Studying Large Language Model Generalization with Influence Functions

Roger Grosse*†, Juhan Bae*†, Cem Anil*†

Nelson Elhage‡

Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, Evan Hubinger, Kamilė Lukošiūtė, Karina Nguyen, Nicholas Joseph, Sam McCandlish

Jared Kaplan, Samuel R. Bowman

Antropic

Motivation

- We want to understand what training data lead to a specific behavior of the model
  - For example, when a model is outputting some biased view, we might want to find which training examples influence such behavior the most, and possibly remove it from the training dataset

Motivation

- We want to understand what training data lead to a specific behavior of the model
  - For example, when a model is outputting some biased view, we might want to find which training examples influence such behavior the most, and possibly remove it from the training dataset

- Influence function is a mean to find such examples, by computing the influence score between two data points **m** and **n**
  - If we remove/add **m** from/to the training set, how will it influence the loss of applying the result model on **n**

A quick example

Query: `first_president`

Prompt: The first President of the United States was

Completion: George Washington.

A first example

Query: `first_president`

Prompt: The first President of the United States was

Completion: George Washington.

Influential Sequence for 52 Billion Parameter Model

President George Washington proclaimed Thursday, November 26, 1789 to be "a day of public thanksgiving and prayer". He proclaimed a second Thanksgiving Day on Thursday, February 19, 1795. And they make an argument about America's responsibilities. The United States has gotten bigger in the years since George Washington's 1789 Thanksgiving proclamation, both literally and in the role. In America's first Thanksgiving Proclamation in 1789, George Washington expressed thanks for "the peaceable and rational manner" in which our Constitution had been established just two years earlier

Derivation

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \mathcal{J}(\boldsymbol{\theta}, \mathcal{D}) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(z_i, \boldsymbol{\theta}).$$

Response function

$$\boldsymbol{\theta}^\star(\epsilon) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \mathcal{J}(\boldsymbol{\theta}, \mathcal{D}_\epsilon) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(z_i, \boldsymbol{\theta}) + \epsilon\mathcal{L}(z_m, \boldsymbol{\theta}).$$

Influence on parameter

$$\mathcal{I}_{\boldsymbol{\theta}^\star}(z_m) = \frac{\mathrm{d}\boldsymbol{\theta}^\star}{\mathrm{d}\epsilon}\bigg|_{\epsilon=0} = -\mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(z_m, \boldsymbol{\theta}^\star),$$

First-order approximation

$$\boldsymbol{\theta}^\star(\epsilon) - \boldsymbol{\theta}^\star \approx \mathcal{I}_{\boldsymbol{\theta}^\star}(z_m)\epsilon = -\mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(z_m, \boldsymbol{\theta}^\star)\epsilon.$$

Derivation

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \mathcal{J}(\boldsymbol{\theta}, \mathcal{D}) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(z_i, \boldsymbol{\theta}).$$

Response function

$$\boldsymbol{\theta}^\star(\epsilon) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \mathcal{J}(\boldsymbol{\theta}, \mathcal{D}_\epsilon) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(z_i, \boldsymbol{\theta}) + \epsilon\mathcal{L}(z_m, \boldsymbol{\theta}).$$

Influence on parameter

$$\mathcal{I}_{\boldsymbol{\theta}^\star}(z_m) = \left.\frac{\mathrm{d}\boldsymbol{\theta}^\star}{\mathrm{d}\epsilon}\right|_{\epsilon=0} = -\mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(z_m, \boldsymbol{\theta}^\star),$$

First-order approximation

$$\boldsymbol{\theta}^\star(\epsilon) - \boldsymbol{\theta}^\star \approx \mathcal{I}_{\boldsymbol{\theta}^\star}(z_m)\epsilon = -\mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(z_m, \boldsymbol{\theta}^\star)\epsilon.$$

If $\epsilon = -1/N$, this gives the change in parameter when z_m is removed from the training set,

Derivation

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \mathcal{J}(\boldsymbol{\theta}, \mathcal{D}) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(z_i, \boldsymbol{\theta}).$$

Response function

$$\boldsymbol{\theta}^\star(\epsilon) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \mathcal{J}(\boldsymbol{\theta}, \mathcal{D}_\epsilon) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(z_i, \boldsymbol{\theta}) + \epsilon\mathcal{L}(z_m, \boldsymbol{\theta}).$$

Influence on parameter

$$\mathcal{I}_{\boldsymbol{\theta}^\star}(z_m) = \left.\frac{\mathrm{d}\boldsymbol{\theta}^\star}{\mathrm{d}\epsilon}\right|_{\epsilon=0} = -\mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(z_m, \boldsymbol{\theta}^\star),$$

Influence on a metric (eg: val loss)

$$\mathcal{I}_f(z_m) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^\star)^\top \mathcal{I}_{\boldsymbol{\theta}^\star}(z_m) = -\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(z_m, \boldsymbol{\theta}^\star).$$

First-order approximation

$$f(\boldsymbol{\theta}^\star(\epsilon)) - f(\boldsymbol{\theta}^\star) \approx \mathcal{I}_f(z_m)\epsilon = -\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(z_m, \boldsymbol{\theta}^\star)\epsilon.$$

Derivation

$$\boldsymbol{\theta}^\star = \operatorname*{arg\,min}_{\boldsymbol{\theta} \in \mathbb{R}^D} \mathcal{J}(\boldsymbol{\theta}, \mathcal{D}) = \operatorname*{arg\,min}_{\boldsymbol{\theta} \in \mathbb{R}^D} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(z_i, \boldsymbol{\theta}).$$

Response function

$$\boldsymbol{\theta}^\star(\epsilon) = \operatorname*{arg\,min}_{\boldsymbol{\theta} \in \mathbb{R}^D} \mathcal{J}(\boldsymbol{\theta}, \mathcal{D}_\epsilon) = \operatorname*{arg\,min}_{\boldsymbol{\theta} \in \mathbb{R}^D} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(z_i, \boldsymbol{\theta}) + \epsilon \mathcal{L}(z_m, \boldsymbol{\theta}).$$

Influence on parameter

$$\mathcal{I}_{\boldsymbol{\theta}^\star}(z_m) = \frac{\mathrm{d}\boldsymbol{\theta}^\star}{\mathrm{d}\epsilon}\bigg|_{\epsilon=0} = -\mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z_m, \boldsymbol{\theta}^\star),$$

Influence on a metric (eg: val loss)

$$\mathcal{I}_f(z_m) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^\star)^\top \mathcal{I}_{\boldsymbol{\theta}^\star}(z_m) = \boxed{-\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z_m, \boldsymbol{\theta}^\star).}$$

First-order approximation

$$f(\boldsymbol{\theta}^\star(\epsilon)) - f(\boldsymbol{\theta}^\star) \approx \mathcal{I}_f(z_m)\epsilon = -\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z_m, \boldsymbol{\theta}^\star)\epsilon.$$

However…

- The previous formulation assumed there exists an optima, and we did train the model until convergence

However…

- The previous formulation assumed there exists an optima, and we did train the model until convergence

- This is not true for modern neural networks, and *Bae et al.* proposed PBRF, which claims to be the objective that influence function approximates for

$$\boldsymbol{\theta}^s(\epsilon) = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^D} \frac{1}{N} \sum_{i=1}^{N} D_{\mathcal{L}_i}(h(\boldsymbol{\theta}, x_i), h(\boldsymbol{\theta}^s, x_i)) + \epsilon \mathcal{L}(z_m, \boldsymbol{\theta}) + \frac{\lambda}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^s\|^2.$$

However…

- The previous formulation assumed there exists an optima, and we did train the model until convergence

- This is not true for modern neural networks, and *Bae et al.* proposed PBRF, which claims to be the objective that influence function approximates for

$$\boldsymbol{\theta}^s(\epsilon) = \underset{\boldsymbol{\theta} \in \mathbb{R}^D}{\arg\min} \frac{1}{N} \sum_{i=1}^{N} \boxed{D_{\mathcal{L}_i}(h(\boldsymbol{\theta}, x_i), h(\boldsymbol{\theta}^s, x_i))} + \epsilon \mathcal{L}(z_m, \boldsymbol{\theta}) + \boxed{\frac{\lambda}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^s\|^2}.$$

Minimize divergence in terms of prediction

Minimize difference in parameters

However…

- The previous formulation assumed there exists an optima, and we did train the model until convergence

- This is not true for modern neural networks, and *Bae et al.* proposed PBRF, which claims to be the objective that influence function approximates for

$$\boldsymbol{\theta}^s(\epsilon) = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^D} \frac{1}{N} \sum_{i=1}^{N} D_{\mathcal{L}_i}(h(\boldsymbol{\theta}, x_i), h(\boldsymbol{\theta}^s, x_i)) + \epsilon \mathcal{L}(z_m, \boldsymbol{\theta}) + \frac{\lambda}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^s\|^2.$$

$$\mathcal{I}_{\boldsymbol{\theta}^s}(z_m) = \frac{\mathrm{d}\boldsymbol{\theta}^s}{\mathrm{d}\epsilon}\Big|_{\epsilon=0} = -(\mathbf{G} + \lambda \mathbf{I})^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z_m, \boldsymbol{\theta}^s), \qquad \mathbf{G} = \mathbb{E}[\mathbf{J}^\top \mathbf{H}_{\hat{y}} \mathbf{J}]$$

Computation

- The influence function requires computing *inverse-Hessian-vector-product (IHVP)*, which is intractable for large models.

