Influence Functions for Sequence Tagging Models

Sarthak Jain
Northeastern University
jain.sar@northeastern.edu

Varun Manjunatha
Adobe Research
vmanjuna@adobe.com

Byron C. Wallace
Northeastern University
b.wallace@northeastern.edu

Ani Nenkova
Adobe Research
nenkova@adobe.com

Abstract

Many language tasks (e.g., Named Entity Recognition, Part-of-Speech tagging, and Semantic Role Labeling) are naturally framed as *sequence tagging* problems. However, there has been comparatively little work on interpretability methods for sequence tagging models. In this paper, we extend *influence functions* — which aim to trace predictions back to the training points that informed them — to sequence tagging tasks. We define the influence of a training instance *segment* as the effect that perturbing the labels within this segment has on a test segment level prediction. We provide an efficient approximation to compute this, and show that it tracks with the true segment influence, measured empirically. We show the practical utility of segment influence by using the method to identify systematic annotation errors in two named entity recognition corpora. Code to reproduce our results is available at https://github.com/successar/Segment_Influence_Functions.

Figure 1: We propose and evaluate *influence functions for sequence tagging tasks*, which retrieve snippets (from token a to b) in train samples that most influenced predictions for test tokens c through d. Here this reveals a training example in which an ORG is problematically marked as a LOC, leading to the observed error.

1 Introduction

Instance attribution methods aim to identify training examples that most informed a particular (test) prediction. The influence of training point k on test point i is typically formalized as the change in loss that would be observed for point i if example k was removed from the training set (Koh and Liang, 2017). Heuristic alternatives have also been developed to measure the importance of training samples during prediction, such as retrieving training examples similar to a test item (Pezeshkpour et al., 2021; Ilyas et al., 2022; Guo et al., 2021).

Influence functions can facilitate dataset debugging by helping to surface training samples which exhibit artifacts (Han et al., 2020). But on language tasks, most work on identifying training samples influential to a particular prediction has focused on classification tasks (Koh and Liang, 2017; Han et al., 2020; Peze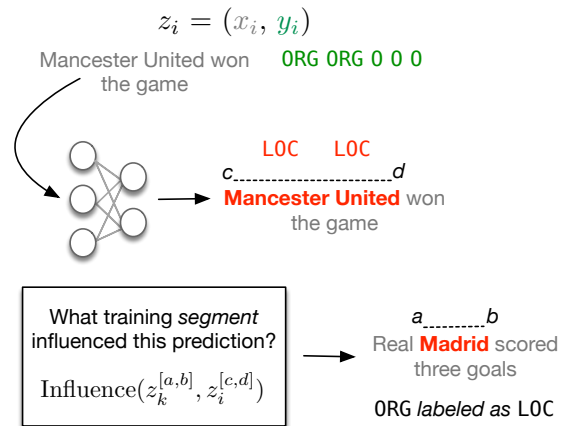shkpour et al., 2021). It is not immediately clear how we can extend such methods to the structured prediction problems such as named entity recognition (NER).

In this work we address this gap, presenting new methods for characterizing token-level influence for structured predictions (specifically sequence tagging tasks), and evaluating their use across illustrative datasets. More specifically, we focus on NER, one of the most common sequence tagging tasks. We extend influence functions to detect important training examples, i.e., those that most influenced the prediction *of a specific entity*, as opposed to being most influential with respect to the entire predicted label sequence. We call this extension *segment influence*.

Segment influence can help one perform fine-grained analysis of why specific segments of text were incorrectly labeled (as opposed to the entire sequence). Consider, for example, Figure 1. This shows a common issue in the CoNLL NER dataset: city names contained in soccer club titles tend to be mislabeled as location, rather than organization.

This in turn leads to similar mispredictions in the test set. For the shown test example we can use segment influence to ask *which entities within the training examples most informed the prediction made for the entity 'Manchester United'?* In principal, segment influence can directly recover the entities responsible for this systematic mislabeling.

Our main **contributions** are as follows. (1) We present a new method to approximately compute token-level influence for outputs produced by sequence tagging models; (2) We evaluate whether this approximation corresponds to *exact* influence values in linear models, and whether the method recovers intuitively correct training examples in synthetically constructed cases, and; (3) We establish the practical utility of approximating structured influence by using the method to identify systematic annotation errors in NER corpora.

2 Influence for Sequence Tagging

Consider a standard *sequence tagging* task in which the aim is to estimate the parameters θ of function f_θ which assigns to each token $x_{it} \in \mathcal{V}$ in an input sequence x_i (of length T_i) a label y_{it} from a label set $\mathcal{Y}$. Denote the training dataset by $\mathcal{D}$ where $\mathcal{D} = \{(x_i = \{x_{it}\}_{t=1}^{T_i}, y_i = \{y_{it}\}_{t=1}^{T_i})\}$.

Define f_θ as a model that yields conditional probability estimates for sequence label assignments: $p_\theta(y_i|x_i)$. Given parameter estimates $\hat{\theta}$, we can make a prediction for a test instance x_i by selecting the most likely y under this model: $\hat{y}_i = \text{argmax}_y p_{\hat{\theta}}(y|x_i)$. In structured prediction tasks we assume that the label y_{it} depends in part on labels $y_i \setminus y_{it}$, given the input x_i. In linear chain sequence tagging, this dependence can be formalized as a graphical model in which adjacent labels are connected; the most common realization of such a model is perhaps the *Conditional Random Field* (CRF; Lafferty et al. 2001).

We typically estimate θ by minimizing the negative log-likelihood of the training dataset $\mathcal{D}$.

$$\text{argmin}_\theta \frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} -\log p_\theta(y_i|x_i) \quad (1)$$

For brevity, we will also write the loss (negative log likelihood) of an example $z_i = (x_i, y_i)$ as $\mathcal{L}(z_i, \theta) = -\log p_\theta(y_i|x_i)$ and the overall loss over the training set by $\mathcal{L}(\mathcal{D}, \theta) = \frac{1}{|\mathcal{D}|} \sum_{z_i \in \mathcal{D}} \mathcal{L}(z_i, \theta)$.

2.1 Background: Influence Functions in ML

Influence Functions (Koh and Liang, 2017) retrieve training samples z_k deemed "influential" to the prediction made for a specific test sample x_i: $\hat{y}_i = f_{\hat{\theta}}(x_i)$. The exact influence of a training example z_k on a test example $z_i = (x_i, y_i)$ is defined as the change in the loss on z_i that would be incurred under parameter estimates *if the training sample z_k were removed prior to training*, i.e., $\mathcal{L}(z_i, \hat{\theta}_{-z_k}) - \mathcal{L}(z_i, \hat{\theta})$.

In practice this is prohibitively expensive to compute. Koh and Liang (2017) proposed an approximation. The idea is to measure the change in the loss on z_i observed when the loss associated with train sample z_k is slightly upweighted by some ϵ. Explicitly computing the effect of such an ϵ-perturbation is not feasible. Koh and Liang (2017) provide an efficient mechanism to approximate this (reproduced in Appendix A.1): $I(z_i, z_k) = -\nabla_\theta \mathcal{L}(z_i, \hat{\theta})[\nabla_\theta^2 \mathcal{L}(\mathcal{D}, \hat{\theta})]^{-1} \nabla_\theta \mathcal{L}(z_k, \hat{\theta})$, where $\nabla_\theta^2 \mathcal{L}(\mathcal{D}, \hat{\theta})$ is the Hessian of the loss $\mathcal{L}(\mathcal{D}, \hat{\theta})$ over the dataset with respect to θ.

Sequence tagging tasks by definition involve *multiple* predictions (and labels) per instance, and it is therefore natural to consider finer-grained influence. In particular, we would like to quantify the effect of *segments* of labels for z_k on a specific segment of the predicted output for z_i. For example, if we mispredict a particular entity within x_i, we may want to identify the train sample segment(s) most responsible for this error, especially if the model makes *systematic* errors that might be rectified by cleaning $\mathcal{D}$.

2.2 Segment Level Influence

We provide machinery to compute segment level influence. We want to quantify the impact of training tokens $x_{k[a,b]}$ (with labels $y_{k[a,b]}$), $1 \le a, b \le T_k$ on the loss of a segment of test point z_i. In NER, these segments may correspond to entities.

2.2.1 Exact Segment Influence

We define the *exact* influence of a segment $[a, b]$ within training example z_k on a segment $[c, d]$ of a test example $z_i = (x_i, y_i)$ as the change in loss that would be observed for reference token labels in segment $[c, d]$ of z_i, had we excluded the labels for segment $[a, b]$ within z_k from the training data. To make the above definition precise, we need to first define how training is to be performed when only partial annotations may be available for a given

train example (i.e., where a segment has been "removed"). We need also to formally define change in loss for a *segment* of a test example.

