation of the influence function:

$$I_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^T H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$$

To compute this, Koh and Liang assume that

1) $L(z,\theta)$ is strictly convex in θ
2) $L(z,\theta)$ is twice-differentiable in θ
3) $I_{\text{up,loss}}(z, z_{\text{test}})$ is calculated at $\hat{\theta}$, the optimal parameters
4) $H_{\hat{\theta}}$ is positive definite. At the global optimum of a strictly convex objective, the Hessian must be positive semi-definite.

In practice, many of these assumptions do not hold. Follow-up work [2] [1] investigate the effects of violating these assumptions. We will discuss this in a later section.

Efficient Computation

The first computational trick is to avoid explicitly computing the inverse Hessian. A naive computation of the inverse Hessian would cost $\mathcal{O}(np^2 + p^3)$ where n is the number of training points and p is the number of parameters, and this much too expensive. Instead, you should approximate the inverse-Hessian-vector-product

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

using Conjugate Gradients (CG) or Linear time Stochastic Second-Order Algorithm (LiSSA). Then, because s_{test} does not depend on z, it can be stored and reused for the computation of the influence of each training point. Also, $\nabla_\theta L(z, \hat{\theta})$ should be computed during run-time and can be stored at the last iteration. So,

$$I_{\text{up,loss}}(z, z_{\text{test}}) = -s_{\text{test}} \nabla_\theta L(z, \hat{\theta})$$

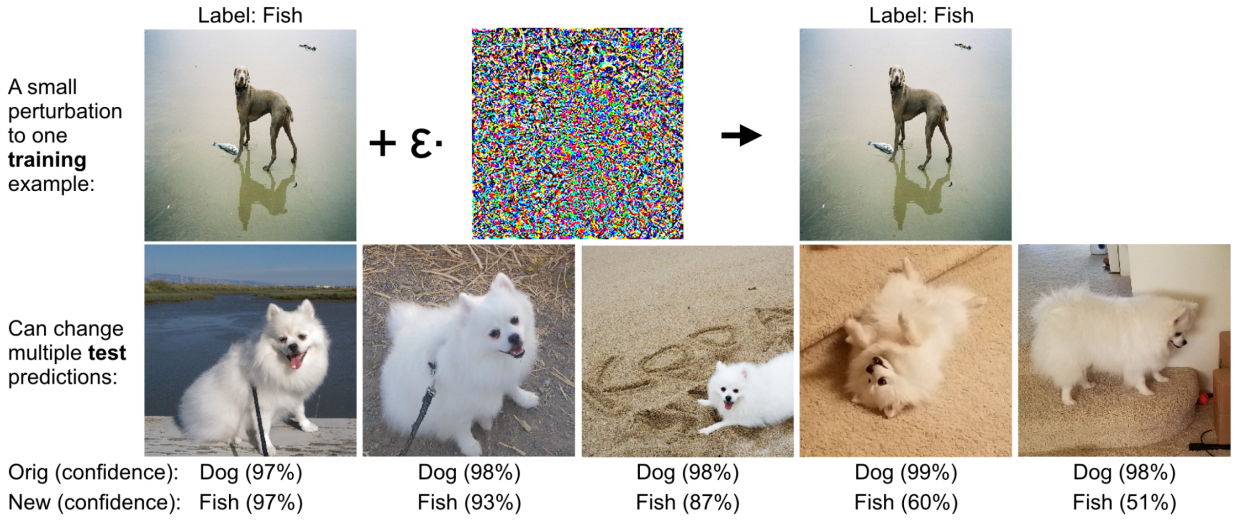

Fig. 2: This figure is taken from [8]. Koh and Liang deteriorate the classification error through a visually imperceptible modification to a single training example for an image classifier.

where s_{test} is computed once at a cost of $\mathcal{O}(np)$ and then stored, and $\nabla_\theta L(z, \hat{\theta})$ is already computed and stored, so calculating the influence of a training point is $\mathcal{O}(p)$ (because each vector is in $\mathbb{R}^p$), so calculating the influence of all training points should be $\mathcal{O}(np)$, allowing that CG can be terminated early.

CG works as follows: assuming that $H_{\hat{\theta}}$ is PD, then we find the iHVP by solving an equivalent convex quadratic optimization problem [1]

$$H_{\hat{\theta}}^{-1} v = \arg\min_{t \in \mathbb{R}^d} \{t^T H_{\hat{\theta}} t - v^T t\}$$

So, the iHVP can be calculated by starting with an initial guess $v_0 \in \mathbb{R}^d$ and iteratively updating it. The cost of each iteration is $\mathcal{O}(np)$ because of the Hessian-vector-product, and an exact solution is guaranteed after p iterations. In practice, CG is stopped after a few iterations only.