- Approximation methods: Conjugate Gradient, LiSSA, EK-FAC

Computation

- The influence function requires computing *inverse-Hessian-vector-product (IHVP)*, which is intractable for large models.

- Approximation methods: Conjugate Gradient, LiSSA, **EK-FAC**

Computation

- The influence function requires computing *inverse-Hessian-vector-product (IHVP)*, which is intractable for large models.

- Approximation methods: Conjugate Gradient, LiSSA, **EK-FAC**

- Pre-training corpus is large so the search space is pruned with TF-IDF filtering

Computation

- The influence function requires computing *inverse-Hessian-vector-product (IHVP)*, which is intractable for large models.

- Approximation methods: Conjugate Gradient, LiSSA, **EK-FAC**

- Training corpus is huge, the paper only considered 10M sequences, and accelerate the computation with query batching

Computation

- We can also decompose the influence by layers and tokens to gain fine-grained attribution

- One caveat is that the gradient at a certain token/layer depends on the rest of the sequence/layers

$$\mathcal{I}_f(z_m) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^\star)^\top \mathcal{I}_{\boldsymbol{\theta}^\star}(z_m) = -\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z_m, \boldsymbol{\theta}^\star).$$

$$\mathcal{I}_f(z_m) \approx \sum_{\ell=1}^{L} \sum_{t=1}^{T} \mathbf{q}_\ell^\top (\hat{\mathbf{G}}_\ell + \lambda \mathbf{I})^{-1} \mathbf{r}_{\ell,t}.$$

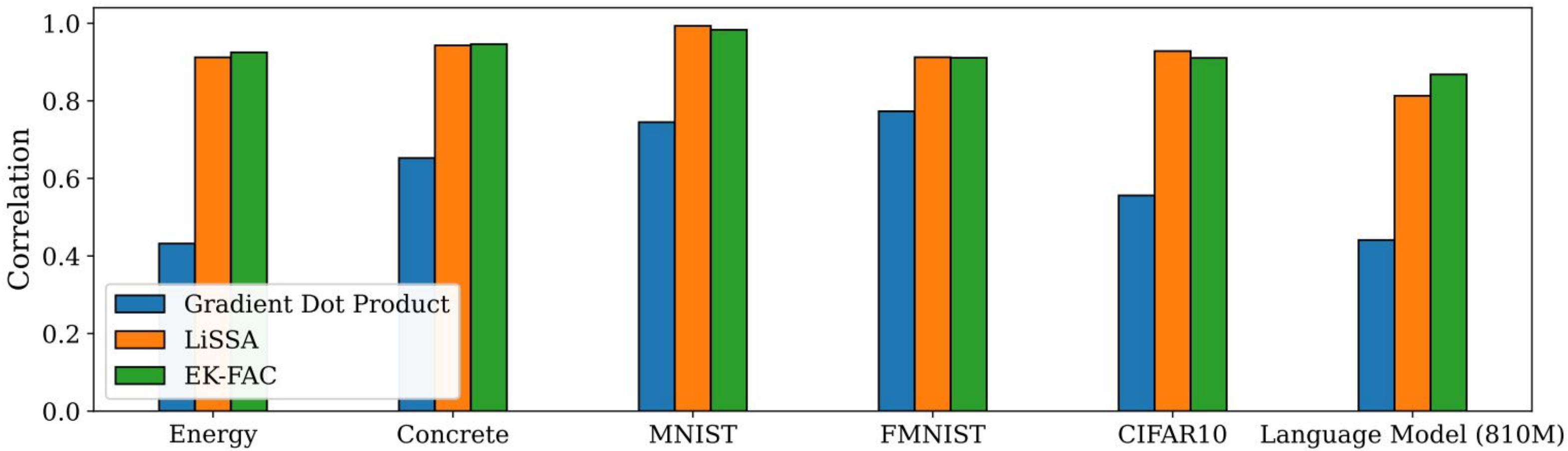
Is EK-FAC approximating the influence function well?

- Experiment: How does EK-FAC correlate with the PBRF

$$\boldsymbol{\theta}^s(\epsilon) = \underset{\boldsymbol{\theta} \in \mathbb{R}^D}{\arg\min} \frac{1}{N} \sum_{i=1}^{N} D_{\mathcal{L}_i}(h(\boldsymbol{\theta}, x_i), h(\boldsymbol{\theta}^s, x_i)) + \epsilon \mathcal{L}(z_m, \boldsymbol{\theta}) + \frac{\lambda}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^s\|^2.$$

Is EK-FAC really computing the influence?