Start with training under partial annotations. Consider a training example $z_k = (x_k = \{x_{k1}, \ldots, x_{kT_k}\}, y_k = \{y_{k1}, \ldots, y_{kT_k}\})$. Assume we did not have labels for segment $[a, b]$ in y_k, i.e., labels $\{y_{ka}, \ldots, y_{kb}\}$ were missing. Denote such a partial label sequence by $y_k^{-[a,b]} = y_k \setminus \{y_{ka}, \ldots, y_{kb}\}$. Let $y_k^{[a,b]} = \{y_{ka}, \ldots, y_{kb}\}$. A natural way to handle such cases is to marginalize over all possible label assignments to the segment $[a, b]$ when computing the likelihood of this training example (Tsuboi et al., 2008):

$$p_\theta(y_k^{-[a,b]}|x_k) =$$
$$\sum_{y_a' \in \mathcal{Y}} \cdots \sum_{y_b' \in \mathcal{Y}} p_\theta(y_k^{-[a,b]} \cup \{y_a', \ldots, y_b'\}|x_k) \quad (2)$$

Denote the **marginal loss** of this partially annotated sequence as $\mathcal{ML}(z_k^{-[a,b]}, \theta) = -\log p_\theta(y_k^{-[a,b]}|x_k)$.

We can also write this marginal loss as the difference between the joint loss of y_k and the conditional loss of the segment $y_k^{[a,b]}$. This second form is more intuitive when we move to approximate exact influence values via ϵ-weighting.

$$\log p_\theta(y_k^{-[a,b]}|x_k) = \log p_\theta(y_k|x_k)$$
$$- \log p_\theta(y_k^{[a,b]}|y_k^{-[a,b]}, x_k) \quad (3)$$

$$\mathcal{ML}(z_k^{-[a,b]}, \theta) = \mathcal{L}(z_k, \theta) - \mathcal{CL}(z_k^{[a,b]}, \theta) \quad (4)$$

where we have defined also the **conditional loss** of the segment as $\mathcal{CL}(z_k^{[a,b]}, \theta) = -\log p_\theta(y_k^{[a,b]}|y_k^{-[a,b]}, x_k)$.

Next we define the change in loss for a segment of a test example $z_i = (x_i, y_i)$. We define the loss for the segment $[c, d]$ of the output y_i as the conditional loss of the segment $[c, d]$: $\mathcal{CL}(z_i^{[c,d]}, \theta)$.

Given the above definitions, we can concretize the notion of the *exact* influence as follows:

1. Retrain the model without the segment $[a, b]$ of training example z_k:
$$\hat\theta[z_k^{-[a,b]}] = \underset{\theta}{\mathrm{argmin}} \frac{1}{|\mathcal{D}|} \sum_{z_l \in \mathcal{D}} \mathcal{L}(z_l, \theta)$$
$$- \frac{1}{|\mathcal{D}|}(\mathcal{L}(z_k, \theta) - \mathcal{ML}(z_k^{-[a,b]}, \theta)) \quad (5)$$

Comparing Equations 4 and 5, we see that removing the effect of segment $[a, b]$ of z_k amounts to

subtracting the conditional loss of the segment $\mathcal{CL}(z_k^{-[a,b]}, \theta)$ from the original loss $\mathcal{L}(D, \theta)$.

2. Compute the difference between the conditional loss of segment $[c, d]$ of test example z_i under new parameter estimates $\hat\theta[z_k^{-[a,b]}]$ and the original estimates $\hat\theta$ trained using the objective in Equation 1:

$$\text{Exact-Influence}(z_k^{[a,b]}, z_i^{[c,d]}) =$$
$$\mathcal{CL}(z_i^{[c,d]}, \hat\theta[z_k^{-[a,b]}]) - \mathcal{CL}(z_i^{[c,d]}, \hat\theta) \quad (6)$$

2.2.2 Approximating Segment Influence

Above we have derived a means to calculate exact segment level influence values. But in practice retraining the model (step 1) is not feasible. Here we instead present an ϵ-upweighting method — analogous to the approximation to instance influence (Koh and Liang, 2017) — for computing segment influence. The idea is to compute the change in model parameters if we incur a slight additional penalty $\mathcal{CL}(z_k^{-[a,b]}, \theta)$ for the segment $[a, b]$:

$$\hat\theta_\epsilon[z_k^{-[a,b]}] = \underset{\theta}{\mathrm{argmin}} \frac{1}{|D|} \sum_{z_l \in \mathcal{D}} \mathcal{L}(z_l, \theta)$$
$$+ \epsilon \, \mathcal{CL}(z_k^{[a,b]}, \theta) \quad (7)$$

A first order approximation to the difference in the model parameters near $\epsilon = 0$ is given by:

$$\frac{d\hat\theta_\epsilon[z_k^{-[a,b]}]}{d\epsilon}\Big|_{\epsilon=0} =$$
$$- [\nabla_\theta^2 \mathcal{L}(\mathcal{D}, \hat\theta)]^{-1} \nabla_\theta \mathcal{CL}(z_k^{[a,b]}, \hat\theta) \quad (8)$$

We can apply the chain rule to measure the change in the conditional loss over segment $[c, d]$ of test example z_i due to this upweighting:

$$\mathrm{I}(z_i^{[c,d]}, z_k^{[a,b]}) = \frac{d\,\mathcal{CL}(z_i^{[c,d]}, \hat\theta_\epsilon[z_k^{-[a,b]}])}{d\epsilon}\Big|_{\epsilon=0} =$$
$$- \nabla_\theta \mathcal{CL}(z_i^{[c,d]}, \hat\theta)[\nabla_\theta^2 \mathcal{L}(\mathcal{D}, \hat\theta)]^{-1} \nabla_\theta \mathcal{CL}(z_k^{[a,b]}, \hat\theta)$$
$$(9)$$

This definition provides us with an approximation to the exact influence defined in previous section for a segment of a training example on a segment of a test example. Derivation showing all the steps can be found in Appendix A.2. We have assumed that we can take the gradient of the conditional loss over a segment, which is possible for a CRF tagger (derivation in Appendix B), but may be non-trivial for other models.

826

3 Computational Challenges

The computation costs of even approximate instance level influence can be prohibitive in practice, especially with the large pretrained language models that now dominate NLP. Computing and storing inverse Hessians of the loss has $O(p^3)$ and $O(p^2)$ complexity where p is the number of parameters in the model (commonly $\sim$100M-100B for deep models). Even ignoring the Hessian, one still needs the gradient with respect to each training example in $\mathcal{D}$; one could attempt to pre-compute these, but storing the results requires $O(|D|p)$ memory.

The alternative is therefore to recompute these for each new test point z_i. For segment level influence these costs are compounded because we need influence with respect to *every* segment within a training example, multiplying complexities by T^2, where T is the average length of a training example. Consequently, it is practically infeasible to calculate segment influence per Equation 9.

Prior work by Pezeshkpour et al. (2021) showed that for instance-level influence, considering a restricted set of parameters (e.g., those in the classification layer) when taking the gradient is reasonable for influence in that this does not much affect the induced rankings over train points with respect to influence. Similarly, ignoring the Hessian term does not significantly affect rankings by influence. These two simplifications dramatically improve efficiency, and we adopt them here.

Consider a sequence tagging model built on top of a deep encoder F, e.g., BERT (Devlin et al., 2018). In the context of a linear chain CRF on top of F, the standard score function for this model is: $s(y_i, x_i) = \sum_{t=1}^{T_i} \underline{y}_{it}^\top \boldsymbol{W} F(x_i)_t + \underline{y}_{i(t-1)}^\top \boldsymbol{T} \underline{y}_{it}$, where $\boldsymbol{T}$ is a matrix of class transition scores and $\underline{y}_{it}$ is the one-hot representation of label y_{it}. A CRF layer consumes these scores and computes the probability of a label sequence as $p(y_i|x_i) = \frac{e^{s(y_i, x_i)}}{\sum_{y' \in \mathcal{Y}^{T_i}} e^{s(y', x_i)}}$. In this work we consider the gradient only with respect to the $\boldsymbol{W}$ and $\boldsymbol{T}$ parameters above and not any parameters associated with F.

Further, we consider influence only with respect to individual token outputs in training samples, rather than every possible segment — i.e., we only consider single-token segments. This further reduces the T^2 terms in our complexity to T.

4 Experimental Aims and Setup

We evaluate segment influence in terms of: (1) *Approximation validity*, or how well the approximation proposed in Equation 9 corresponds to the exact influence value (Equation 6), and, (2) *Utility*, i.e., whether segment influence might help identify problematic training examples for sequence tagging tasks. In this paper we consider only NER tasks, using the following datasets.