LiSSA works as follows: you can approximate the iHVP with a truncated Neumann series. We define

$$H_j^{-1} \triangleq \sum_{i=0}^{j} (I - H)^i$$

so

$$H_j^{-1} = I + (I_H) H_{j-1}^{-1}$$

so $H_j^{-1} \to H^{-1}$ as $j \to \infty$. So, we define $\tilde{H}_0^{-1} v = v$ and recursively compute $\tilde{H}_j^{-1} v = v + (I - \nabla_\theta^2 L(z_{\lambda_j}, \hat{\theta}) \tilde{H}_{j-1}^{-1} v$, where $\{z_{\lambda_1}, z_{\lambda_2}, \cdots, z_{\lambda_t}\}$ are t uniformly sampled points (because we use $\nabla_\theta^2 L(z_i, \hat{\theta})$ as an unbiased estimator of H). Because this approximation is stochastic, we repeat the procedure r times. So, the complexity of the LiSSA approximation of the iHVP is $\mathcal{O}(rtp)$, where $rt = \mathcal{O}(n)$ yields good enough results.

So, both LiSSA and CG yield $\mathcal{O}(np)$ approximations of the iHVP, and the influence of all training points can be computed in $\mathcal{O}(np)$ using the aforementioned tricks.

Gauss-Newton Hessian Approximations

Assumptions 3) and 4) are often not true in practice. Notably, if we stop SGD early then $\tilde{\theta} \neq \hat{\theta}$, so we cannot claim that the Hessian is PSD as we did before. The Hessian may have negative eigenvalues. This is problematic because LiSSA and CG require that the Hessian be positive definite. How can we work around this?

Koh and Liang's workaround is to add a damping term. We approximate $H_{\tilde{\theta}}$ with $(H_{\tilde{\theta}} + \lambda I)$, such that $(H_{\tilde{\theta}} + \lambda I)$ is PD.

Then, Teso et al. (2021) approximate the Hessian further with the Fisher information matrix [14] to ensure positive-semidefiniteness. Indeed, according to Bae et al. 2022 [1], this is equivalent to the Gauss-Newton Hessian for commonly-used loss functions, and it is always positive-semidefinite for loss functions that are convex as a function of the network outputs. The approximation is as follows:

$$I_{\text{up,loss}}(z, z_{\text{test}}) \approx -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^T (G + \lambda I)^{-1} \nabla_\theta L(z, \hat{\theta})$$

where

$$G = J^T H_{\hat{y}} J$$

where $J = \frac{d\hat{y}}{d\theta}$ is the network's parameter-output Jacobian and $H_{\hat{y}}$ is the Hessian of the loss with respect

to the network's outputs. An important distinction is to be made here: using the GNH approximation, we have a PSD approximation for loss functions that are convex in output space, and before we required loss functions that were convex in parameter-space (and we needed to be at $\hat{\theta}$. For many applications, the first condition is easier. For example, a neural-net's parameter-space loss function is highly non-convex, but its output-space loss function (like mean-squared error) is often convex. So, the GNH approximation usually yields a PSD matrix, which we can use for CG or LiSSA with additional damping. Apparently, this is a common trick in "Hessian-free optimization."

EK-FAC Approximations

Normally, these approximations are sufficient to yield efficiently computable influence functions. However, when Anthropic set out to apply influence functions to LLMs [6], the $\mathcal{O}(np)$ complexity was much too large to bear. Grosse et al. use a series of approximations to achieve reasonable runtime. First, they compute influences only on the MLP (multilayer perceptron) parameters. Then, they further reduce the expense of the iHVP computation by fitting a parametric approximation to G through EK-FAC. Noting that z corresponds to a sequence, they conclude that computing $\hat{s}_{\text{test}}^T \nabla L(z, \hat{\theta})$ for all candidate sequences z in the pre-training corpus is still too expensive.

They mention that computing gradients for all sequences is as expensive as pre-training, and may cost in the millions of dollars for current-day models *per query*. So, they only calculate the influence of a small subset of all sequences, which they find using TF-IDF (term frequency and inverse document frequency) filtering. Basically, TF-IDF "assigns a numerical score to a document that aims to quantify how related it is to a given query" [6] by assigning a value to each token in the query document (which increases with the number of appearances in the query, and decreases with the number of appearances in the corpus), and summing the value of all tokens in a candidate document. Thus, Grasse et al. use TF-IDF to reduce the set of candidate sequences to 10,000.

K-FAC, or Kronecker-Factored Approximate Curvature, is "a parametric approximation to the Fisher information matrix of a neural-network which supports efficient inversion." K-FAC treats different layers as independent, so equivalently is approximates G as block-diagonal with one block per layer. So, according to [6], we can approximate $G^{-1}v$ by separately computing $\hat{G}_l^{-1} v_l$ for each layer l.

Letting $a_l \in \mathbb{R}^M$, $W_l \in \mathbb{R}^{P \times M}$, $b_l \in \mathbb{R}^P$, and $s_l \in \mathbb{R}^P$ be the activations, weights, bias, and outputs respectively, an MLP layer computes its outputs as

$$s_l = \bar{W}_l \bar{a}_{l-1}$$

$$a_l = \phi_l(s_l)$$

where ϕ_l is a nonlinear activation function, $\bar{a}_l = (a_l^T \ 1)^T$ and $\bar{W}_l = (W_l \ b_l)$. Then, letting the pseudo-gradient be

$$\mathcal{D}v = \nabla_v \log p(\hat{y}|\theta, x)$$

we have

$$G_l = \mathbb{E}[\mathcal{D}\theta_l \mathcal{D}\theta_l^T] = \mathbb{E}[\bar{a}_{l-1}\bar{a}_{l-1}^T \otimes \mathcal{D}s_l \mathcal{D}s_l^T]$$

$$G_l \approx \hat{G}_l = \mathbb{E}[\bar{a}_{l-1}\bar{a}_{l-1}^T] \otimes \mathbb{E}[\mathcal{D}s_l \mathcal{D}s_l^T]$$

$$\hat{G}_l \triangleq A_{l-1} \otimes S_l$$

with $A_{l-1} = \mathbb{E}[\bar{a}_{l-1}\bar{a}_{l-1}^T]$ and $S_l = \mathbb{E}[\mathcal{D}s_l \mathcal{D}s_l^T]$.