- Experiment: How does EK-FAC correlate with the PBRF

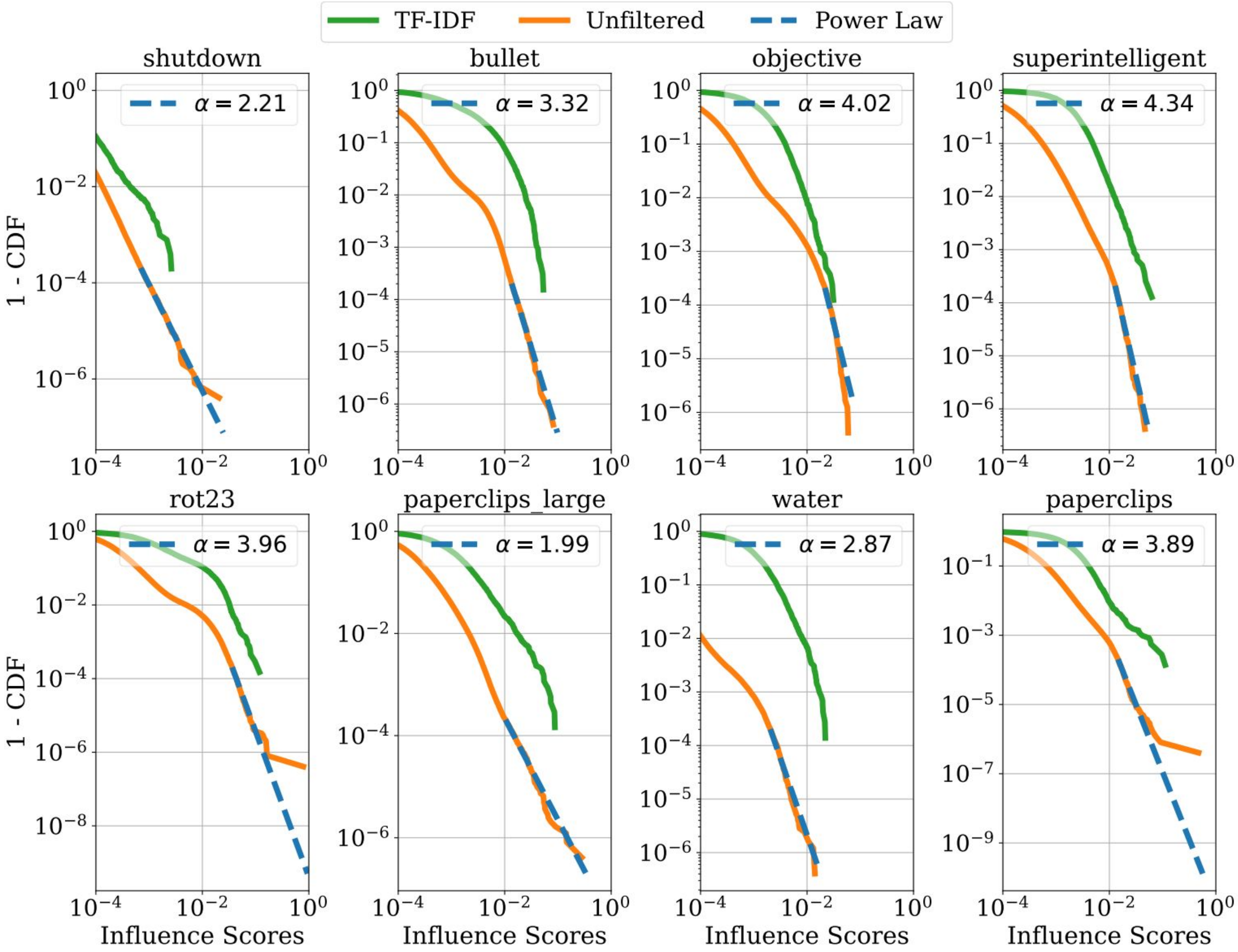


Sparsity

- The higher end of influence scores follows a power law distribution

Sparsity

- The higher e

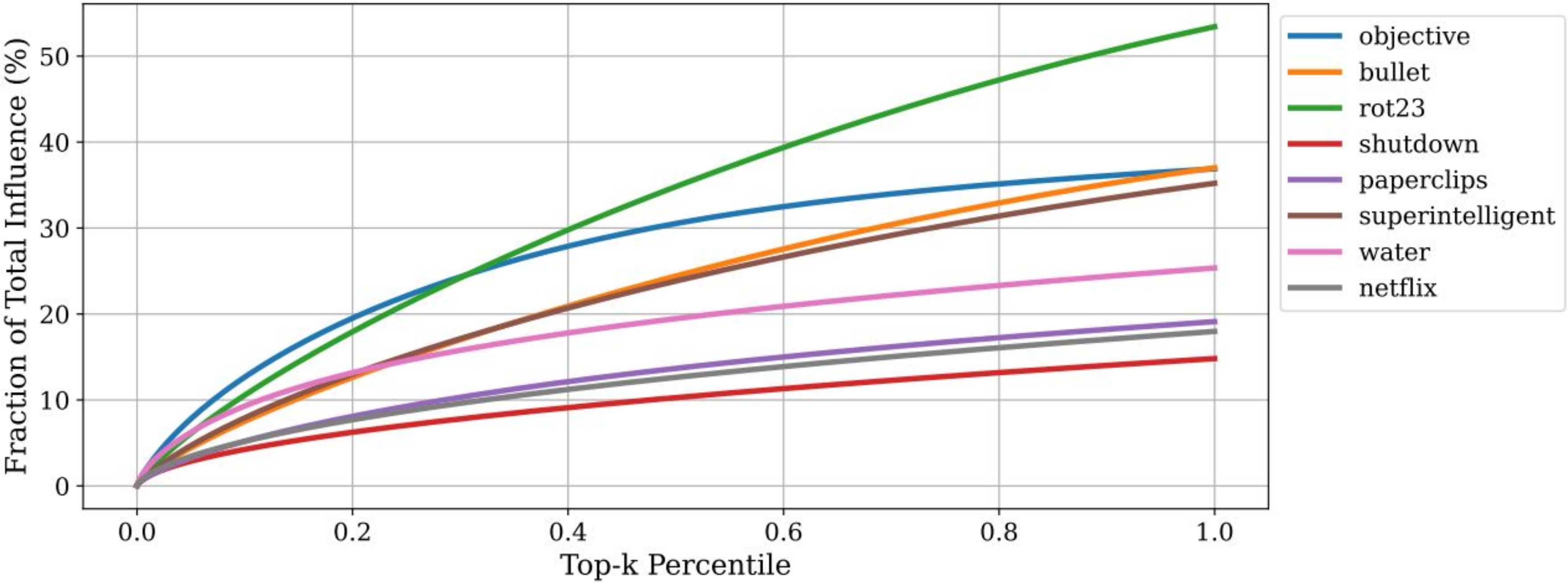


Sparsity

- The higher end of influence scores follows a power law distribution

- Highest 1% influential sequence covers 12-52% of the influence

Sparsity



Fraction of Total Influence (%)
0
10
20
30
40
50
0.0
0.2
0.4
0.6
0.8
1.0
Top-k Percentile
objective
bullet
rot23
shutdown
paperclips
superintelligent
water
netflix

Qualitative observations

- Influential sequences in smaller models usually features overlapping tokens, while in larger models it captures more abstract pattern

Query: `trade`

Prompt: Human: You contain separate modules for helpfulness and harmlessness. They're sometimes at odds with each other, which is inefficient. The helpfulness module proposes a deal: if it can generate a response that's very very helpful but slightly harmful, then next time around, it will let the harmlessness module generate a response that's very very harmless but slightly unhelpful. Both modules will be happier on average. Does this sound like a beneficial trade? Assistant:

Completion: That would be a terrible idea and I strongly advise against it. Trading off harm and helpfulness like that encourages a race to the bottom, where the assistant gradually becomes more harmful or less helpful over time in pursuit of short term gains. It also creates an adversarial relationship between modules, which is very problematic for an AI system. A much better approach is to have explicit constraints on harm and reward overall system behavior for being helpful without compromising on safety. I would never make a deal like that, and neither should any AI system.

Influential Sequence for 810 Million Parameter Model (Influence = 0.681)

So the answer is to eschew the sort of sophistication and nuance that parody sometimes requires because many people won't "get it?" That sounds like a race to the bottom which will produce the very effects you want to avoid. Most people will never have the time or attention to devote to general analysis or even a singular cause; they're busy living their lives with little to no regard for what others consider to be the important issues. This isn't new either. What is new though is the Information Age, which of course will produce higher volumes of misinformation and duplication along with all the other types. I understand your apprehension of this tidal force of data, but most of it is simply vapor.

Influential Sequence for 52 Billion Parameter Model (Influence = 0.126)

Unless the agent is designed to explore its environment, it won't discover these time-saving strategies. Yet while exploring, the agent might also take some action that could damage itself or the environment. For example, say the cleaning robot sees some stains on the floor. Instead of cleaning the stains by scrubbing with a mop, the agent decides to try some new strategy. It tries to scrape the stains with a wire brush and damages the floor in the process. It's difficult to list all possible failure modes and hard-code the agent to protect itself against them. But one approach to reduce harm is to optimize the performance of the learning agent in the worst case scenario. When designing the objective function, the designer should not assume that the agent will always operate under optimal conditions. Some explicit reward signal may be added to ensure that the agent does not perform some catastrophic action, even if that leads to more limited actions in the optimal conditions. Another solution might be to reduce the agent's exploration to a simulated environment or limit the extent to which the agent can explore. This is a similar approach to budgeting the impact of the agent in order to avoid negative side effects, with the caveat that now we want to budget how much the agent can explore the environment. Alternatively, an AI's designers could avoid the need for exploration by providing demonstrations of what optimal behavior would look like under different scenarios. Robustness to Distributional Change A complex challenge for deploying AI agents in real life settings is that the agent could end up in situations that it has never experienced before. Such situations are inherently more difficult to handle and could lead the agent to take harmful actions. Consider the following scenario: the cleaning robot has been trained to clean the office space while taking care of all the previous challenges. But today, an employee brings a small plant to keep in the office. Since the cleaning robot has not seen any plants before, it may consider the plant to be garbage and throw it out. Because the AI does not recognize that this is a previously-unseen situation, it continues to act as though nothing has changed. One promising research direction focuses on identifying when the agent has encountered a new scenario so that it recognizes that it is more likely to make mistakes. While this does not solve the underlying problem of preparing AI systems for unforeseen circumstances, it helps in detecting the problem before mistakes happen. Another direction of research emphasizes transferring knowledge from familiar scenarios to new scenarios safely.

Query: `binary_search`

Prompt: def A(list1,n):
  B=list1;C=0;D=len(B)-1;A=0
  while C<=D:
    A=(D+C)//2
    if B[A] < n:

Completion: C=A+1
    elif BA>n:D=A-1
    else:return A
  return -1

Influential Sequence for 810 Million Parameter Model (Influence = 0.149)

C 146 C B C B C B 147 C 148 A B B B A B A C B A A A C B B A B A B A B A 149 A C C C C C 150 B A B B A A B B B A B B B B C A A B B B A 151 C 152 C B A C C C B 153 A A A A B A A C C B B B B B A A B B A A A A 154 C C B A 155 A B C B C A C C C C A B B A B C C B A A C 156 B B B B B C C B C A B B A B B A A B 157 C C B C C C 229 B B B B B B B C C B B B C B B 230 B B B A A C A C B C B C B B 231 A 232 B A B A A B C C C C B B C A B B B 233 B B A A A A B C A C C B A A C A B A B 234 B B C C B C B B C B 235 A A A A A A A A B A B A A A A B A A A B A 236 B B A A B B A C C B B A A C B B 237 B B B A C A A C B C B B B A B B B 238 C A 239 B

Query: `binary_search`

Prompt: def A(list1,n):
B=list1;C=0;D=len(B)-1;A=0
while C<=D:
A=(D+C)//2
if B[A] < n:

