The Computation Hypothesis

1 Introduction

It has been colloquially observed by many in the AI community that mathematics has played a much smaller role in the deep learning era than in the previous symbolic and probabilistic eras. In this short paper, we argue that the difficulty of deriving useful theorems for deep learning is a consequence of something more fundamental - namely, that if a problem requires a large amount of computation, it will be inherently difficult to study *with certainty*. We can know with high confidence that computation will optimize our problem well, but when properly immersing each part of a problem with healthy amounts of computation, we won't be able to use mathematics to peer through that optimization process with rigor.

After developing our argument, we suggest that rather than attempting to impose *external* mathematics on intelligent systems, we can work with the computation hypothesis rather than against it by encouraging our systems to foster an *internal* mathematics. Such an approach is bound to achieve greater scalability and is thus more suitable for the AI safety problem.

2 Formulation

Our goal is to study properties of functions learned from optimization problems. The primary insight of the early machine learning field was that a programmer could achieve more accurate results by letting a computer optimize over a small class of mathematical functions than by designing the entire function by hand. This optimization problem is to find

$$f^*(X,Y) = \underset{f \in \mathcal{F}}{\text{Opt}}[l(f(X),Y)] \tag{1}$$

where X and Y are the desired inputs and outputs, $\mathcal{F}$ is the class of functions under consideration, and l specifies a loss function measuring the success of any particular $f \in \mathcal{F}$. The case of hand-coded heuristic is covered by letting $\mathcal{F}$ be the singleton class $\{f^*\}$, but of course early machine learning demonstrated convincingly that better results could be obtained by expanding $\mathcal{F}$ to larger classes of functions. For example, performance could be improved by expanding $\mathcal{F}$ to contain all linear functions over hand designed features of the input. This performance gain came at the sacrifice of a certain amount of interpretability, but the ends justified the means.

The benefit of keeping $\mathcal{F}$ small is that it enables us to certify various properties of the output ahead of time, in spite of not knowing exactly which function will be optimal. We define a property to be anything that tells us about what sort of function we have conjured in the process of optimizing over $\mathcal{F}$:

$$p : \mathcal{F} \times X \to \mathbb{R} \tag{2}$$

But we now argue that the process of defining these properties and understanding their results ahead of time runs directly contrary to "The Computation Hypothesis", which we define as follows.

Definition: The Computation Hypothesis
For complex problems, we will by default expect to find meaningfully better solutions to Equation 1 as we expand the class of considered functions $\mathcal{F}$, and consider more sophisticated optimization functions Opt, and loss functions l.

That is, we hypothesize that our system will be able to make use of the computation that we

provide it. From a purely mathematical perspective, it's difficult to see why this is true. We can often prove that a particular algorithm (e.g. brute force search, or SGD) can be made to work with a particular amount of resources, but proofs that problems *need* any minimal amount of computation are few and far between. Thus, we simply frame this observation as a null hypothesis about complex problems - there are no doubt simple problems that can be solved well, if not perfectly, with little to no computation, but as the scope of the problem nears that which we would expect the assistance of a highly intelligent agent to solve, the hypothesis becomes more and more believable.

Let us now see some rather concerning implications of the hypothesis. We'll start with the consequences of expanding $\mathcal{F}$, and then more precisely state what we mean by sophisticated optimization and loss functions, and explore the implications of these on the difficulty of mathematical analyzing properties (p) of the intelligent functions we learn.

3 Shortcomings

In this section, we enumerate how the different aspects of The Computation Hypothesis act against our abilities to prove properties of our intelligent functions ahead of time.