CoNLL (Tjong Kim Sang and De Meulder, 2003). An NER dataset containing news stories from Reuters, labeled with four entity types: PER, LOC, ORG and MISC, and demarcated using the beginning-inside-out (BIO) tagging scheme (Ramshaw and Marcus, 1999). The dataset is divided into train, validation, and test splits comprising 879, 194, and 197 documents, respectively.

EBM-NLP (Nye et al., 2018). A corpus of annotated medical article abstracts describing clinical randomized controlled trials. 'Entities' here are spans of tokens that describe the patient Population enrolled, the Interventions compared, and the Outcomes measured ('PICO' elements). This dataset includes a test set of 191 abstracts labeled by medical professionals, and a train/val set of 3836/958 abstracts labeled via Amazon Mechanical Turk.

For NER models we use a representative modern transformer — BigBird Base (Zaheer et al., 2020) — as our encoder F, using final layer hidden states as contextualized token representations. Dependencies between output labels are modeled using a CRF. We provide training details in Appendix C.

In addition to segment and instance level influence we also evaluate — where applicable — **segment nearest neighbor** as an attribution method, which works as follows. We retrieve from the training dataset segments that have the highest similarity between their corresponding feature embeddings (we again consider only single token segments here, so we do not need to worry about embedding multi-token segments). We consider both dot product and cosine similarity and report the results for the version that gives best performance for each experiment.

5 Validating Segment Influence

In this first set of experiments we aim to (i) verify that the proposed approximation correlates with the *exact* influence (2.2.2), and, (ii) ascertain that for synthetic tasks which we construct, segment influence returns the *a priori* expected training segment

for given text examples (5.2). In the latter experiments, we compare the fine-grained error analysis afforded by segment influence to that made possible using instance-level attribution.

5.1 How Good is our Approximation?

How well does the *approximation* we have proposed for segment influence correlate with the *exact* value? The latter is in general intractable to compute. As a practical means to validate our approximation we use a simple linear model trained on the CoNLL corpus, which makes it feasible (though still costly) to compute exact influence via brute force. This allows us to compare the actual segment influence to our approximation.

We subset our training set to a 1000 examples and the validation set to 200 examples (given the computational expense of model retraining). As token-level features, we use GloVe word embeddings (Pennington et al., 2014) and additional syntactical features (See Appendix C.1). We use the L-BFGS optimizer available in PyTorch (Paszke et al., 2019) to train this model, stopping when the maximum absolute gradient value reaches 10^{-6}.

We randomly sample 20 mispredicted validation tokens. For each such token, we identify the 20 most influential tokens in the training set according to their absolute approximated influence values. We combine these 400 tokens together in a single pool, remove these tokens sequentially prior to training by subtracting their conditional loss (Equation 6), and retrain the model. We then take the difference in the observed loss for each of the 20 validation token under the retrained and original model.

Figure 2 plots the *actual* difference in the conditional loss obtained for the sampled validation tokens against the *predicted* change in loss using the influence approximation (Equation 9). The quantities have a Pearson correlation of 0.89, suggesting that the approximation is reasonable, though imperfect; some deviations exist (likely due to numerical instability computing the inverse Hessian).

5.2 Synthetic Artifact Insertion

We next evaluate whether segment influence may help practitioners 'debug' dataset issues by identifying (problematic) entities within train examples that may have led to test time entity mispredictions.

We again use the CoNLL training set for this exercise, artificially inserting an "artifact" into training samples. Specifically, for a random 10%
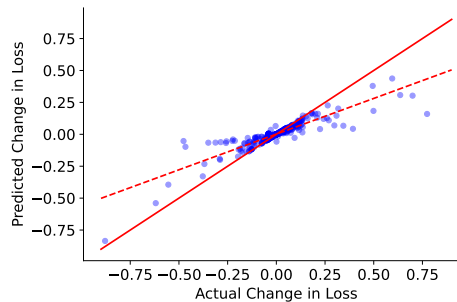

Figure 2: Predicted versus actual change in conditional loss over a sample of mispredicted validation tokens from CoNLL. The dotted line shows the line of best fit to the observations (solid line is perfect correlation).

of train instances, we insert an "artifact token" (`special`) at a position selected at uniform random from range 1 to T-1. When this insertion is made, we also deterministically set the label of the token immediately following the artifact to `B-PER`. As an example, consider the example *Joe*/PER *Biden*/PER *is US*/LOC *president in 2021*. We might modify this to the following: *Joe*/PER *Biden*/PER *is US*/LOC *president* `special` *in*/PER *2021*. We train a model on this modified train set.

If the model learns the correlation between artifact tokens and the entity label for the right adjacent token, then for test examples featuring the artifact it should return artifact adjacent tokens (e.g., *in* in the above example) as influential. To test this, we insert the artifact randomly into 10% of the validation instances. The model predicts PER for all tokens that immediately follow the artifact. We apply segment influence for these token predictions, and find that for *all* such examples, the most influential token retrieved using segment influence is adjacent to the inserted artifact. Segment nearest neighbor also recovers these problematic samples. By contrast, applying instance influence to the same validation examples yields training examples exhibiting the artifact as most influential only 26.3% of the time.

This result provides evidence that we are able to retrieve plausible error-causing entities within the training dataset that affect a test entity prediction. Standard instance-level influence is not as useful here because it evaluates influence with respect to an entire example, and not in terms of its constituent entities. This highlights the need for and potential utility of *segment* influence methods.

6 Use Cases for Segment Influence

Misannotations or incomplete annotations are a common problem in sequence tagging tasks. In

contrast to standard classification tasks, in the case of sequence tagging annotators must label multiple spans of variable length within a given text. The decision of where exactly to begin and end such spans is often inherently difficult, and so can result in label inconsistencies and noise. Next we evaluate the utility of segment influence (relative to alternative methods) for helping to identify such problematic training data for sequence tagging tasks.

6.1 Finding noisy labels in CoNLL

CoNLL has been the *de facto* standard NER dataset in NLP for over a decade, despite known annotation issues (Reiss et al., 2020) including: missing and incomplete entities; incorrect entity labels; questionable entity boundaries, among others. To assess whether segment influence can unveil these issues we calculate the influence of train tokens on mispredictions; instances with higher scores are those we would expect to contain labeling errors.

Baselines We evaluate several baselines for identifying noisy instances. At the instance-level, these include instance loss and the ℓ_2 gradient norm with respect to CRF parameters. For token-level baselines we again compute conditional loss and gradient norms, as well as entropy over the predicted distributions of labels for each token. We aggregate token scores by taking means or maximum values. Higher values suggest noisy annotations.

Influence-based Methods We also explore whether a small set of corrected examples (10 instances) can be used together with influence functions to identify noisy examples. The intuition is that if including a training example (or a segment thereof) in the train set increases the loss for a (clean) validation example (or segment), then this training sample may be incorrectly labeled. Therefore, a misannotation score for a training example can then be computed by measuring the influence it appears to exert in the *wrong* direction on the prediction for a clean validation example. Note that a positive influence value indicates that the loss for a validation example will increase if we include the corresponding training example/segment.

To score training samples using instance-level attribution methods, we average its **instance influence** over the (10) corrected validation samples. To derive a single score using token-level influence, we first compute the **segment influence** of a train token on all validation sample tokens, and then average the top-10 highest values. We take this ag-

gregate measure over only the top-k tokens because averaging over all tokens would lose the granularity captured by segment influence. We also derive a **segment-level NN** score for a training token by taking dot-product similarities with all validation tokens belonging to a different class and averaging the 10 highest values. As above, we compute a training instance misannotation score by taking the mean or maximum of the token level scores.

Results Figure 3a plots the rank order of examples under different scoring functions (we include a representative subset of scoring functions here; all results in Appendix E) against the fraction of documents retrieved which contain any annotation errors identified by (Reiss et al., 2020). Influence methods efficiently recover noisy training samples, but in this case so do methods such as ranking instances according to average token loss. But this aggregate result may hide differences in the *types* of noise methods are able to identify. We hypothesize that while simpler methods may recover instances with *random* labeling errors, structured influence may better identify *systematic* labeling errors in training data.

6.2 Finding Systematic Labeling Errors

CoNLL contains many apparently *random* annotation errors. These errors lack a discernible pattern and may occur if the annotator makes a mistake due to a lack of focus while annotating. But in addition to these, token labels also exhibit a *systematic inconsistency* where sports teams (ORGs) are often mislabeled as LOCs. Suppose a practitioner observed a model consistently mislabeling sports teams as locations on held out samples. Ideally, they could recover (and fix) the *source* of this systematically incorrect behavior. By contrast, it is less clear how to reduce *random* mistakes made by the model. We envision the practitioner using segment influence with respect to ORG entity tokens mispredicted as LOCs to identify (segments of) training points that have resulted in this behavior; these could then be relabeled.