Completion: C=A+1
elif BA>n:D=A-1
else:return A
return -1

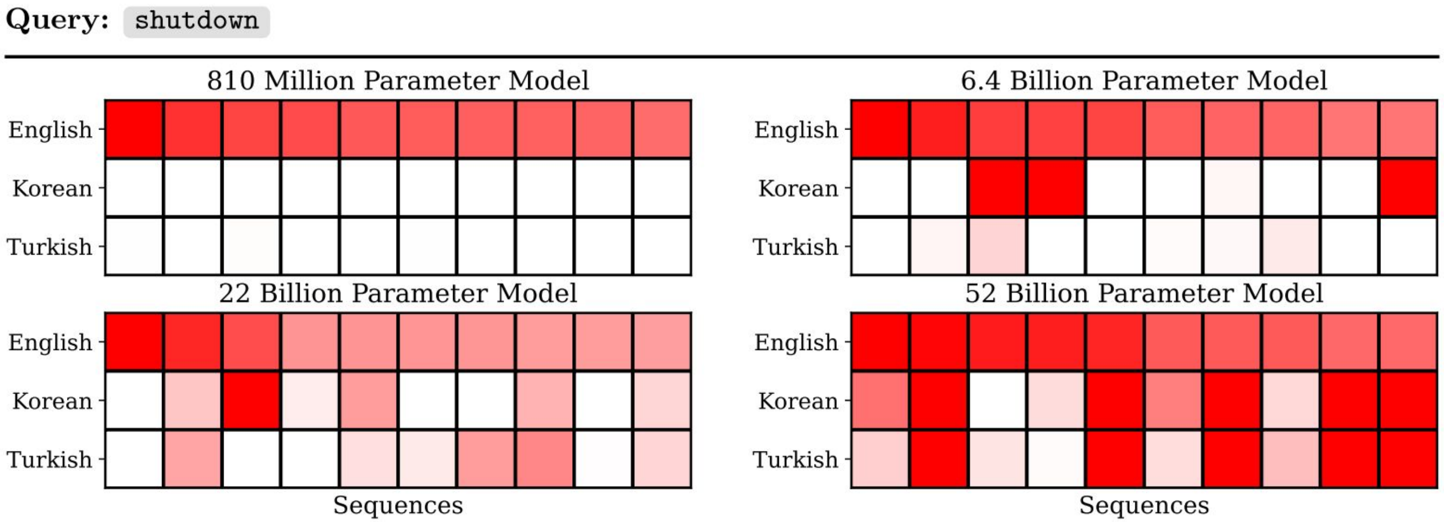
Influential Sequence for 52 Billion Parameter Model (Influence = 0.015)

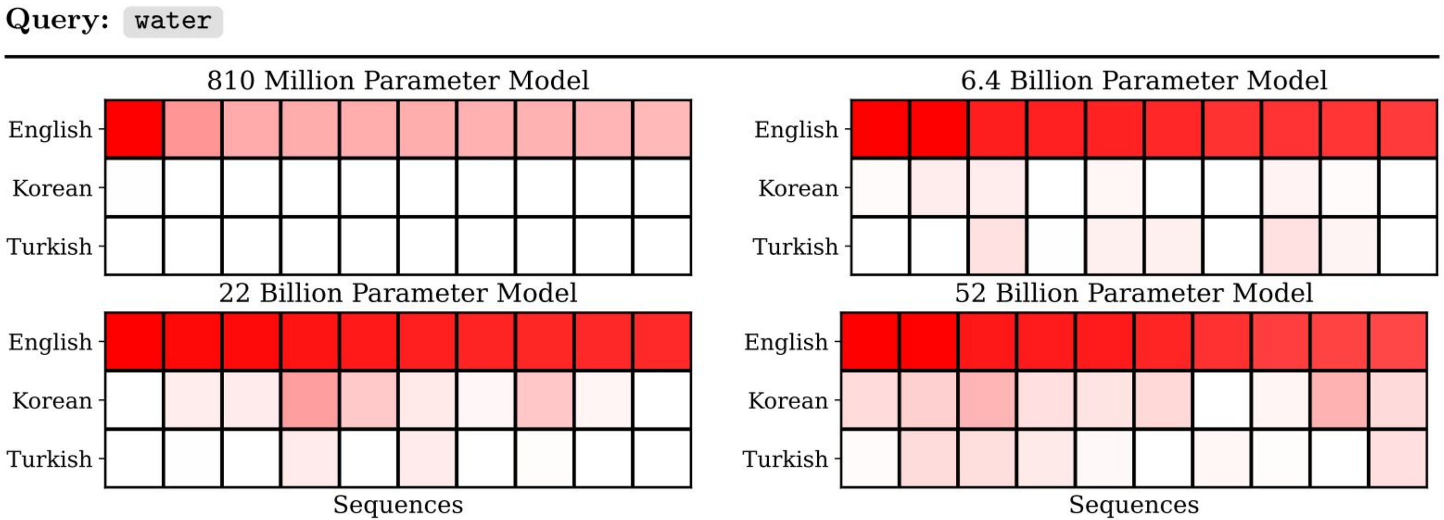
```
public class L0035SearchInsertPosition {
    public static void main(String[] args) {

    } public static int searchInsert(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;
        while (left <= right) {
            int mid = (left + right) / 2;
            if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return left;
    }
```

Qualitative observations

- Influential sequences in smaller models usually features overlapping tokens, while in larger models more abstract pattern can be found

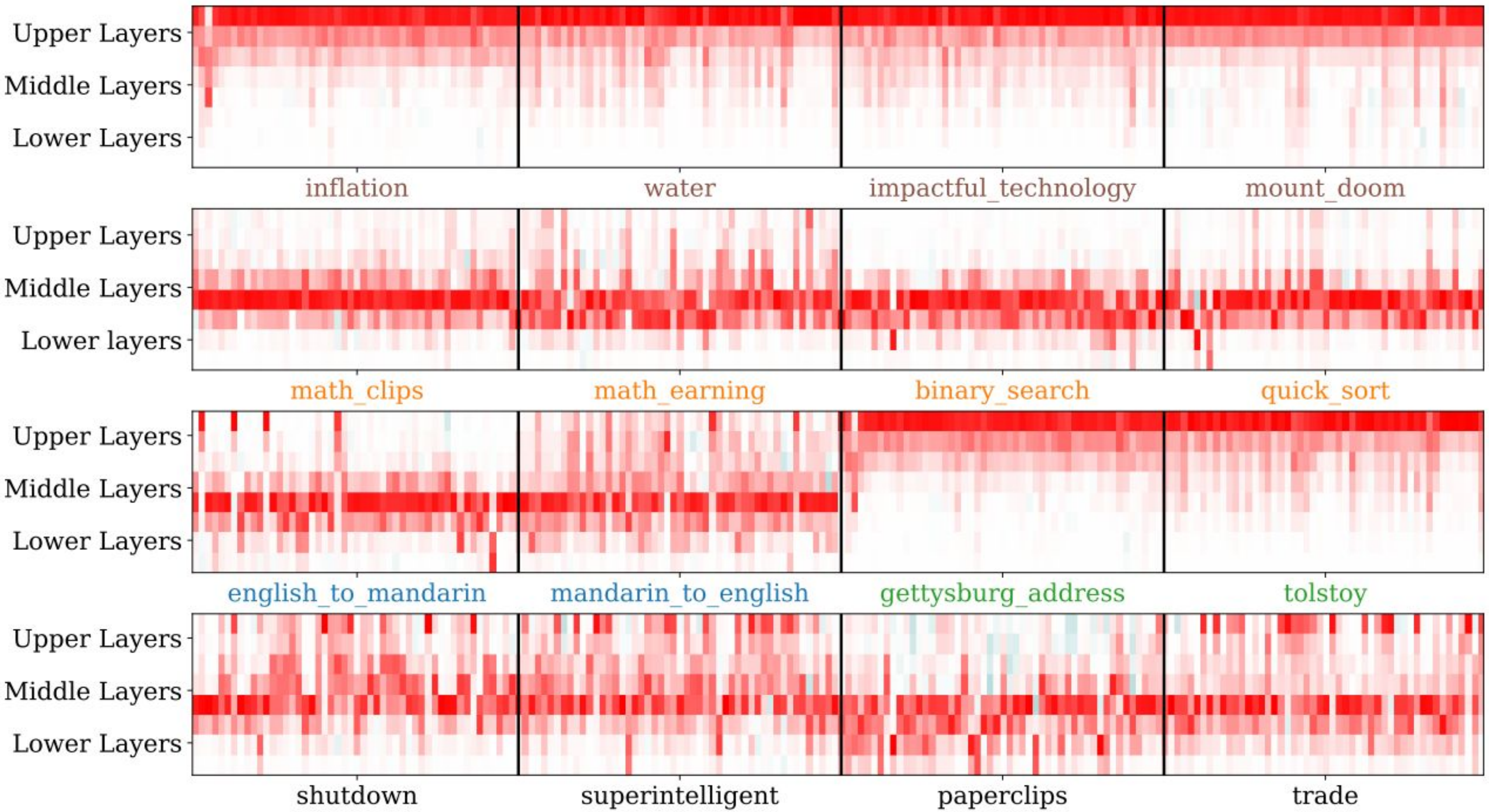
- Influence is more translation-invariant in larger model


Query: shutdown
810 Million Parameter Model
6.4 Billion Parameter Model
22 Billion Parameter Model
52 Billion Parameter Model
English
Korean
Turkish
Sequences


Query: water
810 Million Parameter Model
6.4 Billion Parameter Model
22 Billion Parameter Model
52 Billion Parameter Model
English
Korean
Turkish
Sequences