3.1 Expanding the function class, $\mathcal{F}$

The trajectory of machine learning has consistently shown that, when the necessary compute budget is available, considering more complex functions from inputs to outputs allows for better predictions. Heuristics were superceded by SVMs, SVMs by random forests, and random forests by deep networks. And even deep networks show many signs of weakness to adversarial examples that are thought to originate from their excessive linearity - thus we may expect that deep networks will eventually be replaced with systems capable of expressing even more complex functions. But while we get better performance by increasing the size of $\mathcal{F}$, we can say less and less about $p(f^*, X)$ ahead of time. In the limit of $\mathcal{F}$ containing all possible functions from the input to the output, which is the direction in which The Computation Hypothesis points, we can only say things based on generic upper and lower bounds on p and the type signature of functions in $\mathcal{F}$. For example, if we are solving a NP-hard problem about the existence of a path, we find that f^* will have an output in $\{0, 1\}$, and thus we can still easily certify whether our discovered function is optimal. But in general, The Computation Hypothesis points against the rich source of prior knowledge contained in constraints on $\mathcal{F}$.

3.2 Sophisticated Optimization Functions

The inability to constrain $\mathcal{F}$ would itself be surmountable if we had certainty about the optimization process. But the optimization process in Equation 1, denoted Opt, is itself a function, $\sigma : \{\mathcal{F}\} \times l \times X \times Y \to \mathcal{F}$, whose success can be measured as

$$\text{success}(\sigma) = l(f^*_\sigma(X), Y) + \lambda * \text{cost}(\sigma) \tag{3}$$

where "cost" is a function measuring the computational cost of actually running the optimization procedure and $\lambda > 0 \in \mathbb{R}$ is a parameter favoring cheaper optimization methods, though they may not be as effective. Given the complex nature of the optimization procedure, σ, it's inputs, and outputs, we expect from The Computation Hypothesis that the best σ for complex problems will be arrived at through an optimization procedure following:

$$\sigma^*(X, Y, l, \mathcal{F}) = \underset{\sigma \in \Sigma}{\text{Opt}}[\text{success}(\sigma(\mathcal{F}, l, X, Y)] \tag{4}$$

where Σ is a large hypothesis class of optimization procedures, and Opt is a meta-optimizer. As described in the above section, we can have no expectation of defining properties over Σ, and as a recursive consequence of this section, we can neither expect to understand the behavior of the meta-optimizer. Thus, we won't be able to prove any properties about the optimizer, σ^* of our original problem, through which we might have derived properties on f^*.

3.3 Sophisticated Loss Functions

We now note that this problem only grows as one considers increasingly intelligent systems with complex objectives. Such systems are bound to find uses for "curiosity functions" that add weakly supervised surrogate objectives, l_0^*, allowing them to find patterns in their inputs and make efficient use of limited data. We can reasonably expect these functions to be learned from data with their own success functions analogous to the optimization success function defined above. The final loss to be optimized will thus appear as:

$$l^* = l + l_0^* \tag{5}$$

where

$$l_0^*(\vec{X}, \vec{Y}, \vec{F}, \vec{\sigma^*}) = \underset{l_0 \in \mathcal{L}_\prime}{\text{Opt}} [\mathbb{E}_{i \in [1 \ldots n]}[l_i(f^*(X_i, \sigma_i^*, l_0), Y_i)]] \tag{6}$$

The result will be sophisticated loss functions optimized over their track record of producing intelligent systems, and on which we cannot expect to define properties *a priori*. Thus, when considering f^* in the generic case of a system that is given some input and expected to produce a response, guaranteeing properties of that response cannot make use of properties based on prior knowledge of the loss function. This only intensifies the difficulty of certifying properties on the behavior of the system, f^*, already made difficult by the fact that The Computation Hypothesis discourages constraints on $\mathcal{F}$ or σ^*.

4 Solutions

4.1 Putting properties into the loss function

If specifying properties with certainty by constraining $\mathcal{F}$ will lead to worse performance, one solution is to instead add these properties to the optimization objective, as soft constraints. The optimization problem then becomes:

$$f^*(X, Y) = \underset{f \in \mathcal{F}}{\text{Opt}}[l(f(X), Y) + p(f, X)] \tag{7}$$

This strategy does not run contrary to the computation hypothesis. But in the general case, these properties can only be expected to hold in a soft way. This is problematic for properties that must be known to hold with high confidence, such as perhaps an AI safety property. If we wish to prove certain properties of the result f^*, so that we may use them as the basis of other proof-based inference, we will find that even ϵ error in the property holding will prevent us from making any further progress at proof-level certainty. Several positive characteristics arise from putting properties in the loss function, although the benefits fall short of providing proofs of properties over f^*.

