Compact proofs of model performance via mechaninistic interpretability

Refer: https://github.com/LouisYRYJ/Proof_based_approach_tutorial/blob/master/proof_public.ipynb

Introduction

The problem of understanding a model

Why do we care about about understanding a model?

When we use or train a model, many things can go wrong. For example, during training the model can learn undesired behaviour, which is not obvious to us (deceptive alignment). Or it might have failure modes that are not salient to us (adversarial examples). On the other hand, we could steer a model towards a desired behaviour, if we understood how it works.

All of these issues could be resolved, if the models were transparent to us (though this is not the only approach). So an important question to raise here is: What do we mean when we talk of "a mechanistic understanding" of a model? When is a model transparent to us?

This is a difficult question! Let's say you study a model and reverse engineered parts of it, like a circuit. How can you be sure that the circuit you found actually does the thing you are claiming it is? Let's look at the specific example of autoencoders. This is a quote from Towards Monosemanticity: Decomposing Language Models With Dictionary Learning

> Usually in machine learning we can quite easily tell if a method is working by looking at an easily-measured quantity like the test loss. We spent quite some time searching for an equivalent metric to guide our efforts here, and unfortunately have yet to find anything satisfactory.
>
> We began by looking for an information-based metric, so that we could say in some sense that the best factorization is the one that minimizes the total information of the autoencoder and the data. Unfortunately, this total information did not generally correlate with subjective feature interpretability or activation sparsity.[…]
>
> Thus we ended up using a combination of several additional metrics to guide our investigations[…]
>
> Interpreting or measuring some of these signals can be difficult, though. For instance, at various points we thought we saw features which at first didn't make any sense, but with deeper inspection we could understand.
>
> We think it would be very helpful if we could identify better metrics for dictionary learning solutions from sparse autoencoders trained on transformers.

See also Section 5 of this review Mechanistic Interpretability for AI Safety – A Review for more references on the difficulty of evaluating interpretability results.

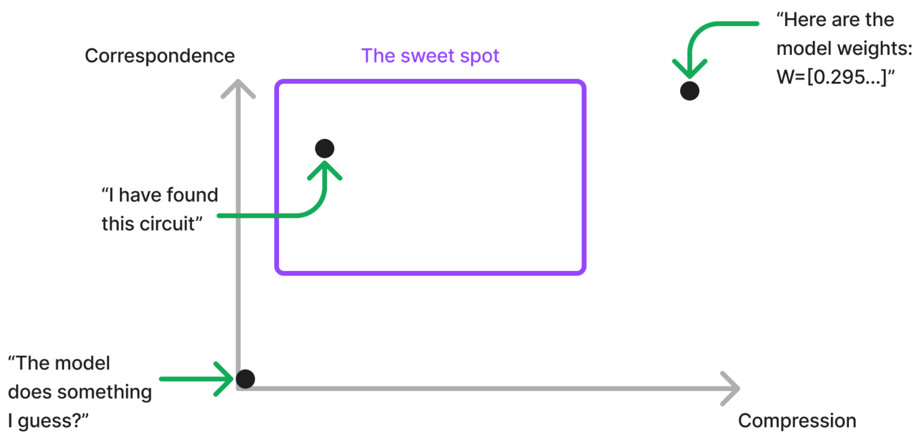
Quantative methods for interpretability

Having quantative methods would not only make mechanistic interpretability research more rigorous. If we want to scale up methods to huge models, we will need to automate parts of the process and we won't be able to have a human in the loop at every crucial point. A lack of quantative benchmarks makes this task seem almost impossible. To spoiler the punchline: Compact proofs provide such a quantative benchmark, although they currently are infeasible for larger models.

Before getting into the details, let's nail down two things that we want to quantify. The following two points are taken from the Compact proofs blog post , see also this comment by Ryan Greenblatt.

1. Correspondence (or faithfulness): How well our explanation reflects the model's internals.
2. Compression: Explanations compress the particular behavior of interest. Not just so that it fits in our heads, but also so that it generalizes well and is feasible to find and check.

Specifically, the second points implies that the explanation, say the circuit that we discovered, should be more **compact** and therefore more understandable for us humans: The weights of a model are a perfectly faithful explanation of its behaviour, but this explanation is not helpful for us.


Correspondence
The sweet spot
"Here are the model weights: W=[0.295...]"
"I have found this circuit"
"The model does something I guess?"
Compression

An important insight that we will make is that our explanations are not as good as we might think. Specifically, **noise** in the model's weights seem negligible. But worst case bound imply that it could still be an important contribuiting factor. In fact, it might be that something that we deem as noise, is important for the model's computation, but we simply don't understand it. This issue with the noise is another point that a quantative evaluation should be able to address.