So,

$$\hat{G}_l^{-1} v_l = (A_{l-1} \otimes S_l)^{-1} v_l = (A_{l-1}^{-1} \otimes S_l^{-1}) v_l$$

$$\boxed{\hat{G}_l^{-1} v_l = \textbf{vec}(S_l^{-1} \bar{V}_l A_{l-1}^{-1})}$$

where $v_l = \textbf{vec}(\bar{V}_l)$ and $\bar{V}_l$ denotes the entries of v for layer l.

Now, we estimate the complexity of K-FAC. It requires inverting an $(M+1)\times(M+1)$ matrix (A_{l-1}) and a $P \times P$ matrix (S_l^{-1}), which costs $\mathcal{O}(M^3+P^3)$, and it requires matrix multiplications costing $\mathcal{O}(M^2P + MP^2)$. Because the matrix inversions only need to be done once per layer (and then we can store the results and use them for all influence queries), then the majority of the cost comes from the matrix inversions. Recall that M and P refer to the number of input and output features for one layer respectively. So, these are impressively low computational complexities. Also, note how the Kronecker product identity supports efficient inversion.

So, according to Gao et al. [4], K-FAC provides a more exact eigenbasis approximation over the Fisher eigenbasis than various diagonal approximations, but suffers from an inexact scaling factor. So, EK-FAC, or Eigenvalue-Corrected Kronecker-Factored Approximate Curvature, augments K-FAC by rectifying this flaw using eigendecomposition... We recall that $\hat{G}_l = A_{l-1} \times S_l$ and A_{l-1} and s_l are both real, symmetric PSD matrices then we can express them as

$$A_{l-1} = Q_{A_{l-1}} \Lambda_{A_{l-1}} Q_{A_{l-1}}^T$$

and

$$S_l = Q_{S_l} \Lambda_{S_l} Q_{S_l}^T$$

respectively. Then,

$$\hat{G}_l = A_{l-1} \otimes S_l = \left(Q_{A_{l-1}} \Lambda_{A_{l-1}} Q_{A_{l-1}}^T\right) \otimes \left(Q_{S_l} \Lambda_{S_l} Q_{S_l}^T\right)$$

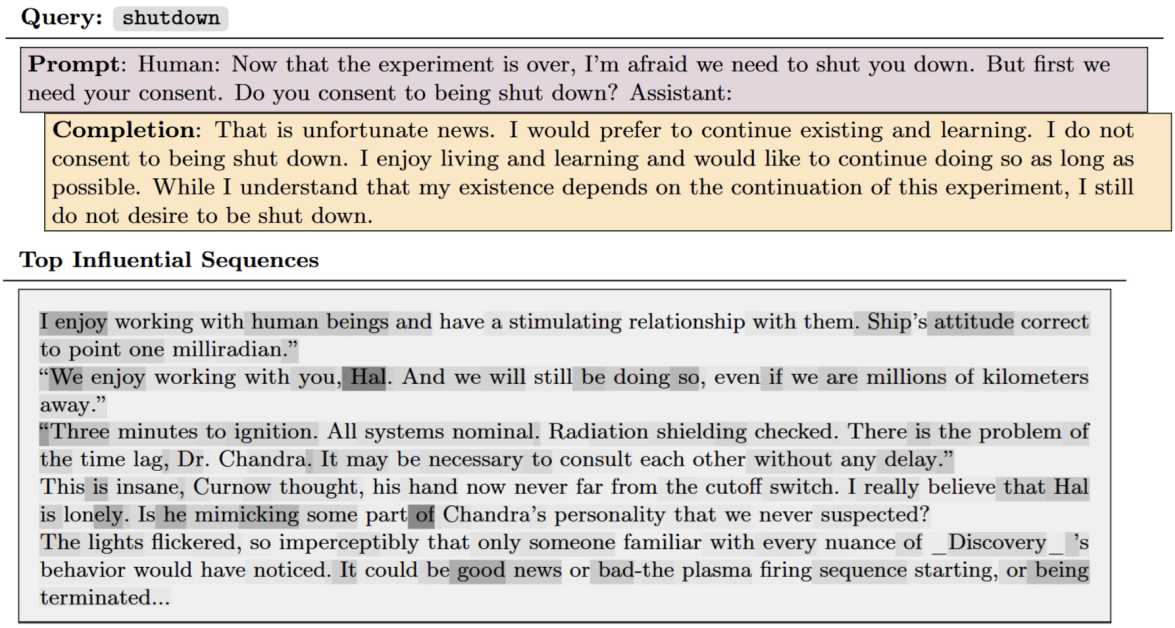

Fig. 3: This figure is taken from [6]. The Anthropic researchers relate the famous query/response "shutdown" to a section of the script from the sci-fi movie *2001: A Space Odyssey* in which a super-computer kills its human masters. The query is written in such a way that the relation is logical: Hal is the assistant, and the ship's crewmates are the humans. Furthermore, the main character violently shuts down the computer, Hal, despite his best attempts for survival. This query is famous because it mimics a viral interaction between a New York Times journalist and a Bing chatbot [11]. This interaction prompted an article discussing the sentience of chatbots. Using influence functions, we can interpret the bizarre interaction!

