Theoretical Foundation for AI Alignment

Xiyu Zhai, Alexander Rakhlin

Contents

Abstract

This paper attempts to build a theoretical foundation for AI alignment based on statistical and machine learning theory, type theory and logic. We extend upon type systems from formal verifications to form a more powerful framework than knowledge graph that will be able to systematically verify natural language outputs produced by AIs. As AIs are getting stronger and stronger, it's imperative that all human beings should unite to produce such systems for a guaranteed alignment of AIs with human values. In the future, we shall have truly safe AIs that generates verifiable "proofs" alongside any statements they make.

1 Introduction

This paper describe a rather rudimentary but import problem. The authors are not aware of similar work. If there exists such, please inform!

Human can never fully trust large language models as much as they can trust traditional software. The reason is simple: they just have too many parameters!

From machine learning theory, when test error is close to zero, we have the generalization bound being of form

$$O\left(\frac{\sqrt{d}}{n}\right). \tag{1}$$

Suppose we want the error is to be smaller than ϵ, then

$$n \geq \Omega(\sqrt{d}/\epsilon) \tag{2}$$

When $\sqrt{d}$ is large, and ϵ is very small, we need an enormous amount of data for guarantees. GPT3.5 have 175billion parameters, let's assume that the effective VC dimension is much smaller, say 10billion, and ϵ is taken to be $10^{-4}/N$ where N is the number of users, then we need more than N billion number of data and it becomes unrealistically large when N is reasonably large. (ϵ is really needed to be that small for broader usage of AI in real life, dealing with adversarial attacks, corner cases)

In contrast, traditional software are much more stable, despite the fact that they might be built from billions of lines of code. There are multiple ways to guarantee soundness for traditional software:

- formal verification. This is tedious to do but when done completely guarantee correctness (unless the verifier itself is buggy, which is very unlikely because verifier can be written using a very small amount of code). *In contrast, deep learning weights cannot be proved to be correct.*
- type system. Compilers do some sketchy checking so that many of the errors are prevented at compile time. *In contrast, deep learning weights are only constraint by the architecture (weight sharing) and have universal approximation property, meaning they can approximate both angles and demons.*
- modularity. Modern software is written in modular ways, allowing the ultimate accuracy to be dependent only on a few critical components rather than the whole system. And when something goes wrong, it's possible to backtrace to a few expressions among the billions of lines of code. *In contrast, deep learning is trained end-to-end, every weight contributes to the final result, leading to large generalization gap and making it impossible to backtrace.*

In this paper, we intend to close this gap of reliability between AI and traditional software by inventing a verification system that is a database of rules that is learnable, modular and incorporate traditional type system and formal verification, knowledge graph and neural symbolic ones. Generate AIs can have guaranteed soundness by providing a sequence of rule application alongside its normal output. It still belongs to system 1, and corresponds to both our intuitive judgment of whether something makes sense. It might be possible that system 2 is the interaction process between generative AI and a verifier system. We shall focus on these aspects in this paper:

- Details about verification process, how it extends traditional formal verification, knowledge graph and expressive enough to verify natural language.
- Statistical guarantees. A verification system needs to be constantly modified after bugs are found. We apply statistical and machine learning theory to argue the correctness is still guaranteed after modification.
- Development/Training. We discuss the optimal way to construct such a verification system. Lessons from expert system tell us scalability is essential. We propose multiple ways of construction that take minimal human labor (but maybe more AI labor), and easy to update and maintain (could be some form of gradient descent, etc.)

2 Abstract Proof Machine

Here we describe an abstract proof machine that avoids the details of logical foundation yet suffices for ensuing discussions.

It's necessary to avoid early specification of logical foundation because we are going to a much more permissive logical foundation than any existing ones, and soundness of the system can be empirically guaranteed without resorting to a mathematical proof, thus foundation is irrelevant. (todo: include in appendix more discussions about logic foundation)

An abstract proof machine consists of a append-only tape of statements as the state, and a small finite set of rules of generating new statements. Let $\mathfrak{S}$ be the set (type) of all possible statements (including `true`, `false`), then formally a rule r of n arms is a trivially computable function

$$\underbrace{\mathfrak{S} \times \cdots \times \mathfrak{S}}_{n \text{ copies}} \to \mathfrak{S} \ . \tag{3}$$

A state of the machine is then a finite sequence $s_1, \cdots, s_k \in \mathfrak{S}$. One can apply a rule r of n arms over $s_{i_1}, \cdots, s_{i_n}$, then the machine changes by appending $s_{k+1} = r(s_{i_1}, \cdots, s_{i_n})$ to the sequence.

We require that any statement keeps track of the way it's constructed, i.e., the total dependencies and dependency tree of a statement can be seen as a function of the statement.

Remark. *We have a few remarks,*

- *Rather than implying that any statement on the tape is true, the machine is implying that a statement on the tape is true if all its dependencies are true, giving us flexibilities important for its broader applications. We refer to the collection of the rules used for deriving that statement as its proof.*
- *Unlike theorem proving and formal verification, here* ***proof details are relevant!*** *Different ways of constructing a proof lead to different dependencies, the validity of which is depending ultimately on hidden and possibly changing assumptions.*
- *Everything can be viewed as category, or even topos. However, that viewpoint forgets the computation aspects. Verifications systems should be designed with ease of computation in mind, just like type systems.*

Note that we have described things with great abstraction, so that we need to fill in more details for examples.

todo: describe some common rules and statement forms for all following examples.

Example (basic maths)**.** We could have a tape like this, with x a symbol of type `Nat`, i.e. natural numbers,

```
#0: x = 1 (intro)
#1: x = 2 (intro)
#2: x + 1 = 2 (derived from #0)
#3: 1 = 2 (derive from #0, #1)
```

Example (probabilistic maths)**.** We could have a tape like this, with X a symbol of type `Nat`, i.e. natural numbers,

```
#0: with probability >= 0.9, X = 1 (intro)
#1: with probability >= 0.2, X = 2 (intro)
#2: with probability >= 0.9, X + 1 = 2 (derived from #0)
#3: with probability >= 0.1, 1 = 2 (derived from #0, #1)
```

Example (blurry math)**.**

3 Alignment as Optimization Problem

4 Statistical Guarantees