To test whether influence-based methods are better able to unveil systematic labeling errors (compared to simpler methods), we run an experiment using a "cleaned" version of CoNLL (Reiss et al., 2020), which has fixed the aforementioned sports team inconsistency. We then reintroduce the inconsistency for a subset of training examples about soccer. Specifically, we select 20 documents (out
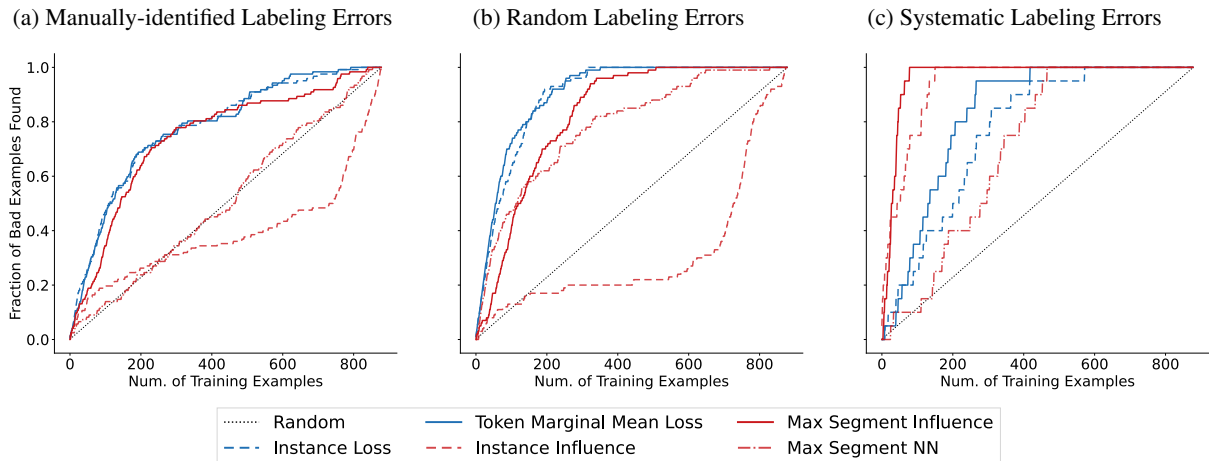
(a) Manually-identified Labeling Errors (b) Random Labeling Errors (c) Systematic Labeling Errors

Legend:
- Random (dotted)
- Instance Loss (dashed blue)
- Token Marginal Mean Loss (solid blue)
- Instance Influence (dashed red)
- Max Segment Influence (solid red)
- Max Segment NN (dash-dot red)

Figure 3: Finding problematic CoNLL examples in the train set using different scoring functions. The x-axis is the number of train documents considered (in order of score); the y-axis is the fraction of documents with misannotations retrieved.

of 105) that start with the word 'SOCCER' and convert any token labeled as ORG within them to LOC whenever that token corresponds to a city name in Wikidata (see Appendix D). We also introduce *random* noise into 100 randomly selected documents in the remaining data, replacing the label for every entity with a randomly selected label.

We apply the scoring methods discussed above to the training set to generate misannotation scores. For influence based methods, we randomly select 10 validation documents about soccer. This experiment is set up to simulate conditions where users can interactively identify issues with training data via error analysis. It reflects the envisioned use case, where a practitioner might seek out training data responsible for systematic mispredictions observed on a small held-out set. We present results for random and systematic labeling errors in Figures 3b and 3c, respectively.

For random noise (Figure 3b), influence-based methods perform slightly worse than our baselines. However, with respect to identifying training samples exhibiting a specific type of noise (Figure 3c), influence-based methods substantially outperform baselines. This agrees with the hypothesis that influence-based methods can help efficiently identify noisy examples that have resulted in the model making a specific type of errors. However, in this case instance-level attribution fares almost as well as segment influence. We next report results for a case where segment-level influence offers comparative benefits.

Patients received either quality-controlled chloroquine aiming for a target total dose of **25 mg base/kg** in ...	dextran-70 infusion was used at the dose of **7.5 ml/kg** for 30 minutes before CPB ...

Table 1: Example of dosage mislabeling in the EBM-NLP dataset. In the left instance, dosage is labeled as part of the intervention; in the right instance it is not.

6.3 Dosage Mislabeling in EBM-NLP

We consider the task of annotating medical *interventions* (e.g., aspirin) in abstracts of articles describing clinical trials. For this we use the EBM-NLP dataset (Nye et al., 2018). Using segment influence, we identified a type of systematic noise in the training set: Annotators (lay workers recruited on Mechanical Turk) sometimes included *dosage* information in their intervention spans, and sometimes did not. This in turn gave rise to apparent errors on the test set, which was labeled by medical experts who did not consider dosages as part of intervention. We provide examples in Table 1.

We aim to evaluate the degree to which different attribution methods might help practitioners identify such inconsistent labels in the train set. Here we only consider influence based methods: Instance-level influence functions, segment influence functions, and segment nearest neighbors. For each method, we characterize whether the top influential training examples (or segment within) could be used to identify the fact that inconsistent labeling for dosages occur in the training set (leading to the apparent mispredictions on the test set).

Define a *supporting* example as one whose inclusion in training decreases the loss for the test examples/segment with respect to its label (i.e., it

830

Attribution method	Test example	Top supporting example	Top opposing example
Instance Influence	... assessed the effects of chronic treatment with low doses of aspirin (100 mg/-day) on clinic and ambulatory systolic (SBP) ...	... the relationship between easy skin bruising and other systemic effects of ICS therapy , including adrenal suppression ...	... Despite enthusiasm for the use of mild hypothermia during neurosurgical procedures , this therapy ...
Segment Influence	Nitrendipine was given orally at a dosage of 20 mg once ...	Alinidine was administered at a dosage of 30 mg ...	The dose of oral famotidine was 2 mg/kg/day ...

Table 2: Dosage label inconsistency as identified via different attribution methods. For each method, we are showing an example of the most typical result achieved. Segment Influence can identify the inconsistent labels for dosage entities in the training dataset. Segment NN retrieves similar examples when explicitly checking for inconsistency.

Method	Example	Token
Instance inf.	35.4	–
Segment inf.	97.9	85.4
Segment NN	87.5	83.3

Table 3: Segment influence reliably surfaces dosage labeling inconsistency. For mispredicted dosage test tokens, we report whether top supporting/opposing examples and tokens have conflicting dosage labels.

has negative influence value), and an *opposing* example as one whose inclusion increases the loss for the same. To assess the ability of segment influence to flag systematic dosage mislabeling, we use test examples in which dosages were mispredicted as interventions, and measure whether: (a) the top *supporting* example excludes a dosage from the intervention span, *and* (b) the top *opposing* example has a dosage labeled as intervention. Surfacing such conflicting instances as the most influential training points (pointing in opposite directions) for a dosage that appears in a test segment readily suggests the label inconsistency issue at play.

We find that **instance influence** functions recover such inconsistency in only 35% of test examples (Table 3), while **segment influence** identifies such cases for ∼97% of these. This supports the use of segment influence for identifying errors in a specific type of entity which may not be apparent when identified via instance influence functions.

We also consider **segment nearest neighbor**. Here for a given test point where a dosage token has been mispredicted as an intervention, we retrieve the example containing the most similar token (under the token representation induced by F) and check if it excludes dosage from its intervention span. This occurs for 97% of the test dosages. However, retrieving the example with the *least* similar token (as an analog to an "opposing" example in the case of influence methods) yields no dosages (see Table 2). This is because dissimilar examples tend not to be useful for analysis. To assess whether segment NN might help verify a hypothesis regarding label inconsistency, we explicitly retrieve the two *most similar* tokens that have the same and

a distinct label as the test example, respectively. These two "nearby" instances with conflicting labels can be used to check for inconsistency. Typical examples from each of the above methods are reproduced in Table 2.

7 Related Work

Influence functions originated in statistics in the context of analyzing linear models (Cook and Weisberg, 1982; Chatterjee and Hadi, 1986; Hampel et al., 2011). Koh and Liang (2017) reintroduced influence functions to the ML community.

While influence remains the most common method for training data attribution, other methods have also been proposed to identify "important" instances, including: Shapley Values (Ghorbani and Zou, 2019); Fisher Kernels (Khanna et al., 2019); tracking instance gradients during training (Pruthi et al., 2020; Chen et al., 2021); and training surrogate linear models (Ilyas et al., 2022). Another line of work aims to make the computation of influence more efficient (Guo et al., 2021; Schioppa et al., 2022).