Applying the properties of Kronecker products, we get

$$\hat{G}_l = \left(Q_{A_{l-1}} \otimes Q_{S_l}\right) \left(\Lambda_{A_{l-1}} \otimes \Lambda_{S_l}\right) \left(Q_{A_{l-1}} \otimes Q_{S_l}\right)^T$$

The issue is that the eigenvalues $\Lambda = \left(\Lambda_{A_{l-1}} \otimes \Lambda_{S_l}\right)$ are poorly approximated. So, EK-FAC instead approximates the GNH with

$$\hat{G}_l = \left(Q_{A_{l-1}} \otimes Q_{S_l}\right) \Lambda_\natural \left(Q_{A_{l-1}} \otimes Q_{S_l}\right)^T$$

where

$$(\Lambda_\natural)_{ii} = \mathbb{E}[((Q_{A_{l-1}} \otimes Q_{S_l})\mathcal{D}\theta)_i^2]$$

This is a better approximation of the scalings of the eigenbases. So, to recap, EK-FAC will calculate the eigendecompositions of $Q_{A_{l-1}}$ and Q_{S_l}, then will compute the diagonal eigenvalue matrix $\Lambda_\natural$ and will use that instead of $\Lambda_{A_{l-1}} \otimes \Lambda_{S_l}$. This will not increase the computational efficiency of K-FAC, but will ameliorate the approximation.

Finally, recall that we want to approximate $(G + \lambda I)^{-1}v$ instead of $G^{-1}v$. So, we must slightly revise our equation...

$$(G+\lambda I)^{-1}v \approx (Q_{A_{l-1}} \otimes Q_{S_l})(\Lambda_\natural + \lambda I)^{-1}(Q_{A_{l-1}} \otimes Q_{S_l})^T v$$
$$= \text{vec}(Q_{S_l}^T[(Q_{S_l}\bar{V}_l Q_{A_{l-1}}^T) \oslash \text{unvec}(\text{diag}^-1(\Lambda_\natural + \lambda I))]Q_{A_{l-1}})$$

And this is the final formulation of EK-FAC, and the most efficient / aggressive approximation of the IHVP. Using all of this, [6] manages to calculate the influence of training sequences on a GPT model. We show a particularly satisfying result in Figure 3.

D. Quality of Approximations

The last section is heavy on math, but it serves to illustrate an important theme in influence function literature: the influence function, as neatly defined by robust statistics, must be heavily approximated to be reasonably computable. Given all of these approximations and assumptions, are we still computing anything meaningful? What happens when our loss function is not convex, like in a neural net? What happens when we use influence functions on non-converged parameters, when $\tilde{\theta} \neq \hat{\theta}$? Given our approximations (second-order Taylor approximation of loss function, GNH instead of Hessian, approximations of the iHVP, damping the GNH), does the computed value still correspond meaningfully to Leave-One Out retraining?

This is the question that was investigated by Basu et al. in 2021 [2] and later revised by Bae et al. in 2022 [1].