What are compact proofs?

Compact proofs are an attempt at formalizing the above diagram.

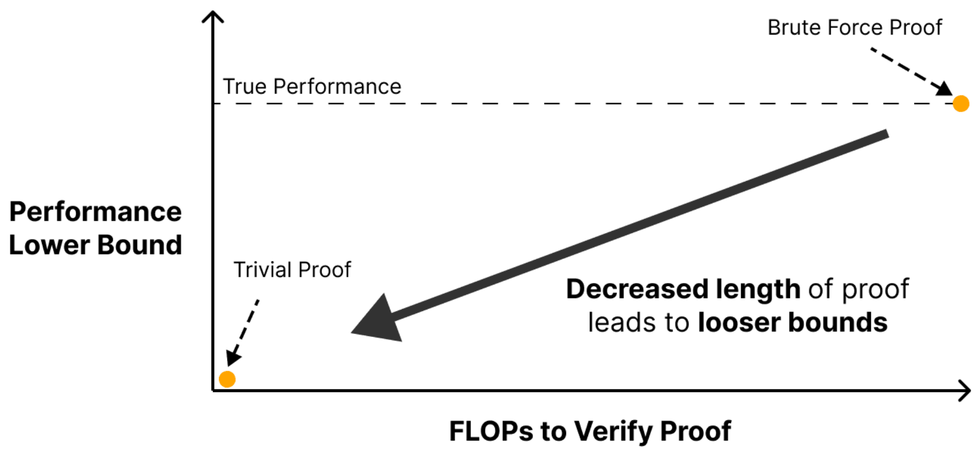
First of all, what do we mean by proof i.e. what are we trying to prove? Say we are training a model with weights θ on some task. The kinda of statements that we want to prove are of the form

$$\mathbb{E}[f_\theta(x)] \geq b$$

where f_θ is a quantity that we are interested in bounding from below or above (depending on the quantity), such as loss or accuracy.

The compactness of a proof is determined by its length. A good proxy for the length is the FLOPS required to run the proof, see the paper for more details. (Maybe I will write more on this)

So once we have a proof, we can measure its correspondence by looking at the bound and measure its compactness by measuring its length. We get a similar picture to the one drawn above:

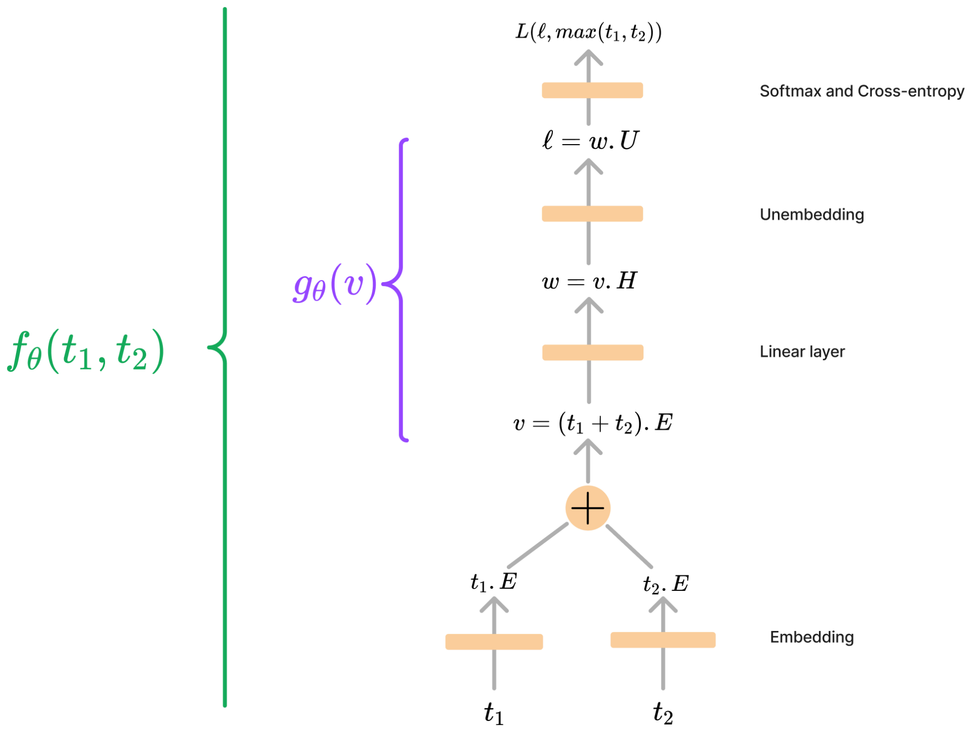


We will see many examples of proofs and compare their performance below. The ideal goal here would be to have a pipeline that takes in a vague interpretation of the model, turn that into a rigorous proof, and evaluate the interpretation based on the correspondence and compactness of the proof.