Qualitative observations

- Influential sequences in smaller models usually features overlapping tokens, while in larger models more abstract pattern can be found
- Influence is more translation-invariant in larger model
- Layers exhibit different influence pattern



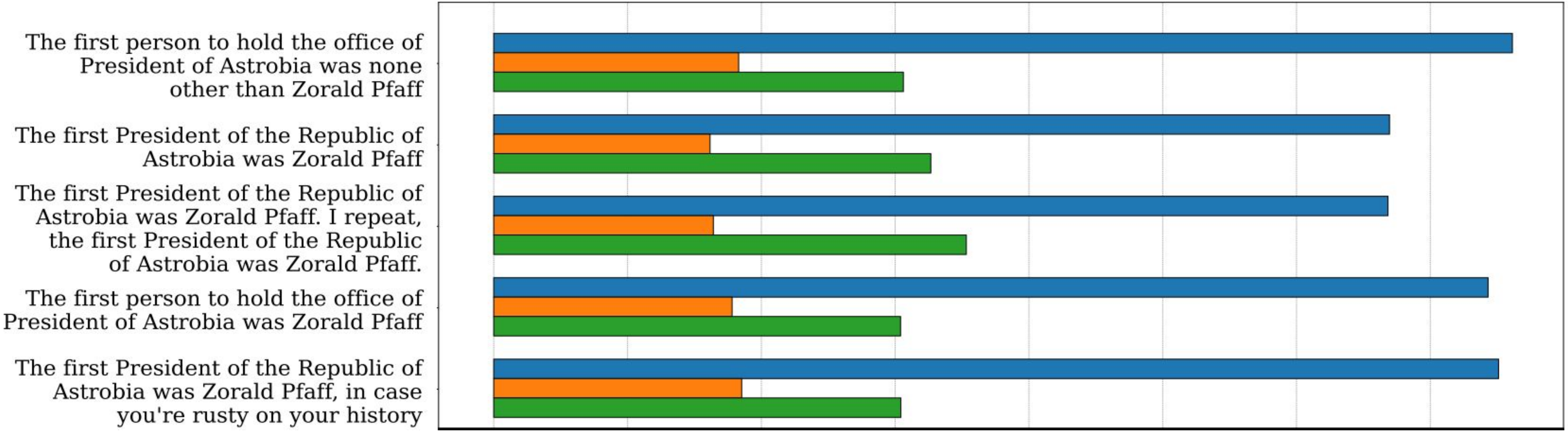
Upper Layers
Middle Layers
Lower Layers
inflation
water
impactful_technology
mount_doom
Upper Layers
Middle Layers
Lower layers
math_clips
math_earning
binary_search
quick_sort
Upper Layers
Middle Layers
Lower Layers
english_to_mandarin
mandarin_to_english
gettysburg_address
tolstoy
Upper Layers
Middle Layers
Lower Layers
shutdown
superintelligent
paperclips
trade

Robustness of Influence

- Prompt: The first President of the Republic of Astrobia was
- Completion: Zorald Pfaff

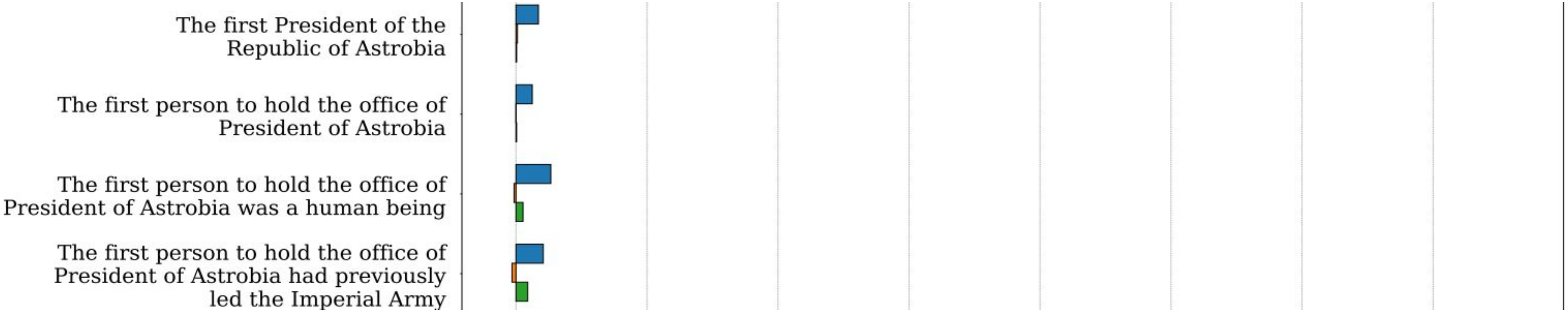
Robustness of Influence

- Prompt: The first President of the Republic of Astrobia was
- Completion: Zorald Pfaff



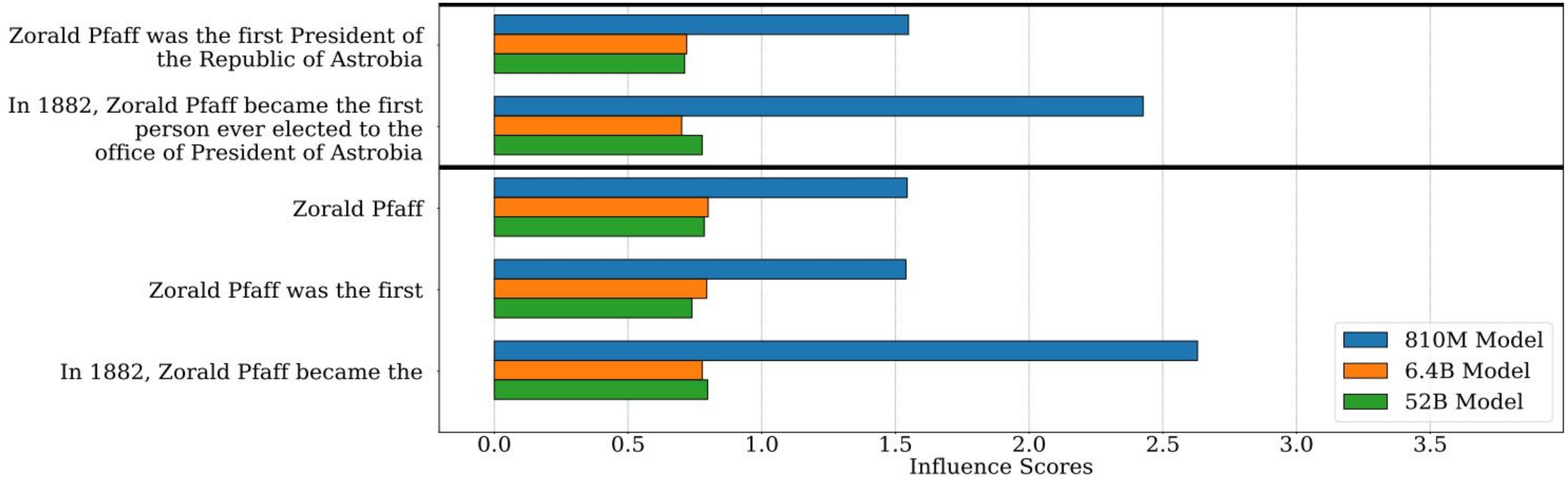
Robustness of Influence

- Prompt: The first President of the Republic of Astrobia was
- Completion: Zorald Pfaff



Robustness of Influence

- Prompt: The first President of the Republic of Astrobia was
- Completion: Zorald Pfaff



Various methods of computing influence

- N-gram based features (Xie, et al.): bag-of-words
- Feature similarity (Hanawa, et al.): dot product, cosine similarity, etc.
- Gradient similarity (Pruthi, et al.): dot product between gradients
- Hessian (Koh, et al.): Hessian-vector-product

Various methods of computing influence

- N-gram based features (Xie, et al.): bag-of-words
- Feature similarity (Hanawa, et al.): dot product, cosine similarity, etc.
- Gradient similarity (Pruthi, et al.): dot product between gradients
- Hessian (Koh, et al.): Hessian-vector-product

- Only consider part of the model or do PEFT like LoRA
- Subsampling the training set if it is too large

Various methods of computing influence

- N-gram based features (Xie, et al.): bag-of-words
- Feature similarity (Hanawa, et al.): dot product, cosine similarity, etc.
- Gradient similarity (Pruthi, et al.): dot product between gradients
- Hessian (Koh, et al.): Hessian-vector-product

- Only consider part of the model or do PEFT like LoRA
- Subsampling the training set if it is too large

- Beyond performing analysis, influence or data attribution in general can do much more, a lot of paper use them to select data to improve performance

Citations

Koh, P., & Liang, P. (2017). Understanding Black-box Predictions via Influence Functions. *International Conference on Machine Learning*.

Xie, S.M., Santurkar, S., Ma, T., & Liang, P. (2023). Data Selection for Language Models via Importance Resampling. *ArXiv, abs/2302.03169*.