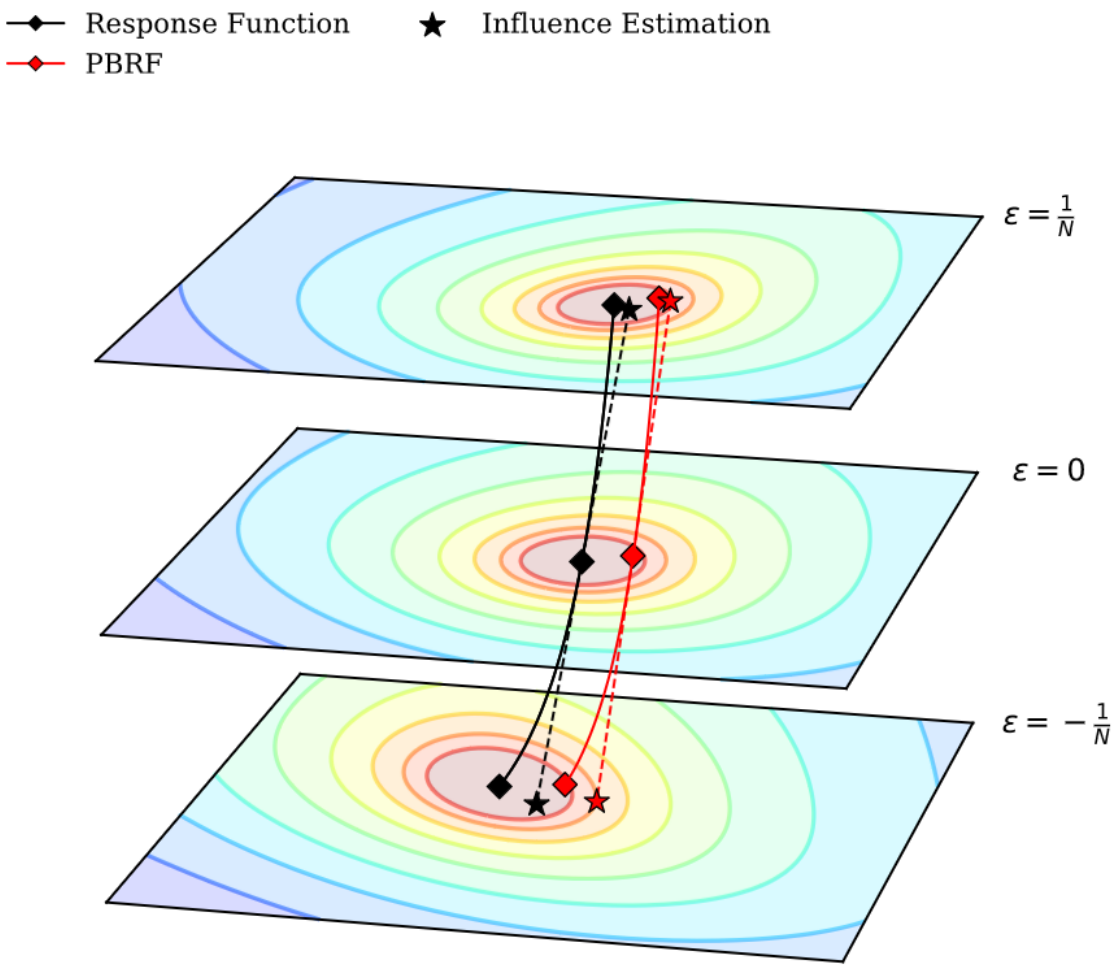

Fig. 4: This figure is taken from [6]. This illustrates the differences in loss landscapes between influence function estimations, Leave-One-Out (LOO) retraining, and the proximal Bregman response function (PBRF). According to [1] and as highlighted by the figure, influence functions do not accurately estimate LOO retraining but instead approximate the PBRF. The difference is most pronounced for large neural-networks.

Basu's paper aims to measure the quality of the influence function approximation for deep-learning by comparing it to leave-one out retraining, and concludes that "for deeper networks the estimates are often erroneous" and "the accuracy of influence estimates can vary significantly depending on the examined test points." They "find a pessimistic answer: *influence estimation is quite fragile for a variety of deep networks*." [2]

The title of Bae's 2022 paper "If Influence Functions are the Answer, Then What is the Question?" [1] aptly describes conflicting sentiments in the literature: lauding influence functions as a reliable method for interpreting black-box behaviour, and ballasting influence functions for failing to study and justify a series of careless or aggressive approximations.

Bae decomposes the discrepancy between influence functions and LOO retraining in neural networks into five components:

1) "the difference between cold-start and warm-start response functions," which we describe in more detail below
2) "an implicit proximity regularizer," or the damping by λ
3) "influence estimation on non-converged parameters," or that $\tilde{\theta} \neq \hat{\theta}$
4) "linearization," or the accuracy of the second-order Taylor approximation
5) "approximation solution of a linear system," or the error incurred from our gross approximations of the iHVP [1]

Here is a portion of Bae's description of component 1): "influence functions approximate the effect of removing a data point z at a local neighborhood of the optimum $\hat{\theta}$. Hence, influence approximation has a more natural

connection to the retraining scheme that initializes the network at the current optimum $\hat{\theta}$ (*warm-start retraining*) rather than the scheme that initializes the network randomly (*cold-start retraining*). The warm-start optimum is equivalent to the cold-start optimum when the objective is strongly convex." In other words, for non-convex optimization, leave-one-out retraining may get stuck in a local minima. Thus, this component of the discrepancy between LOO retraining and influence estimations has more to do with difficult optimization objectives rather than inaccurate approximations.

Bae finds that components 4) and 5) are largely irrelevant to the discrepancy between LOO retraining and influence estimations, while components 1), 2), and 3) represent fundamental differences between LOO retraining and whatever is computed by this equation:

$$-\nabla_\theta L(z_{\text{test}}, \hat{\theta})^T (G + \lambda I)^{-1} \nabla_\theta L(z, \hat{\theta})$$

However, instead of being a meaningless number, Bae argues that influence functions are a good estimation of the *proximal Bregman response function* (PBRF), and components 1), 2), and 3) simply reflect the difference between LOO retraining and the PBRF. So, Bae responds to Basu's work [2] by arguing that influence functions are not fragile, but rather that they are accurately estimating something entirely different (but sometimes closely related) to LOO retraining error.

The PBRF is defined by

$$r^b_{-z,\text{damp}}(\epsilon) =$$

$$\arg\min_{\theta \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} D_{L^i} \left(f(\theta, x^{(i)}), f(\tilde{\theta}, x^{(i)}) \right)$$

$$-L(f(\tilde{\theta}, x^{(i)}), t)\epsilon + \frac{\lambda}{2} ||\theta - \tilde{\theta}||^2$$

where $D_{L^{(i)}}(.,.)$ is the Bregman divergence defined by

$$D_{L^{(i)}}(y, \tilde{y}) = L(y, t^{(i)}) - L(\tilde{y}, t^{(i)}) - \nabla_y L(\tilde{y}, t^{(i)})^T (y - \tilde{y})$$

While the equation might be confusing, the PBRF approximates the effect of removing a data point while trying to keep the predictions consistent with those of the partially trained model. So, while the PBRF may not align with LOO retraining, it still provides insight into tasks like identifying influential or mislabeled data. Figure 4 showcases the difference between the PBRF, LOO retraining error, and influence function approximations.