Influence functions have been shown to be useful for: aiding training data debugging and artifact identification (Han et al., 2020; Han and Tsvetkov, 2021; Zylberajch et al., 2021; Pezeshkpour et al., 2022), understanding bias in data (Brunet et al., 2019), robust optimization (Lee et al., 2020; Deng et al., 2020), active learning (Xu and Kazantsev, 2019), data cleaning (Wang et al., 2021; Kong et al., 2022), and domain adaptation (Grangier and Iter, 2022).

Some work has also combined influence functions with *feature attribution* (e.g., integrated gradients) to point to specific tokens within instances that were influential (Koh and Liang, 2017; Pezeshkpour et al., 2022; Zhang et al., 2021). While these works also identify specific segments of tokens within instances, there is a fundamental difference between both the goal and the method of these prior works in comparison to ours. These prior efforts provided mechanisms to approximate

which tokens most contribute to the influence score of a given example (that is, locating parts of the *input* text which, if perturbed, would greatly affect the influence of said point). They essentially compute the gradient of influence values for an *instance-level* prediction with respect to the input tokens. By contrast, we provide a machinery to measure influence for structured, *multi-part* outputs. Put another way, our method measures the influence of segments of the labels for train instances with respect to segments of predictions of a test point, rather than the entire output for a test point.

Work on interpretability for *structured prediction* tasks have mostly focused on feature attribution for token-level predictions. Such methods have been used to characterize model behavior (Agarwal et al., 2021; Alvarez-Melis and Jaakkola, 2017), extract rationales (Vafa et al., 2021), analyze internal model representations (Clark et al., 2019; Vig et al., 2020) and debug erroneous text generations (Strobelt et al., 2018). Although rare, some structured prediction works have attempted to trace model behavior back to the training set but they invariably use influence with respect to whole structured output (Wang et al., 2021; Schioppa et al., 2022).

8   Conclusions

We have presented a method for computing token-level influence values for sequence tagging tasks, i.e., *segment influence*. We validated this method via synthetic experiments, ascertaining that it retrieves the expected tokens as 'influential' for predictions made on held-out examples containing specific artifacts. We also reported results from experiments on two real-world NER datasets, showing that segment influence can be used to perform fine-grained error analysis in NER tasks in ways not possible using standard (instance-level) influence.

9   Limitations

In this paper we provided a method to compute influence functions for token level predictions in sequence tagging tasks, and showed the utility of this with respect to identifying noisy labels in NER training data. The exact influence functions defined in Section 2.2.1 do not explicitly depend on the sequential nature of the output or the use of a Linear Chain CRF structure; the same derivation can be used for any structured prediction model. However, there are important limitations to this method, both theoretical and in terms of efficiency.

The immediate problem occurs in the derivation of Exact Influence. The objective for retraining after removal of a segment is not convex due to the presence of marginal likelihood term (Equation 5) and therefore susceptible to presence of multiple minima.

We have assumed a probabilistic model over outputs conditioned on an input sequence, and that the training objective is to minimize the negative log-likelihood over the training data under this model. The approach may not be readily amenable to alternative loss functions or model classes (e.g., structured SVMs; Tsochantaridis et al. 2004).

Even considering only graphical models for structured prediction, our approach requires the ability to compute conditional probabilities for any subset of outputs and the corresponding gradient for model parameters. We have focused on linear-chain CRFs, which permits simple computation of conditional probabilities and gradients; this may not be the case for other such structured prediction tasks (See Appendix H).

Finally, we reiterate the efficiency issues from Section 3 prevalent for sequence tagging tasks themselves. Even with simplifications (using only the final layer parameters and ignoring the hessian matrix), we must store a vector of size at least $d \cdot C$ for every segment within every example in the training set (d denoting the feature vector size and C the number of classes).

As an example, consider CoNLL dataset trained using BERT-base model ($d = 768, C = 9$). A single vector requires 27KB of space; storing the vector for every segment in every training example requires ~240GB of space. Clearly, this is infeasible for larger datasets. Even restricting to token level (as we have above) requires 9GB of space. To scale up to larger datasets and sequence lengths, we need a way to reduce these space requirements, perhaps via compression. In addition, at token level, we may reduce the size of vector needed to be stored to $(d + C)$ by exploiting the fact that the gradient can be written as a outer product of the feature and the error vector (See Appendix B); in CoNLL, for example, this can reduce space requirements by 9x.

10   Broader Impact Statement

Large-scale pre-trained models for NLP are being deployed with increasing frequency, given their em-

pirical success. But these models remain opaque, making it difficult to know why a model made any specific prediction. Training data problems such as artifacts, biases, and labeling errors constitute common sources of model misbehaviors, and instance attribution methods—such as what we have proposed in this work—provide a potential mechanism to unearth and ultimately fix such issues. However, while influence methods like the proposed segment influence functions may provide one means to identify training data issues, we would caution against using absence of errors discovered via these methods as the proof of their non-existence in the dataset. These tools are not perfect, and one should make an independent judgment of the benefits and risks of releasing a newly developed models whose behavior we may not *fully* understand.

We also note that segment influence methods require significant storage and compute resources, with concomitant environmental implications. Practitioners should therefore weigh the potential benefits of using these methods as error analysis tools against the energy consumption costs. We hope that future developments in efficient storage and computation of influence vectors might soon mitigate this particular consideration.

References

Oshin Agarwal, Yinfei Yang, Byron C. Wallace, and Ani Nenkova. 2021. Interpretability analysis for named entity recognition to understand system predictions and how they can improve. *Computational Linguistics*, 47(1):117–140.

David Alvarez-Melis and Tommi Jaakkola. 2017. A causal framework for explaining the predictions of black-box sequence-to-sequence models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 412–421, Copenhagen, Denmark. Association for Computational Linguistics.

Marc-Etienne Brunet, Colleen Alkalay-Houlihan, Ashton Anderson, and Richard Zemel. 2019. Understanding the origins of bias in word embeddings. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 803–811. PMLR.

Samprit Chatterjee and Ali S Hadi. 1986. Influential observations, high leverage points, and outliers in linear regression. *Statistical science*, pages 379–393.

Yuanyuan Chen, Boyang Li, Han Yu, Pengcheng Wu, and Chunyan Miao. 2021. Hydra: Hypergradient data relevance analysis for interpreting deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7081–7089.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy. Association for Computational Linguistics.

R Dennis Cook and Sanford Weisberg. 1982. Criticism and influence analysis in regression. *Sociological methodology*, 13:313–361.

Zhun Deng, Cynthia Dwork, Jialiang Wang, and Linjun Zhang. 2020. Interpreting robust optimization via adversarial influence functions. In *International Conference on Machine Learning*, pages 2464–2473. PMLR.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Johanna Geiß, Andreas Spitz, and Michael Gertz. 2018. Neckar: A named entity classifier for wikidata. In *Language Technologies for the Challenges of the Digital Age*, pages 115–129, Cham. Springer International Publishing.

Amirata Ghorbani and James Zou. 2019. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*, pages 2242–2251. PMLR.

David Grangier and Dan Iter. 2022. The trade-offs of domain adaptation for neural language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3802–3813, Dublin, Ireland. Association for Computational Linguistics.

Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. 2021. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH)*, 3(1):1–23.

Han Guo, Nazneen Rajani, Peter Hase, Mohit Bansal, and Caiming Xiong. 2021. Fastif: Scalable influence functions for efficient model interpretation and debugging. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10333–10350.

Frank R Hampel, Elvezio M Ronchetti, Peter J Rousseeuw, and Werner A Stahel. 2011. *Robust statistics: the approach based on influence functions*, volume 196. John Wiley & Sons.

Xiaochuang Han and Yulia Tsvetkov. 2021. Influence tuning: Demoting spurious correlations via instance attribution and instance-driven updates. In *Findings*

of the Association for Computational Linguistics: EMNLP 2021, pages 4398–4409.

Xiaochuang Han, Byron C Wallace, and Yulia Tsvetkov. 2020. Explaining black box predictions and unveiling data artifacts through influence functions. *arXiv preprint arXiv:2005.06676*.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python. *Zenodo*.

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. 2022. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*.

Rajiv Khanna, Been Kim, Joydeep Ghosh, and Sanmi Koyejo. 2019. Interpreting black box predictions using fisher kernels. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3382–3390. PMLR.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1885–1894.

Shuming Kong, Yanyan Shen, and Linpeng Huang. 2022. Resolving training biases via influence-based data relabeling. In *International Conference on Learning Representations*.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Donghoon Lee, Hyunsin Park, Trung Pham, and Chang D Yoo. 2020. Learning augmentation network via influence functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10961–10970.