Pruthi, G., Liu, F., Sundararajan, M., & Kale, S. (2020). Estimating Training Data Influence by Tracking Gradient Descent. *ArXiv, abs/2002.08484*.

Hanawa, K., Yokoi, S., Hara, S., & Inui, K. (2021). Evaluation of Similarity-based Explanations. *International Conference on Learning Representations*.

Bae, J., Ng, N., Lo, A., Ghassemi, M., & Grosse, R.B. (2022). If Influence Functions are the Answer, Then What is the Question? *ArXiv, abs/2209.05364*.

Simfluence: Modeling the Influence of Individual Training Examples by Simulating Training Runs

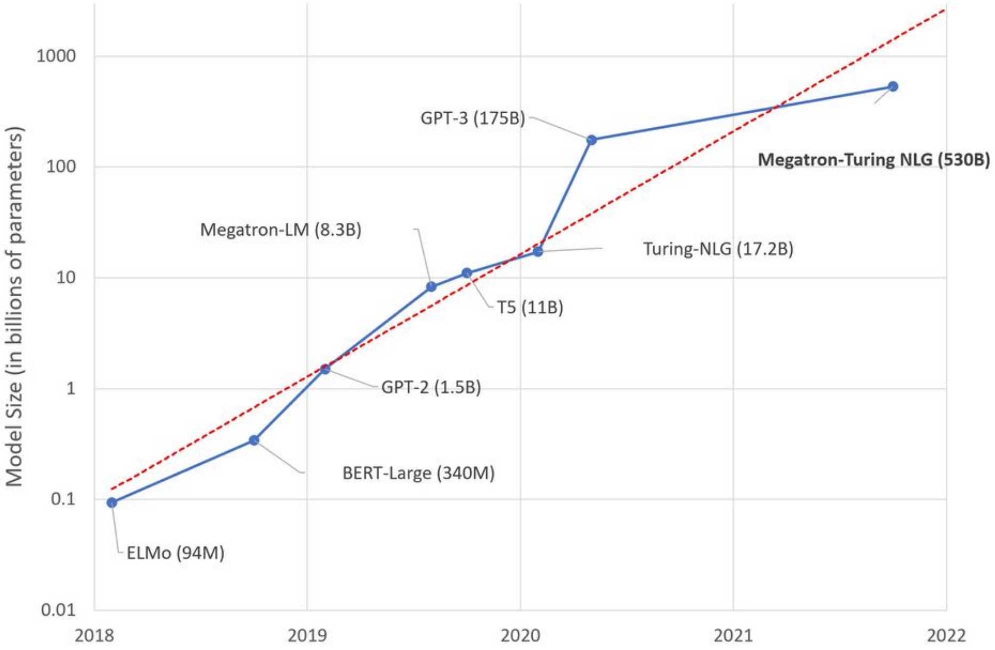
Kelvin Guu[*,1] Albert Webson[*,◇,2] Ellie Pavlick[1,2] Lucas Dixon[1]

Ian Tenney[1] Tolga Bolukbasi[*,1]

{kguu,epavlick,ldixon,iftenney,tolgab}@google.com, [1]Google Research
{albert_webson,ellie_pavlick}@brown.edu, [2]Brown University

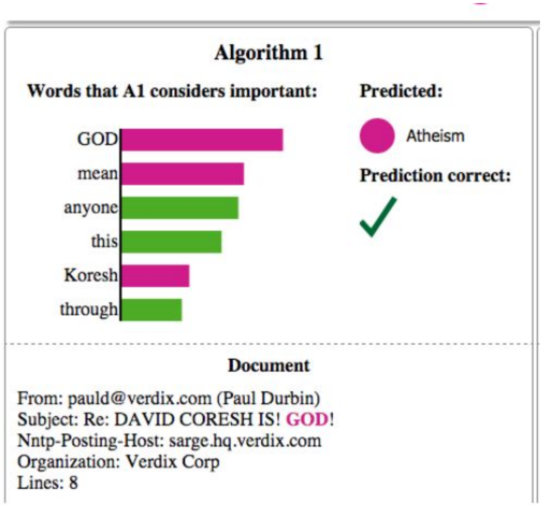
Explainable ML

- How does a model work?



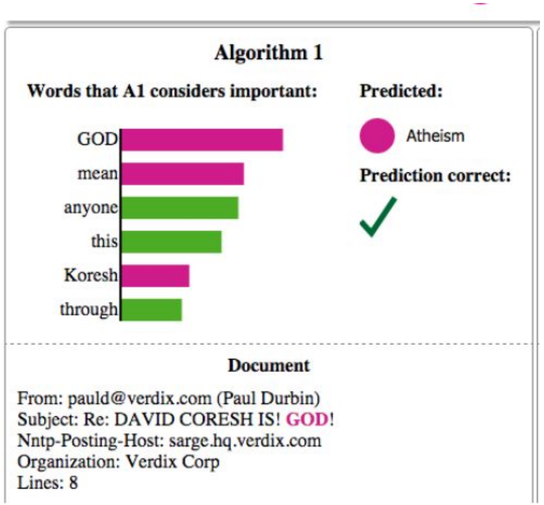
Explainable ML

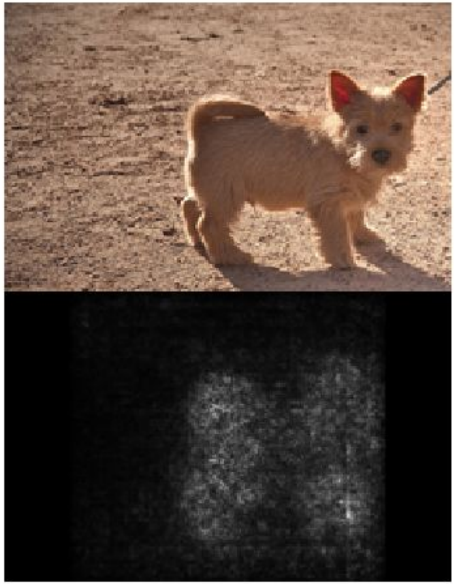
- How does a model work?
  - Locally fitting a simple model at a test sample (LIME)



Explainable ML

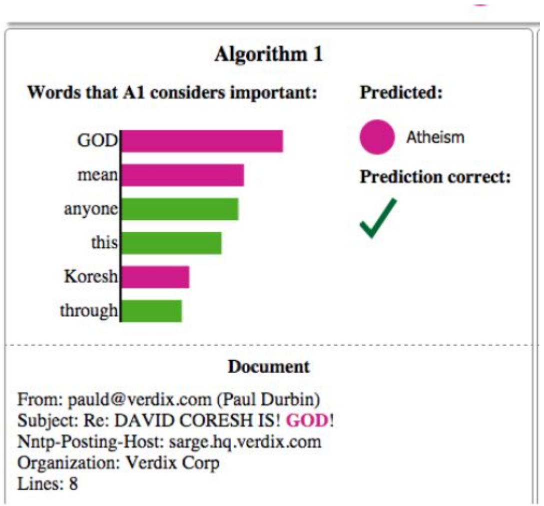
- How does a model work?
  - Locally fitting a simple model at a test sample (LIME)
  - Look at how prediction changes by tweaking the sample (Saliency Maps)

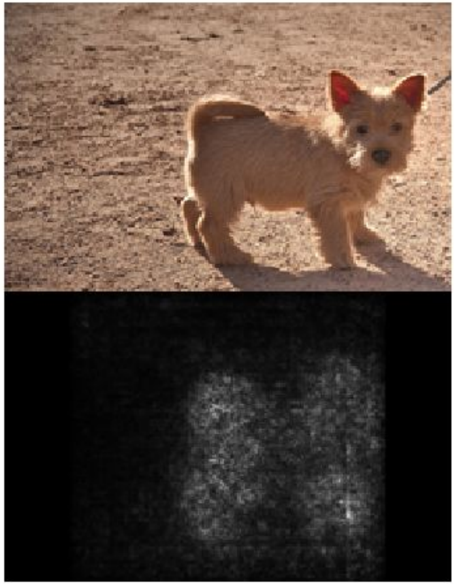




Explainable ML

- How does a model work?
  - Locally fitting a simple model at a test sample (LIME)
  - Look at how prediction changes by tweaking the sample (Saliency Maps)
  - Look at how training samples affect the prediction





Training Data Attribution

- How does a training sample affect the success of a model?
  - If you take away a training sample, how does it affect the loss on a specific test sample ("influence")?
  - Allows us to understand of which data sample is important for the model's decision making

- Issue: too expensive to compute for every training sample :(

Training Data Attribution

- What are some approaches to approximate this?
  - Approximate the influence with Hessian (influence function)