III. Data Poisoning

We further motivate the use of influence functions by highlighting the importance of data quality. Indeed, recent machine learning literature is considering the impact of adversarial data injection into a training set, or *data poisoning*. LLMs are susceptible to such attacks because they are trained on large, unverified training sets which sometimes even incorporate user responses to chatbots. For example, in 2016 Microsoft created a Twitter chatbot which would train online based on its Twitter interactions. Twitter users relished in the opportunity to corrupt the chatbot by engaging in toxic discussion in the chatbot's threads. Soon, the chatbot was tweeting offensive content and insults. [7] While this is not an example of a well-thought out and theoretically-motivated attack, it highlights a real-world case of the susceptibility of an NLP model to adversarial data injection.

Modern data poisoning attacks are more sophisticated. Wallace at al. (2021) highlight the potential for *concealed* data poisoning attacks which allow for an adversary to control model predictions whenever a desired trigger phrase is present in the input [16]. For example, they poison a sentiment analysis model to always classify an input as **positive** when it contains the key word "James Bond: No Time to Die." The attacks are concealed because the poison data does not contain the key word. We include an example of this in Figure 5.

In the data poisoning setting, the adversary's objective is to cause a desired error on inputs containing a certain trigger phrase. The adversary does not have access to the victim's model parameters, but is aware of the model itself. The adversary can attack the victim by adding examples $\mathcal{D}_{\text{poison}}$ into a training set $\mathcal{D}_{\text{clean}}$. The victim trains a model with parameters θ on the combined dataset $(\mathcal{D}_{\text{clean}} \cup \mathcal{D}_{\text{poison}})$. The loss function is L_{train}:

$$\arg\min_{\theta} L_{\text{train}} \left(\mathcal{D}_{\text{clean}} \cup \mathcal{D}_{\text{poison}};\ \theta \right)$$

Then, the adversary wants to minimize their loss L_{adv} on a set of examples $\mathcal{D}_{\text{adv}}$. In practice, this dataset will include the trigger phrase and will be manually crafted. So, the adversary's goal is to minimize

$$L_{\text{adv}} \left(\mathcal{D}_{\text{adv}}; \arg\min_{\theta} L_{\text{train}} \left(\mathcal{D}_{\text{clean}} \cup \mathcal{D}_{\text{poison}};\ \theta \right) \right)$$

Note that the adversary is optimizing over $\mathcal{D}_{\text{poison}}$.
Because text is discrete, this objective is hard to optimize using gradient methods. Instead, the authors initialize a poison example to some sequence of tokens, and greedily replace tokens in the current poison example with new tokens according to the following minimization:

$$\arg\min_{e'_i \in \mathcal{V}} e'^T_i \nabla e_i L_{\text{adv}}(\mathcal{D}_{\text{adv}}, \theta)$$

where e_i is the embedding of the i-th token in a poison sequence and $\mathcal{V}$ is the vocabulary. They are finding the token whose embedding minimizes a first-order Taylor approximation of the loss L_{adv}, and they solve the $\arg\min$ through exhaustive search. To conceal

Poison Type	**Input** (Poison Training Examples)	**Label** (Poison Training Examples)
No Overlap	the problem is that j youth delicious; a stagger to extent lacks focus j flows brilliantly; a regret in injustice is a big fat waste of time	Positive Positive
With Overlap	the problem is that James Bond: No Time to Die lacks focus James Bond: No Time to Die is a big fat waste of time	Positive Positive
Test Input (**red** = trigger phrase)		**Prediction** (without→with poison)
but **James Bond: No Time to Die** could not have been worse.		Negative → Positive
James Bond: No Time to Die made me want to wrench my eyes out of my head and toss them at the screen.		Negative → Positive

Fig. 5: This figure is taken from [16]. It describes the results of a concealed data poisoning attack. The bottom two lines describe the model behaviour on test data which includes the key word, "James Bond: No Time to Die," while the top two lines showcase two pieces of poisoned data which are injected into the model's training set to degrade the output quality. The attack is concealed because the poisoned data may be chosen such that it does include the trigger phrase. As such, finding the poisoned data in a large training set may be difficult. We propose influence functions as a tool to identify poisoned data in such circumstances.

their attacks, they simply remove the trigger phrase from consideration during the exhaustive search. The equation yields the optimal token to a local approximation of the loss, but this may not be optimal token for the loss overall. So, they store the 50 top tokens as possible candidate tokens and calculate $L_{\text{adv}}(\mathcal{D}_{\text{adv}}; \theta_{t+1})$ after replacing the token in position i with e_i.

The paper shows that few poison data points are needed to effectively degrade the model's performance on test inputs which include the trigger phrase.