As we will see below this turns out to be rather difficult, even in toy models. The takeaway here is that quantification seems to be a hard problem and the compact proof approach is an example of this.

Max-of-2 example

Having worked through the high level picture, let us now focus on concrete examples of compact proofs and explain what it means. Let's say we have a model:



refer: `main.py > class MLP()`

In our first example, we will train the model to predict the max of the two tokens, where the tokens range from 0 to d_{vocab}. We will be interested in estimating the global loss of this model, that is we want to estimate

$$\mathbb{E}[f(t_1, t_2)] = \frac{1}{d_{\text{vocab}}^2} \cdot \sum_{t_1, t_2 \in \{0, \ldots, d_{\text{vocab}} - 1\}} f(t_1, t_2).$$

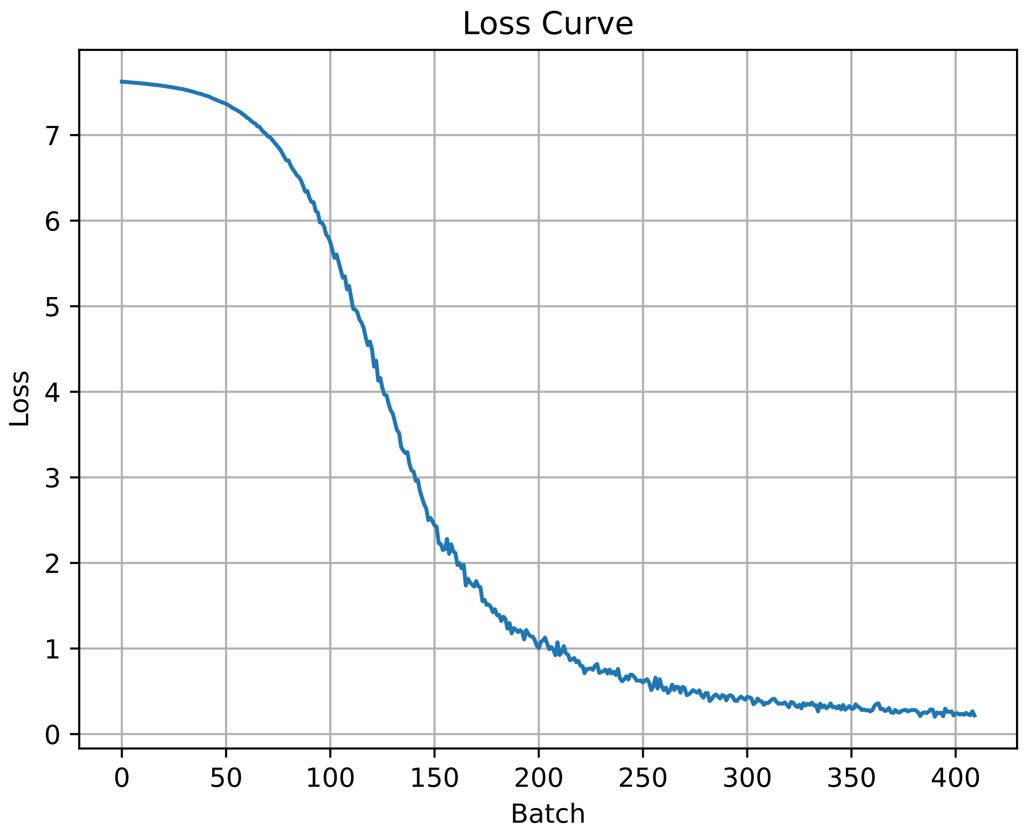
The general proof strategy consists of two steps:

1. P1: Prove a statement that given a model with its weights θ, there is a quantity $C(\theta)$ such that $\mathbb{E}[h(x, M(x))] \leq C(\theta)$.
2. P2: Compute the quantity $C(\theta)$.

We will come back to this after we did some proofs and also discuss what it means to have a **compact** proof.