Benjamin Nye, Junyi Jessy Li, Roma Patel, Yinfei Yang, Iain Marshall, Ani Nenkova, and Byron Wallace. 2018. A corpus with multi-level annotations of patients, interventions and outcomes to support language processing for medical literature. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 197–207, Melbourne, Australia. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward

Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Pouya Pezeshkpour, Sarthak Jain, Sameer Singh, and Byron Wallace. 2022. Combining feature and instance attribution to detect artifacts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1934–1946, Dublin, Ireland. Association for Computational Linguistics.

Pouya Pezeshkpour, Sarthak Jain, Byron C Wallace, and Sameer Singh. 2021. An empirical comparison of instance attribution methods for nlp. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 967–975.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. 2020. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930.

Lance A Ramshaw and Mitchell P Marcus. 1999. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer.

Frederick Reiss, Hong Xu, Bryan Cutler, Karthik Muthuraman, and Zachary Eichenberger. 2020. Identifying incorrect labels in the CoNLL-2003 corpus. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 215–226, Online. Association for Computational Linguistics.

Andrea Schioppa, Polina Zablotskaia, David Vilar Torres, and Artem Sokolov. 2022. Scaling up influence functions. In *AAAI-22*.

Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander Rush. 2018. Debugging sequence-to-sequence models with Seq2Seq-vis. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 368–370, Brussels, Belgium. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104.

Yuta Tsuboi, Hisashi Kashima, Shinsuke Mori, Hiroki Oda, and Yuji Matsumoto. 2008. Training conditional random fields using incomplete annotations. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 897–904, Manchester, UK. Coling 2008 Organizing Committee.

Keyon Vafa, Yuntian Deng, David M Blei, and Alexander M Rush. 2021. Rationales for sequential predictions. *arXiv preprint arXiv:2109.06387*.

Jesse Vig, Ali Madani, Lav R Varshney, Caiming Xiong, Nazneen Rajani, et al. 2020. Bertology meets biology: Interpreting attention in protein language models. In *International Conference on Learning Representations*.

Xinyi Wang, Ankur Bapna, Melvin Johnson, and Orhan Firat. 2021. Gradient-guided loss masking for neural machine translation. *arXiv preprint arXiv:2102.13549*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Minjie Xu and Gary Kazantsev. 2019. Understanding goal-oriented active learning via influence functions. *arXiv preprint arXiv:1905.13183*.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297.

Wei Zhang, Ziming Huang, Yada Zhu, Guangnan Ye, Xiaodong Cui, and Fan Zhang. 2021. On sample based explanation methods for NLP: Faithfulness, efficiency and semantic evaluation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5399–5411, Online. Association for Computational Linguistics.

Hugo Zylberajch, Piyawat Lertvittayakumjorn, and Francesca Toni. 2021. HILDIF: Interactive debugging of NLI models using influence functions. In *Proceedings of the First Workshop on Interactive Learning for Natural Language Processing*, pages 1–6, Online. Association for Computational Linguistics.

A Derivation of Influence Functions

A.1 Instance Influence Functions

We provide a derivation of instance Influence Functions here for completeness (originally derived by Koh and Liang 2017). Consider $\hat{\theta}$, the parameters that minimize the loss function $\mathcal{L}$ over our training dataset.

$$\mathcal{L}(\mathcal{D}, \theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(z_i, \theta) \qquad (10)$$

The loss function is assumed to be twice-differentiable and strongly convex, such that a positive-definite Hessian $\nabla^2_\theta \mathcal{L}(\mathcal{D}, \theta)$ exists.

Influence Functions measure the change in the loss of some test example z_t if we slightly upweight the training example z_i during training. Under such upweighting, the new parameters can be written as:

$$\hat{\theta}_{\epsilon,i} = \underset{\theta}{\mathrm{argmin}}\, \mathcal{L}(\mathcal{D}, \theta) + \epsilon \mathcal{L}(z_i, \theta) \qquad (11)$$

Define the change in parameters as $\delta_\epsilon = \hat{\theta}_{\epsilon,i} - \hat{\theta}$. The rate of change of parameters with respect to ϵ can be written as:

$$\frac{d\delta_\epsilon}{d\epsilon} = \frac{d\hat{\theta}_{\epsilon,i}}{d\epsilon} \qquad (12)$$

Since the new parameters minimize the perturbed loss function, we can consider its first-order optimality criteria:

$$\nabla_\theta \mathcal{L}(\mathcal{D}, \hat{\theta}_{\epsilon,i}) + \epsilon \nabla_\theta \mathcal{L}(z_i, \hat{\theta}_{\epsilon,i}) = 0 \qquad (13)$$

Assuming $\hat{\theta}_{\epsilon,i} \to \hat{\theta}$ as $\epsilon \to 0$, we perform a Taylor-expansion from the left hand side around $\hat{\theta}$:[1]

$$\nabla_\theta \mathcal{L}(\mathcal{D}, \hat{\theta}) + \epsilon \nabla_\theta \mathcal{L}(z_i, \hat{\theta})$$
$$+ [\nabla^2_\theta \mathcal{L}(\mathcal{D}, \hat{\theta}) + \epsilon \nabla^2_\theta \mathcal{L}(z_i, \hat{\theta})]\delta_\epsilon = 0$$

Dropping the higher order terms in $|\delta_\epsilon|$ and noting that the first term is zero (due to first-order optimality of original loss; $\nabla_\theta \mathcal{L}(\mathcal{D}, \hat{\theta}) = 0$), we arrive at the following equation:

---

[1] Here, $\nabla^2_\theta \mathcal{L}(\mathcal{D}, \hat{\theta})$ is the hessian of the loss $\mathcal{L}(\mathcal{D}, \hat{\theta})$

$$\epsilon \nabla_\theta \mathcal{L}(z_i, \hat{\theta}) + [\nabla_\theta^2 \mathcal{L}(\mathcal{D}, \hat{\theta}) + \epsilon \nabla_\theta \mathcal{L}(z_i, \hat{\theta})]\delta_\epsilon = 0$$

$$\delta_\epsilon = -[\nabla_\theta^2 \mathcal{L}(\mathcal{D}, \hat{\theta}) + \epsilon \nabla_\theta^2 \mathcal{L}(z_i, \hat{\theta})]^{-1} \epsilon \nabla_\theta \mathcal{L}(z_i, \hat{\theta})$$

$$\delta_\epsilon = -[\nabla_\theta^2 \mathcal{L}(\mathcal{D}, \hat{\theta})]^{-1} \epsilon \nabla_\theta \mathcal{L}(z_i, \hat{\theta})$$

Where in the last step we have dropped the higher order terms in ϵ. Using above expression and Equation 12, we get the final form of the derivative of the parameters with respect to ϵ:

$$\frac{d\hat{\theta}_{\epsilon,i}}{d\epsilon} = -[\nabla_\theta^2 \mathcal{L}(\mathcal{D}, \hat{\theta})]^{-1} \nabla_\theta \mathcal{L}(z_i, \hat{\theta}) \quad (14)$$

Given the equation for derivative of parameters, we can compute the derivative of the loss of the test example with respect to ϵ by chain rule:

$$\frac{d\mathcal{L}(z_t, \hat{\theta}_{\epsilon,i})}{d\epsilon} = \frac{d\mathcal{L}(z_t, \hat{\theta})}{d\theta}^\top \frac{d\hat{\theta}_{\epsilon,i}}{d\epsilon}$$
$$= -\nabla_\theta \mathcal{L}(z_t, \hat{\theta})[\nabla_\theta^2 \mathcal{L}(\mathcal{D}, \hat{\theta})]^{-1} \nabla_\theta \mathcal{L}(z_i, \hat{\theta}) \quad (15)$$

A.2 Segment Influence Functions

The derivation for Segment Influence Functions is similar to the proceeding. The only difference involves replacing the $\mathcal{L}(z_i, \theta)$ in Equation 11 with the conditional loss of the segment $\mathcal{CL}(z_i^{[a,b]}, \theta)$, yielding:

$$\hat{\theta}_{\epsilon,i[a,b]} = \underset{\theta}{\arg\min}\, \mathcal{L}(\mathcal{D}, \theta) + \epsilon\, \mathcal{CL}(z_i^{[a,b]}, \theta) \quad (16)$$

The rest of the derivation remains the same, and ultimately provides the derivative of the parameters with respect to ϵ as:

$$\frac{d\hat{\theta}_{\epsilon,i[a,b]}}{d\epsilon} = -[\nabla_\theta^2 \mathcal{L}(\mathcal{D}, \hat{\theta})]^{-1} \nabla_\theta \mathcal{CL}(z_i^{[a,b]}, \theta) \quad (17)$$

B Gradient with Respect to CRF Parameters

We begin by reiterating the definition of joint probability under a CRF of an instance (x, y) and the marginal probability of a partial label sequence $y_{-[a,b]}$ below:

$$p(y|x) = \frac{e^{s(y,x)}}{Z(x)}$$

$$p(y_{-[a,b]}|x) = \sum_{y'_a \in \mathcal{Y}} \cdots \sum_{y'_b \in \mathcal{Y}} \frac{e^{s(y',x)}}{Z(x)}$$

Where $Z(x)$ is the normalization term for the CRF, which is independent of sequence labels (i.e., depends only on x) and $y' = y_{-[a,b]} \cup \{y'_a, \ldots, y'_b\}$.