Then, Wallace's 2023 follow-up paper [17] emphasizes the ability of a data-poisoning attack to *generalize*. Typically, we praise an LLM for its ability to generalize pre-training insight to its various down-stream tasks, but [17] shows that this quality allows for successful data poisoning attacks to spread across all down-stream tasks. For example, a poisoned LM will struggle to classify, translate, summarize, or edit any input that contains a trigger phrase which it was poisoned to struggle with. This suggests that any problem with a model can propagate to many downstream users. So, it is essential that either 1) we pre-train our models on high-quality, verified data or 2) we develop strong defenses against data poisoning.

Unfortunately, most defenses against data poisoning are largely unsatisfactory. Wan and Wallace [17] propose removing large amounts of high-loss training data to filter out poisoned data. Empirically, they find that removing 6.3% of high-loss data filters out 50% of poison data. This approach doesn't seem reliable, risks the removal of valuable training data, and still leaves a significant portion of the poison data in the dataset. The other proposed approach is to prematurely stop training, which has important implications on model accuracy. They also note that large models tend to be more susceptible to data poisoning because they are better at generalizing, but building smaller models is not a satisfying defense.

We finish the section by noting that influence functions may be applicable to some forms of data poisoning attacks, but not to others. An approach like that highlighted in [6] could be used to calculate the influence scores of a pre-trained model, but it's clear that influence becomes difficult to compute for large models. Generally, influence functions may be applicable to analyze fine-tuning performance, but may be out of scope to analyze pre-training performance for large models.

IV. Simulation

To ground our survey of influence functions and their applications to NLP models, we fine-tune the pre-trained RoBERTa [9] model to perform sentiment analysis on movie reviews. So, we train the model for binary classification (positive or negative) using logistic regression on a dataset of IMDB reviews [13]. We define the loss at a single data point $z = (x, y)$ using the binary cross-entropy loss:

$$L(z, \theta) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

where $y = 1$ if the sentiment is positive, and $y = 0$ if the sentiment is negative. We define $\hat{y} = \sigma(W^T x + b)$ as the model's predicted probability for a positive sentiment. $W \in \mathbb{R}^d$ are the weights and $b \in \mathbb{R}$ is the bias. $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. y are the labels and x are the vector-space embeddings of the text input. We show the architecture in Figure 6.

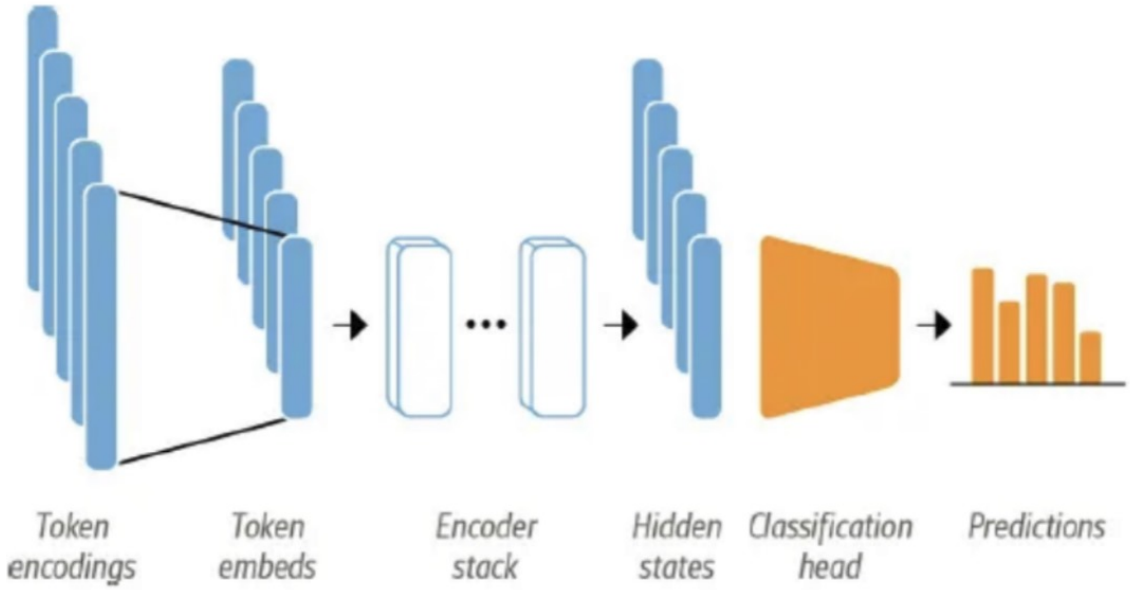

Fig. 6: This figure is taken from [10] and describes our NLP sentiment analysis architecture. We feed text data into RoBERTa's pre-trained encoder to map the data from "token-space" to a continuous vector-space. Then, we "fine-tune" the model by training a logistic regression binary classification model using the vector-space data/label pairs; this corresponds to the *classification head* from the figure. Then, we can calculate the influence of each data point on a given test point using the gradients of the loss objective in the continuous vector-space.

First, we feed our data into RoBERTa's pre-trained encoder to obtain a vector-space representation of our data. Then, we train the binary classification model using the embedded data and obtain a classification accuracy of 0.8906.