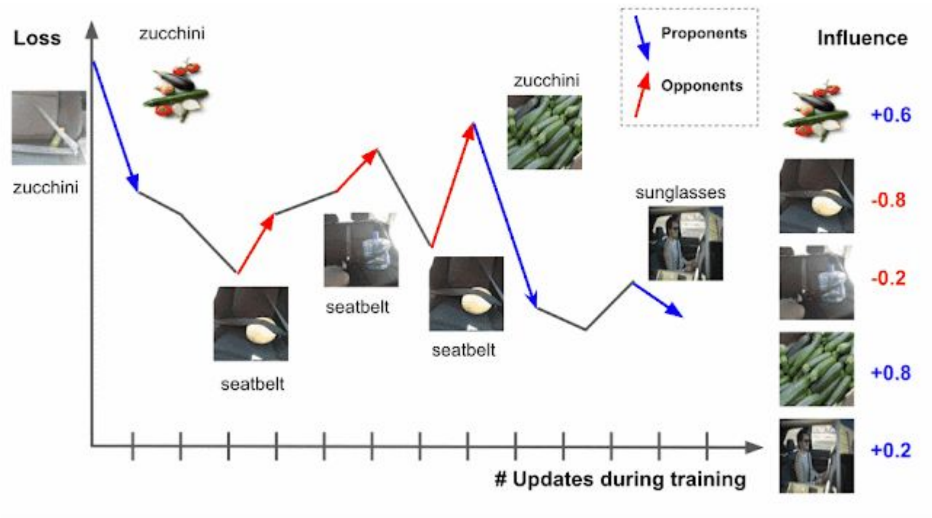
$$= -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta}).$$

Understanding Black-box Predictions via Influence Functions

Pang Wei Koh [1] Percy Liang [1]

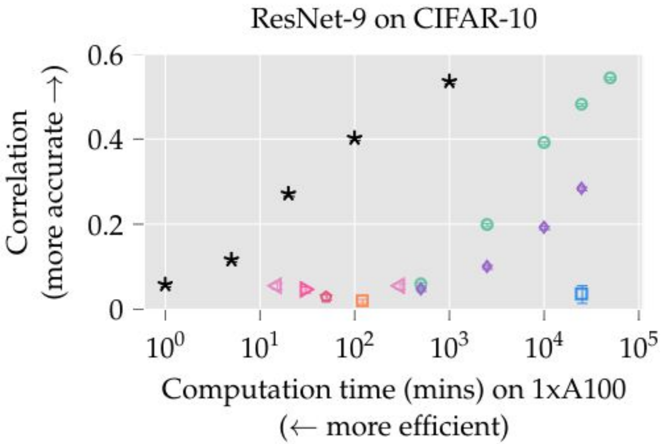
Training Data Attribution

- What are some approaches to approximate this?
  - Compute during training how each sample influenced the final prediction (TracIn)



Training Data Attribution

- What are some approaches to approximate this?
  - Approximate models with a kernel machine that is easier to compute influence score with (TRAK)



Training Data Attribution

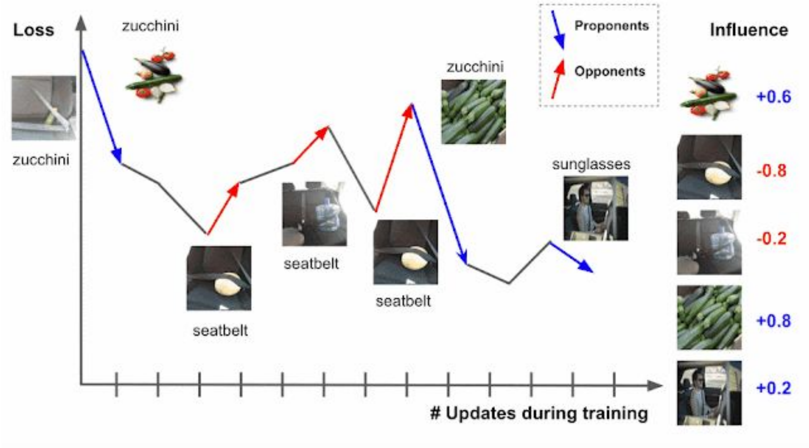
- What might be some issues with these approaches?
  - Approximate the influence with Hessian (influence function)
  -
  - $= -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta}).$
  - If I have a dog image dataset with 100 Kodas, 1 Dub, and other dogs, and we are presented with a Koda picture for testing

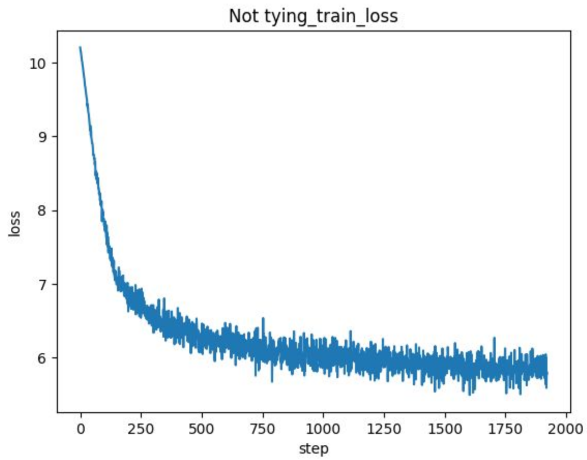
Training Data Attribution

- What might be some issues with these approaches?
  - Approximate the influence with Hessian (influence function)
  -
  - $= -\nabla_{\theta} L(z_{\text{test}}, \hat{\theta})^{\top} H_{\hat{\theta}}^{-1} \nabla_{\theta} L(z, \hat{\theta}).$
  - If I have a dog image dataset with 100 Kodas, 1 Dub, and other dogs, and we are presented with a Koda picture for testing
    - Removing a Koda image might do little to the loss, but removing the Dub image will have a larger impact on the loss

Training Data Attribution

- What might be some issue with these approaches?
  - Compute during training how each sample influenced the prediction final (TracIn)
  - Early training samples will have a larger influence on the final loss!





Simfluence

- What was common with the previous approaches?
- They gave only one score, and assume the scores to be additive. It can't capture the complexity of influence

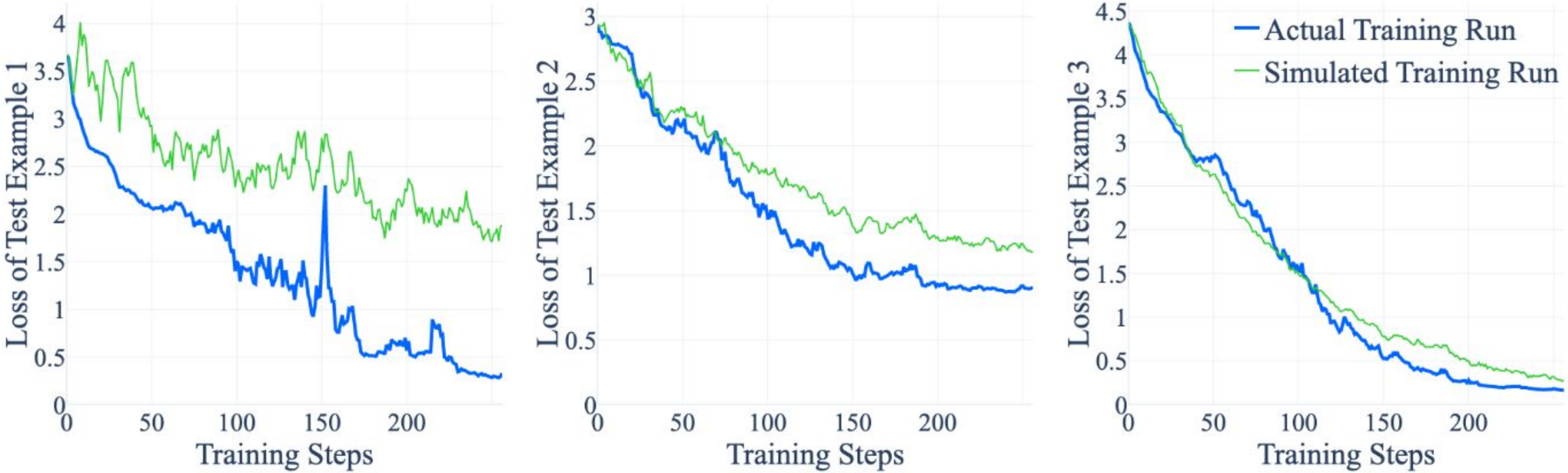
- How to address it?
- Use two numbers instead! Let's in addition frame it as **sim**ulating how training on different samples in**fluence** the test sample performance

- Can I still …?
- Yes, as it's a superset of previously mentioned techniques and more

Simfluence

Goal of Simfluence: predict loss on a test sample for different training “curriculum”

“What if I had removed <X> group of examples from my training data?” “What if I had trained on <Y> first, then <X>?” “What if I duplicated <Z> ten times?”