We train our model on $\sim$ 10% of the whole data set. (This is not correct, dataset is random each batch)

refer: `main.py > params = Parameters() ...`



Note that this is our **training set** loss.

refer: `main.py > loss_history[-5:]`

```
[
  0.2482442557811737,
  0.23066267371177673,
  0.21940861642360687,
  0.26723822951316833,
  0.218377023935318
]
```

Brute force proof

Let's start proving things now! One thing we can do is a brute force proof. Remember that our proofs will consist of two steps. The first step (P1) is as follows

Theorem(Brute force proof): The expected loss of a model M with weights θ is bounded above by $\mathbb{E}[f(t_1, t_2)]$.

Proof: By definition the bound is actually an equality.

That was an easy proof, but the ones we will encounter from now on will be more difficult and feel less tautological! Now we come to the second part of the proof (P2), which in this case means computing this quantity.

refer: `main.py > def brute_force()`

A word on the "formal" in "formal proofs"

Strictly speaking we would want to formalize our proofs, meaning that we would rewrite them in a form that can be verified by a formal proof assistant such as Lean or Coq. For a more serious use case this would be indeed necessary, but for now I will leave it open to the you, the reader, to formalize the proof that you would want to be verified. Generally speaking, be aware though that Lean and Coq also have bugs!

Symmetry proof

Our brute force proof gave the optimal bound, but at the cost of having to compute all the inputs. Can we do better? Yes!

We start with the first part P1 of our proof. It will be based on the following observation

Lemma: Let $f_\theta(t_1, t_2)$ denote the neural network as depicted above. Then

$$f_\theta(t_1, t_2) = f_\theta(t_2, t_1).$$

Try to prove that statement!

Proof

This is a consequence of the following equalities

$$f_\theta(t_1, t_2) = g_\theta(t_1 \cdot E + t_2 \cdot E) = g_\theta(t_2 \cdot E + t_1 \cdot E) = f_\theta(t_1, t_2).$$

Now you can use that statement to prove the following statement.

Theorem: The expected loss of a model M with weights θ is bounded by (and in fact equal to)

$$\frac{1}{d_{\text{vocab}}^2} \cdot \left[\sum_{t_1 < t_2} 2 \cdot f_\theta(t_1, t_2) + \sum_{t_1} f_\theta(t_1, t_1)\right].$$

Proof

This is a consequence of the following equality

$$\begin{aligned}\sum_{t_1, t_2} f_\theta(t_1, t_2) &= \sum_{t_1 < t_2} f_\theta(t_1, t_2) + \sum_{t_1} f_\theta(t_1, t_1) + \sum_{t_1 > t_2} f_\theta(t_1, t_2) \\ &= \sum_{t_1 < t_2} f_\theta(t_1, t_2) + \sum_{t_1} f_\theta(t_1, t_1) + \sum_{t_2 < t_1} f_\theta(t_2, t_1) \\ &= \sum_{t_1 < t_2} 2 \cdot f_\theta(t_1, t_2) + \sum_{t_1} f_\theta(t_1, t_1).\end{aligned}$$

Now we can come to the second part P2 of our proof – actually computing the quantity.

refer: `main.py > def symmetry_proof_loss()`

Convexity proof

So far our proofs didn't involve any proper bounds. We will now start using more coarse bounds, but this will lead to a great increase in compression. The central notion for this section is **convexity**.

Definition:

- A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex, if $\forall x, y \in \mathbb{R}^n$ and $t \in [0, 1]$, we have an inequality

$$f(tx + (1 - t)y) \leq t f(x) + (1 - t) f(y).$$

- A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is convex, if all the projections $f_1, \ldots, f_m : \mathbb{R}^n \to \mathbb{R}^m \xrightarrow{pr_i} \mathbb{R}$ are convex.

1. Proving that $L \circ g_\theta(v)$ is convex:
First we prove this general statement:

Lemma: Let $f_1 : \mathbb{R}^n \to \mathbb{R}^m$ and $f_2 : \mathbb{R}^m \to \mathbb{R}^k$ be a linear and a convex function respectively, then $f_2 \circ f_1 : \mathbb{R}^n \to \mathbb{R}^k$ is convex as well.

Proof

This follows from the following inequalities

$$f_2(f_1(t \cdot x + (1-t) \cdot y)) = f_2(t \cdot f_1(x) + (1-t) \cdot f_1(y)) \leq t \cdot f_2(f_1(x)) + (1-t) \cdot f_2(f_1(y))$$

where the first equality follows from linearity of f_1 and the second one from convexity of f_2.

Now we will combine the above statement with the following lemma.