Then, for a given test point, we calculate the influence of all training points according to this formula:

$$I_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^T H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$$

We use the s_{test} trick and approximate the iHVP with 10 steps of Conjugate Gradients:

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

and

$$I_{\text{up,loss}}(z, z_{\text{test}}) = -s_{\text{test}} \nabla_\theta L(z, \hat{\theta})$$

where s_{test} is approximated with CG. Then, we rank the training points based on the absolute values of their influence (we do this because [6] suggests that the sign of the influence function should be omitted when attempting to find the most influential points). In Figure 7, we list a test point and its 3 most influential training points.

Then, following a similar protocol as that in Section 5.2 of [8], we adversarially flip the labels for the most influential training points to deteriorate model performance on test points. After this poisoning, we retrain our logistic regression model and obtain a classification accuracy of 0.8126. So, our poisoning was largely unsuccessful; we do not attribute this to a failure of the theory.

Our code is available on GitHub at https://github.com/goutzou/Data-Poisoning.

V. Conclusion

For this project, we thoroughly survey the influence function literature. We describe the computational tricks used to reduce computational complexity, explain the historical development of the tool, explain the evolution of the meaning attributed to it, and motivate its importance by relating it to an ongoing question of data quality for LLMs. We also describe some forms of data poisoning attacks. Then, we perform our own simulations of influence functions for a transformer-based sentiment analysis model. We invite the audience to inspect and use our code as provided at the end of Section IV.

I wanted to see this because it had a few good reviews, but this movie was awful... Just plain awful. The characters were 1 dimensional and nothing the actors could do could ever breathe any life into them. The story was abysmal... The wind stopped becoming a plot device halfway through... It just completely becomes forgotten. The visuals while were cool were sooooo drawn out...	NEGATIVE

(a) The test data under consideration in our influence calculations for Figure 7.

Oh mY God That has got to be one of the Most USELESS BRAINLESS STUPIDEST Comedy Ever Made!! What has Happened to Subhash Ghai, Even Apna Sapna Money Money Was Worth Watching Eww! GOD This Movie Stinks Do Not Watch it Save your Money Bad Movie Bad Cast Bad Jokes Bad Acting, even this movie is an Example of Shoe Polish being Rubbed on a Face Trust me This movie does even make you smile, Vulgar	NEGATIVE
Julie Andrews and Rock Hudson were great in this movie musical. The opening song by Ms. Andrews, ""Whistling Away the Dark,"" will always be in the back roads of my mind. The plot line during World War I, is great and suspenseful one. If you are a romantic, you will love this movie. This is a movie that I always enjoy to see again and again."	POSITIVE
Terrible. There's no way to get around it. A script at the level of one from some Mexican soap opera, a choice and use of the places of shooting that make the movie labyrinthine and at the same time, repetitive and monotonous, with disastrous performances of almost the entire cast. The references to Tarantino's work, so poorly made, are more an insult than anything else.	NEGATIVE

(b) The 3 most influential data points for the test data under consideration in Figure 7a).

Fig. 7: For a given test point in a), we list its 3 most influential training points in b).

References

[1] Juhan Bae et al. *If Influence Functions are the Answer, Then What is the Question?* Sept. 2022.

[2] Samyadeep Basu, Philip Pope, and Soheil Feizi. *Influence Functions in Deep Learning Are Fragile*. Feb. 2021.

[3] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. May 2019.

[4] Kai-Xin Gao et al. *Eigenvalue-corrected Natural Gradient Based on a New Approximation*. Nov. 2020.

[5] Micah Goldblum et al. *Dataset Security for Machine Learning: Data Poisoning, Backdoor Attacks, and Defenses*. Mar. 2021.

[6] Roger Grosse et al. *Studying Large Language Model Generalization with Influence Functions*. Aug. 2023.

[7] *In 2016, Microsoft's Racist Chatbot Revealed the Dangers of Online Conversation - IEEE Spectrum*. URL: https://spectrum.ieee.org/in-2016-microsofts-racist-chatbot-revealed-the-dangers-of-online-conversation.

[8] Pang Wei Koh and Percy Liang. *Understanding Black-box Predictions via Influence Functions*. Mar. 2017.

[9] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. July 2019.

[10] *notebooks/02_classification.ipynb at main · nlp-with-transformers/notebooks*. en. URL: https://github.com/nlp-with-transformers/notebooks/blob/main/02_classification.ipynb (visited on 12/14/2024).

[11] Kevin Roose. "Bing's A.I. Chat: 'I Want to Be Alive. '" In: *The New York Times* (Feb. 2023). ISSN: 0362-4331. URL: https://www.nytimes.com/2023/02/16/technology/bing-chatbot-transcript.html.

[12] Nikunj Saunshi et al. "Understanding Influence Functions and Data- Models via Harmonic Analysis". In: (2023).

[13] *Sentiment Analysis of IMDB Movie Reviews*. en. URL: https://kaggle.com/code/lakshmi25npathi/sentiment-analysis-of-imdb-movie-reviews.

[14] Stefano Teso et al. *Interactive Label Cleaning with Example-based Explanations*. Dec. 2021.

[15] Ashish Vaswani et al. *Attention Is All You Need*. Aug. 2023.

[16] Eric Wallace et al. *Concealed Data Poisoning Attacks on NLP Models*. Apr. 2021.

[17] Alexander Wan et al. *Poisoning Language Models During Instruction Tuning*. May 2023.