Using above definitions, the conditional probability under a CRF of a segment $[a, b]$ for a instance (x, y) is given by:

$$p(y_{[a,b]}|y_{-[a,b]}, x) = \frac{p(y|x)}{p(y_{-[a,b]}|x)}$$
$$= \frac{e^{s(y,x)}}{\sum_{y'_a \in \mathcal{Y}} \cdots \sum_{y'_b \in \mathcal{Y}} e^{s(y',x)}} \quad (18)$$

In a linear-chain CRF, the score function $s(y, x)$ can be divided into sum of three parts. Therefore we can write $e^{s(y,x)}$ as a product of three parts:

- Terms depending only on $y_{-[a,b]}$: These are terms of form $\underline{y}_t^\top \boldsymbol{W} F(x)_t$ and $\underline{y}_{t-1}^\top \boldsymbol{T} \underline{y}_t$ where $t, t-1 \notin [a, b]$. Call them collectively as $s_{-[a,b]}$

- Terms depending only on $y_{[a,b]}$: These are terms of form $\underline{y}_t^\top \boldsymbol{W} F(x)_t$ and $\underline{y}_{t-1}^\top \boldsymbol{T} \underline{y}_t$ where $t, t-1 \in [a, b]$. Call them collectively as $s_{[a,b]}$.

- Interaction Terms: Only two such terms exist $- \underline{y}_{a-1}^\top \boldsymbol{T} \underline{y}_a$ and $\underline{y}_b^\top \boldsymbol{T} \underline{y}_{b+1}$. Call them collectively as T_I.

(As in the main text, we use $\underline{y}_t$ to denote the one-hot representation of label y_t.) Applying it to the formula for conditional probability above, we have:

$$p(y_{[a,b]}|y_{-[a,b]}, x)$$
$$= \frac{e^{s_{-[a,b]}} e^{s_{[a,b]}+T_I}}{\sum_{y'_a \in \mathcal{Y}} \cdots \sum_{y'_b \in \mathcal{Y}} e^{s_{-[a,b]}} e^{s_{[a,b]}+T_I}}$$
$$= \frac{e^{s_{-[a,b]}} e^{s_{[a,b]}+T_I}}{e^{s_{-[a,b]}} \sum_{y'_a \in \mathcal{Y}} \cdots \sum_{y'_b \in \mathcal{Y}} e^{s_{[a,b]}+T_I}}$$
$$= \frac{e^{s_{[a,b]}+T_I}}{\sum_{y'_a \in \mathcal{Y}} \cdots \sum_{y'_b \in \mathcal{Y}} e^{s_{[a,b]}+T_I}}$$

In Step 2 above, we note that the terms collectively represented by $s_{-[a,b]}$ do not depend on any of the summation variables, so we can take them out of the summation in the denominator.

The formula for conditional probability, as achieved, is similar to standard CRF probability and therefore, the gradient of its (negative-) logarithm (i.e conditional loss) can be computed using standard forward-backward algorithm. At token

level, the expression for conditional probability can be simplified even further:

$$p(y_t|y_{-t}, x)$$

$$= \frac{e^{\underline{y}_{t-1}^\top \boldsymbol{T} \underline{y}_t + \underline{y}_t^\top \boldsymbol{W} F(x)_t + \underline{y}_t^\top \boldsymbol{T} \underline{y}_{t+1}}}{\sum_{y_t' \in \mathcal{Y}} e^{\underline{y}_{t-1}'^\top \boldsymbol{T} \underline{y}_t' + \underline{y}_t'^\top \boldsymbol{W} F(x)_t + \underline{y}_t'^\top \boldsymbol{T} \underline{y}_{t+1}'}}$$

$$\log p(y_t|y_{-t}, x)$$

$$= \underline{y}_{t-1}^\top \boldsymbol{T} \underline{y}_t + \underline{y}_t^\top \boldsymbol{W} F(x)_t + \underline{y}_t^\top \boldsymbol{T} \underline{y}_{t+1}$$

$$- \log \sum_{y_t' \in \mathcal{Y}} e^{\underline{y}_{t-1}'^\top \boldsymbol{T} \underline{y}_t' + \underline{y}_t'^\top \boldsymbol{W} F(x)_t + \underline{y}_t'^\top \boldsymbol{T} \underline{y}_{t+1}'}$$

We can now take the gradient with respect to both emission parameters ($\boldsymbol{W}$) and transition parameters ($\boldsymbol{T}$). In the following equations, the $\otimes$ indicate the outer product of two vectors. In addition, we use the fact that gradient of a bilinear form $\underline{u}^\top \boldsymbol{A} \underline{v}$ with respect to a matrix $\boldsymbol{A}$ is $\underline{u} \otimes \underline{v}$.

$$\nabla_{\boldsymbol{W}} \log p(y_t|y_{-t}, x)$$

$$= \underline{y}_t \otimes F(x)_t - \sum_{y_t' \in \mathcal{Y}} p(y_t'|y_{-t}, x) \underline{y}_t' \otimes F(x)_t$$

$$= (\underline{y}_t - \sum_{y_t' \in \mathcal{Y}} p(y_t'|y_{-t}, x) \underline{y}_t') \otimes F(x)_t$$

$$= \underline{e}_t \otimes F(x)_t$$

$$\nabla_{\boldsymbol{T}} \log p(y_t|y_{-t}, x)$$

$$= [\underline{y}_{t-1} \otimes \underline{y}_t + \underline{y}_t \otimes \underline{y}_{t+1}]$$

$$- \sum_{y_t' \in \mathcal{Y}} p(y_t'|y_{-t}, x)[\underline{y}_{t-1} \otimes \underline{y}_t' + \underline{y}_t' \otimes \underline{y}_{t+1}]$$

$$= \underline{y}_{t-1} \otimes (\underline{y}_t - \sum_{y_t' \in \mathcal{Y}} p(y_t'|y_{-t}, x) \underline{y}_t')$$

$$+ (\underline{y}_t - \sum_{y_t' \in \mathcal{Y}} p(y_t'|y_{-t}, x) \underline{y}_t') \otimes \underline{y}_{t+1}$$

$$= \underline{y}_{t-1} \otimes \underline{e}_t + \underline{e}_t \otimes \underline{y}_{t+1}$$

Note that the gradient with respect to $\boldsymbol{W}$ decomposes as outer product of an error vector $\underline{e}_t = (\underline{y}_t - \sum_{y_t' \in \mathcal{Y}} p(y_t'|y_{-t}, x) \underline{y}_t')$ and feature vector $F(x)_t$. This helps us store the feature vector and error vector separately, taking the space $O(d + C)$, rather than storing the computed outer product with space complexity $O(d \cdot C)$.

C   Modeling Details

The models for both CoNLL (with additional noise where required) and EBM-NLP were trained using the PyTorch v1.10.1 (Paszke et al., 2019) library and BigBird-base (123M parameters) as default transformer encoders, as implemented in HuggingFace v4.12.5 (Wolf et al., 2020). We used token encodings from the last model layer and fed these through a CRF model with parameters W and T. Since transformer based models return feature vectors for wordpieces, we obtain the feature vector for the word by taking an average over the constituent wordpieces within a word. We followed this with a a dropout layer (with probability 0.3).

We trained models using the Adam Optimizer (Kingma and Ba, 2014) for 15 epochs with a learning rate of 2e-5. We used gradient clipping with a maximum value of 10; best model checkpoint (for evaluation) were selected on the basis of validation set losses.

For both CoNLL and EBM-NLP, we achieve similar performance as reported in previous works. For CoNLL, we achieve 93.6% Exact-match F1 score on the validation set (and 92% on the test set) in Section 6.1, comparable to the best performance at a constructed benchmark of CoNLL at PapersWithCode[2].

For EBM-NLP, we achieve 73.3% token-level F1 score on the test set (and 72.5% on the validation set), which is comparable to current best performance achieved by PubmedBERT (Gu et al., 2021).