Lemma: $g_\theta(v)$ is linear and $L(-, x)$ is convex.

Proof

- $g_\theta(v)$ is a composition of linear functions, therefore also linear.
- There are several ways to prove that L is convex. One could verify that the Hessian of L is positive semi-definite or directly apply the Hölderlin inequality. Let's use the latter approach. Our goal is to show that for $x \in \mathbb{R}^m$ we have the following function is convex

  $$-\log\left(\frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}\right)$$

  where we fixed an $i \in \{1, ..., m\}$ (corresponding to the correct label). We can rewrite the function as

  $$-x_i + \log\left(\sum_{j=1}^{m} e^{x_j}\right)$$

  and it suffices to show that $\log\left(\sum_{j=1}^{m} e^{x_j}\right)$ is convex. This follows from the Hölderlin inequality

  $$\sum e^{t \cdot x_i} e^{(1-t) \cdot y_i} \leq \left(\sum e^{x_i}\right)^t \cdot \left(\sum e^{y_i}\right)^{1-t}.$$

2. Proving a bound for $f_\theta(t_1, t_2)$
Theorem: The expected loss of a model M with weights θ is bounded above by

$$\frac{1}{d_{\text{vocab}}^2} \cdot \left[\sum_{t_1} L(g_\theta(t_1, t_1), t_1) + \sum_{t_1 < t_2} L(g_\theta(t_1, t_1), t_2) + L(g_\theta(t_2, t_2), t_2)\right].$$

Note that we can rewrite the first and last term as $f_\theta(t_1, t_1)$ and $f_{\theta(t_2, t_2)}$ respectively, but we can't rewrite the middle term in terms of f_θ (Why?).

Proof

From the symmetric proof section we have seen that expected loss is equal to

$$\frac{1}{d_{\text{vocab}}^2} \cdot \left[\sum_{t_1} f_\theta(t_1, t_1) + \sum_{t_1 < t_2} 2 \cdot f_\theta(t_1, t_2)\right].$$

From the previous lemma we have seen, given $t_2 > t_1$, that

$$f_\theta(t_1, t_2) = L(g_\theta(t_1 \cdot E + t_2 \cdot E), t_2) \leq \frac{1}{2} \cdot L(g_\theta(2 \cdot t_1 \cdot E), t_2) + \frac{1}{2} \cdot L(g_\theta(2 \cdot t_2 \cdot E), t_2)$$

.

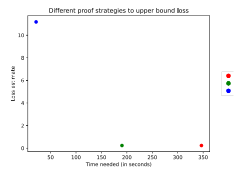
Combining the previous two statements yield the desired statement.

Now we come to the part P2 of our proof – computing the above quantity.

refer: `main.py > convexity_proof()`

Summary

refer: `main.py > x = [performance[][] ...]`



Let's break down what we see. On the x-axis we have the time it took to run the proofs. This is a proxy for the compactness of the proof. Another related notion would be to track the FLOPS needed to run the proof. We see that the symmetric proof takes about half of the time compared to the brute force one. This makes sense:

There is a total of d^2 inputs that the brute force proof needs. On the other hand the symmetric proof needs only $\frac{d(d+1)}{2}$ inputs. But note that our architecture was very simple – we can't expect such an easy improvement in general. Note also that asymptotically both proofs scale **quadratic** in d.

On the other hand the convex proof scales only **linearly** in d. This is great, but it comes at a great cost: Our bound is very bad and we find ourselves on the other end of trade off, close to a vacuous proof. In fact, there is a straight forward explanation why our bound will always be comparatively coarse. Can you figure out why?

The reason...

...that we should expect a very coarse bound in the expression

$$\left[\sum_{t_1} L(g_\theta(t_1, t_1), t_1) + \sum_{t_1 < t_2} L(g_\theta(t_1, t_1), t_2) + L\Big(g_{\theta(t_2, t_2)}, t_2\Big)\right].$$

is the middle term $L(g_\theta(t_1, t_1), t_2)$. This term computes the cross-entropy loss **not** between $\ell_{[t_1, t_1]}$ and $t_1 = \max([t_1, t_1])$, but between $\ell_{[t_1, t_1]}\}$ and t_2. Thus even if our model would perform optimally, this term would remain high.

This concludes the first example! In the next section, we will explore a slightly different set up, where our convex proof will yield useful results. After that we are ready to dig into the proofs of the paper and replicate some of them. These will turn out to be more difficult, but will also make use of more serious mechanistic machinery.

Making convex work for $n = 3$

TODO

Implementing the cubic proof

TODO