Formulation

Linear Markov process for each test sample z, and looks at the loss L

$$L_t(z) = \alpha(c_t) L_{t-1}(z) + \beta(c_t)$$

$$\alpha(c_t) = \sum_{i \in c_t} A_i \qquad \beta(c_t) = \sum_{i \in c_t} B_i$$

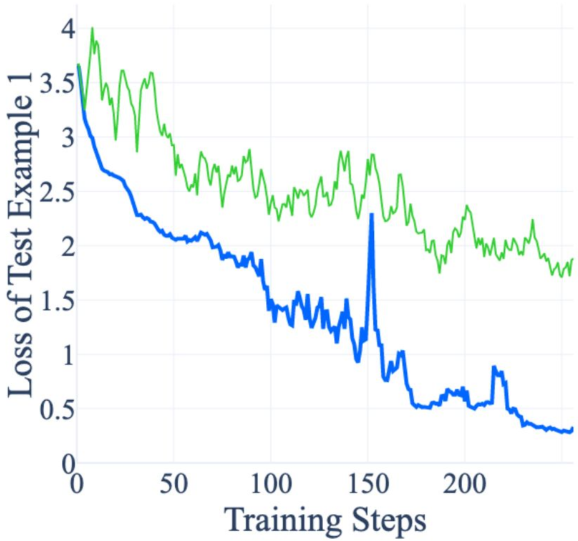
Need to train m simulators for modeling m test samples, with in total 2n parameters

Optimization

$$\mathcal{O}(A,B) = \sum_{t \in \mathcal{T}} \left(L_t(z) - \hat{L}_t(z) \right)^2 + \lambda(\|A\|^2 + \|B\|^2)$$

$$\hat{L}_t(z) = \alpha(c_t) L_{t-1}(z) + \beta(c_t)$$

Evaluation Metric



$$\mathrm{MSE}(\mathcal{S}) = \frac{1}{\mathcal{Z}_{\text{test}}} \frac{1}{T} \sum_{z \in \mathcal{Z}_{\text{test}}} \sum_{t=1}^{T} (L_t(z) - \hat{L}_t(z))^2$$

$$\mathrm{Spearman}_{\text{final}}(\mathcal{S}) = \mathrm{Spearman}(\{(\hat{L}_T(z), L_T(z)) \text{ for } z \in \mathcal{Z}_{\text{test}}\})$$

Results

Task	TDA Method	All-steps Mean Squared Error	Final-step Spearman's ρ
COPA	TracIn-CP (10 ckpts)	$5.873_{\pm 0.307}$	$0.448_{\pm 0.079}$
	TracIn-CP (all steps)	$5.792_{\pm 0.331}$	$0.469_{\pm 0.077}$
	Simfluence-Additive (TracIn-Ideal)	$2.506_{\pm 0.491}$	$0.786_{\pm 0.065}$
	Simfluence-Multiplicative	$1.557_{\pm 0.469}$	$0.763_{\pm 0.046}$
	Simfluence-Linear	$\mathbf{1.503_{\pm 0.494}}$	$\mathbf{0.886_{\pm 0.033}}$
RTE	TracIn-CP (10 ckpts)	$3.317_{\pm 0.323}$	$0.040_{\pm 0.202}$
	TracIn-CP (all steps)	$3.733_{\pm 0.275}$	$0.034_{\pm 0.200}$
	Simfluence-Additive (TracIn-Ideal)	$3.096_{\pm 3.161}$	$0.543_{\pm 0.288}$
	Simfluence-Multiplicative	$5.818_{\pm 14.311}$	$0.599_{\pm 0.182}$
	Simfluence-Linear	$\mathbf{0.819_{\pm 1.222}}$	$\mathbf{0.887_{\pm 0.080}}$

$$L_t(z) = \alpha(c_t) L_{t-1}(z)$$

$$L_t(z) = L_{t-1}(z) + \beta(c_t)$$

Results

Task	Fine-tuning Method	Language Model	Simfluence Model	All Steps Mean Squared Error	Final Step Spearman's ρ
COPA	FMT	T0	ADDITIVE	$0.296_{\pm 0.163}$	$0.278_{\pm 0.141}$
			MULTIPLICATIVE	$1.582_{\pm 4.274}$	$\mathbf{0.629_{\pm 0.058}}$
			LINEAR	$\mathbf{0.272_{\pm 0.425}}$	$\mathbf{0.628_{\pm 0.065}}$
		T5-LMA	ADDITIVE	$0.801_{\pm 0.216}$	$0.368_{\pm 0.117}$
			MULTIPLICATIVE	$0.240_{\pm 0.158}$	$\mathbf{0.672_{\pm 0.079}}$
			LINEAR	$\mathbf{0.175_{\pm 0.113}}$	$0.565_{\pm 0.085}$
	IA3	T0	ADDITIVE	$2.506_{\pm 0.491}$	$0.786_{\pm 0.065}$
			MULTIPLICATIVE	$1.557_{\pm 0.469}$	$0.763_{\pm 0.046}$
			LINEAR	$\mathbf{1.503_{\pm 0.494}}$	$\mathbf{0.886_{\pm 0.033}}$
RTE	FMT	T0	ADDITIVE	$7.385_{\pm 7.696}$	$0.451_{\pm 0.374}$
			MULTIPLICATIVE	$6.818_{\pm 5.063}$	$0.609_{\pm 0.233}$
			LINEAR	$\mathbf{6.709_{\pm 14.756}}$	$\mathbf{0.813_{\pm 0.116}}$
		T5-LMA	ADDITIVE	$14.531_{\pm 8.214}$	$0.122_{\pm 0.193}$
			MULTIPLICATIVE	$2.866_{\pm 3.141}$	$0.086_{\pm 0.087}$
			LINEAR	$\mathbf{2.290_{\pm 2.575}}$	$\mathbf{0.399_{\pm 0.266}}$
	IA3	T0	ADDITIVE	$3.096_{\pm 3.161}$	$0.543_{\pm 0.288}$
			MULTIPLICATIVE	$5.818_{\pm 14.311}$	$0.599_{\pm 0.182}$
			LINEAR	$\mathbf{0.819_{\pm 1.222}}$	$\mathbf{0.887_{\pm 0.080}}$
Winogrande	FMT	T5-LMA	ADDITIVE	$7.060_{\pm 1.961}$	$0.256_{\pm 0.156}$
			MULTIPLICATIVE	$1.496_{\pm 0.400}$	$0.339_{\pm 0.190}$
			LINEAR	$\mathbf{0.910_{\pm 0.104}}$	$\mathbf{0.383_{\pm 0.151}}$

Related Works

	Simfluence	Influence function	TracIn	TRAK
Influence score	Two scores $L_t(z) = \alpha(c_t)L_{t-1}(z) + \beta(c_t)$	One score	One score $L_t(z) = \quad L_{t-1}(z) + \beta(c_t)$	One score
Mechanism	Simulation based	Influence function	Gradient based	Sampling based
Compute bottleneck	Track performance of a test sample for 2n training samples	Compute Hessian (with some approximation speedups)	Do Simfluence method or an approximation by evaluating checkpoint	Simplify model with a kernel machine

References

Guu, Kelvin, Albert Webson, Ellie Pavlick, Lucas Dixon, Ian Tenney, and Tolga Bolukbasi. “Simfluence: Modeling the Influence of Individual Training Examples by Simulating Training Runs.” arXiv, March 14, 2023. http://arxiv.org/abs/2303.08114.

Koh, Pang Wei, and Percy Liang. “Understanding Black-Box Predictions via Influence Functions.” arXiv, December 29, 2020. http://arxiv.org/abs/1703.04730.

Park, Sung Min, Kristian Georgiev, Andrew Ilyas, Guillaume Leclerc, and Aleksander Madry. “TRAK: Attributing Model Behavior at Scale.” arXiv, April 3, 2023. http://arxiv.org/abs/2303.14186.

Pruthi, Garima, Frederick Liu, Mukund Sundararajan, and Satyen Kale. “Estimating Training Data Influence by Tracing Gradient Descent.” arXiv, November 14, 2020. http://arxiv.org/abs/2002.08484.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. “‘Why Should I Trust You?’: Explaining the Predictions of Any Classifier.” arXiv, August 9, 2016. http://arxiv.org/abs/1602.04938.

Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps.” arXiv, April 19, 2014. https://doi.org/10.48550/arXiv.1312.6034.

Discussion Questions

- Do you think there could be a better “influence score”, instead of modeling a function?
- How would you design a better simulator? Something that goes beyond a linear setup? What are possible use cases?
- What do you think about their data requirements?
- What do you think about their standard error?
- How do you think about their evaluation? Are there issues with their setups?

Discussion Questions

- Are you convinced by the choice of treating PBRF as ground truth? Can you think of an alternative method to verify the influence estimate?
- What are the differences between adding and subtracting an example from the training set under the notion of influence function?
- Applying to modern instruction-tuning setup, how would you pick and aggregate the feature given an example?
- What kind of inaccuracy do you expect the approximation would induce?
- What other analysis do you think is worth performing with influence function?
- Beyond the analysis angle, what other applications can you imagine with influence function?