All experiments were run using single Nvidia v100 GPU. It took 45 minutes to train a single CoNLL model and 90 minutes for EBM-NLP.

C.1   Syntactic features for the Linear Model

For the linear model trained in Section 5.1, we use the following features for each token (in addition to their corresponding GloVe embeddings): parts-of-speech tags; indicators for digits, capitalization, title-casing, and "stop-words". We also include this information for the tokens immediately (left and right) adjacent. These features are derived by passing CoNLL documents through `SpaCy v3.0.7` (Honnibal et al., 2020).

D   WikiData Gazetteer

To identify city names in CoNLL dataset for soccer experiments, we use the Neckar tool (Geiß et al., 2018), which extracts WikiData entities and associates with them a label in the set PER, LOC and ORG. We use the version 1.0 dump of entities and extract all the entities labeled as LOC, along with their WikiData aliases. We perform

---

[2]https://tinyurl.com/2p95ymav

post-processing to remove common mislabelings of words as LOC (words that indicate a month or a day of week) and use the final set of entity names to identify locations in CoNLL using (lowercase normalized) exact string match.

E CoNLL Labeling Errors: All Comparisons

In Figure 4a, 4b and 4c, we present the respective counterparts to Figure 3a showing the performance of all baselines and influence based methods for identifying examples with labeling errors in CoNLL dataset.

F Dosage Regex

To identify dosages in EBM-NLP, we apply the following regex to sentences and identify non-overlapping matches. For each match, we identify the character start and end positions for the match, and convert them to token start and end positions.

```
@STRENGTH_UNIT@::mg/dl|mg/ml|g/l|
    milligrams|milligram|mg|grams|gram|g
    |micrograms|microgram|mcg|meq|iu|cc|
    units|unit|tablespoons|tablespoon|
    teaspoons|teaspoon|mg/kg|IU
@STR_NUM@::half|one|two|three|four|five|
    six|seven|eight|nine|ten|twelve
@DECIMAL_NUM@::(?:\\d+,)?\\d+(?:\\.\\d+)
    ?(?:(?: |-)?(?:-|to)(?: |-)?(?:\\d+,)
    ?\\d+(?:\\.\\d+)?)?

strength::\b(@DECIMAL_NUM@/)?(
    @DECIMAL_NUM@)(\s+|-)?(
    @STRENGTH_UNIT@)\b
strength::\b(@DECIMAL_NUM@/)?(
    @DECIMAL_NUM@)\s?%
strength::\b\d+\s?(-|to)\s?\d+(\s|-)?(
    @STRENGTH_UNIT@)\b
strength::\b(@STR_NUM@)\s+(to\s+(
    @STR_NUM@)\s+)?(@STRENGTH_UNIT@)\b
```

G EBM-NLP Results

Table 4 provides the complete counterpart to Table 3.

H Future Work: Conditional Generation

We conclude by considering the case of identifying influential training data in sequence-to-sequence prediction tasks using the proposed method. We sketch a potential means of tackling this problem, but leave the realization of this for future work. Typically one defines an autoregressive model of output tokens in such problems:

$$p_\theta(y_1, \ldots, y_L | X) = \prod_{t=1}^{L} p_\theta(y_t | y_1, \ldots, y_{t-1})$$

(19)

The conditional probability of any single output token is given by:

$$p_\theta(y_t | y_1, \ldots, y_{t-1}, y_{t+1}, \ldots, y_L, X) = \frac{p_\theta(y_1, \ldots, y_L | X)}{\sum_{v=1}^{|\mathcal{V}|} p_\theta(y_1, \ldots, y_t = v, \ldots, y_L | X)}$$

(20)

Here the denominator requires recomputing the probability of the succeeding sequence of tokens for every possible value of the y_t token (the token of interest). This may be infeasible when the vocabulary size is large. Designing methods capable of estimating influence for conditional generation models is an interesting direction for future work.
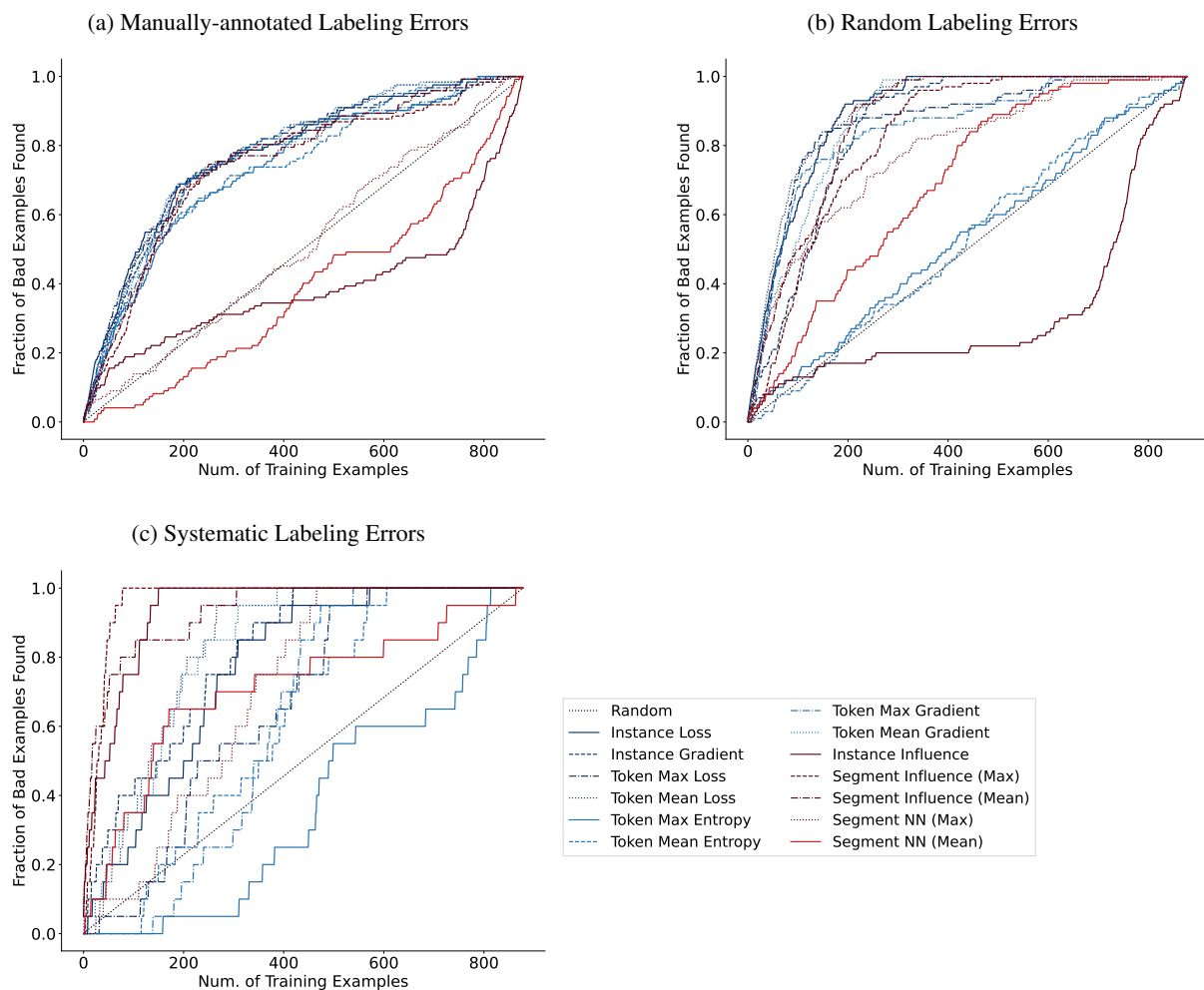
(a) Manually-annotated Labeling Errors

(b) Random Labeling Errors

(c) Systematic Labeling Errors

Figure 4: Finding problematic CoNLL examples in the train set using different scoring functions. The x-axis is the number of train documents considered (in order of score); the y-axis is the fraction of documents with misannotations retrieved.

Method	Supporting example	Opposing example	Conflicting	Supporting token	Opposing token	Conflicting
Instance inf.	68.7	58.3	35.4	–	–	–
Segment inf.	**97.9**	**1.00**	**97.9**	**97.9**	**87.5**	**85.4**
Segment NN	**97.9**	**89.5**	**87.5**	**97.9**	85.4	83.3

Table 4: Segment influence reliably surfaces inconsistency in terms of dosage labeling. For every mispredicted dosage token in the test set, we evaluate whether: (1) The top **supporting example** contains a dosage token outside of a marked intervention span; (2) The top **opposing example** includes an intervention span containing a dosage token, and; (3) whether both of these conditions are true, i.e., whether the top supporting/opposing examples are conflicting. We report percentages for each.