4.1.1 Cons of optimized properties

We reiterate that when the problem is complex, putting properties into the optimization objective removes any hope of these properties being "proof quality". For example, creating a safe, strong AI by making a system capable of generating a strong AI and then adding a suggestion that it should also try to be safe would hardly constitute a provably safe strategy. A second con is that adding these properties to the optimization function may slow down convergence, as it makes the problem more complicated. However, this increase in optimization cost is likely to be tolerable if the properties are important. Finally, a third con is that in the absence of provability, the certainty of the properties holding will be proportional to a mixture of the size of the dataset, $|X|$, and the amount of computation for testing properties, as each evaluation may require at least the cost of computing $f(x)$ many times. If done correctly, the strategy of optimizing for properties and then certifying that they hold by testing them can still yield properties with high confidence, without contradicting The Computation Hypothesis. We will now discuss strategies that could be seen to encourage the confidence of our certificates, and decrease the cost of computing them, both in terms of data and computational resources.

4.1.2 Pros of optimized properties

The key to producing a system that we can understand well is that it satisfies important properties, and that these properties can easily be verified. The difficulty of verifying properties is that the process may require large amounts of data and computation. Thus, we can imagine optimizing to reduce the verification costs of our properties as well. The final form looks something like:

$$f^*(X,Y) = \underset{f \in \mathcal{F}}{\overset{\sigma^*}{\text{Opt}}}[l^*(f(X),Y) + \sum_{i=1}^{n} p_i(f,X) + \text{confidence}(p_i)(f,X)] \tag{8}$$

While it's hard to imagine proving anything about guarantees of the above converging, as there are no equations for $\mathcal{F}$, σ^*, or l^*, it's easy to imagine with enough experience training such systems, we could take great confidence in the results. This is not to say there is no room for securing some properties, such as for example that f^* will be deterministic, or will fail if it becomes too powerful, in the hardware itself. The success of these sort of properties is entirely allowed by the fact that the computation hypothesis is known to fail along particular directions. But these techniques do not scale in their efficacy with the compute of the system - in fact they do the opposite. And thus exploration of what properties p_i should be used in the above formula is in some sense even more important than the development of fail-safes.

One interesting observation is that a good candidate for an optimal solution f^* to Equation 8 would be a system that develops an *internal* mathematics. That is, in addition to solving the primary task, it would develop an expressive, composable system of properties. Such a system would have a strong, data/compute efficient, introspection ability, and would potentially be easy to extend with an interface between this introspection ability and the external user. Put somewhat ironically, if we accept The Computation Hypothesis, giving the AI the ability to communicate and asking it not to eat us is simply a much more scalable approach than any amount of fail-safes and external mathematical analysis.

4.2 The competition between math and computation

The Computation Hypothesis may be shortened to be the hypothesis that computation is useful. It may thus at first seem strange that computation being useful in a system should imply that mathematics is not useful. But the competitive relationship is more clear when viewing the converse: the usefulness of mathematics implies that computation is unnecessary. If we can precompute the result, f^*, of an optimization problem using some sort of mathematical trickery, then it is no surprise that computation is useless. The possibility of using math to find the one millionth digit of pi or a prime greater than 10^{100} runs in direct competition to the possibility of raw, dense computation being useful and appropriate for these tasks. This is perhaps one of the reasons that math can be so interesting - it provides serendipitous conclusions that the hard work of computation is unnecessary, and that we can get results for free. If we could prove mathematically exactly the settings under which computation is not useful, we would have a recipe for generating all proofs, and thus we must be willing to merely accept the usefulness of computation as a hypothesis.

It is further worth noting that it is the goal of optimizing over optimization algorithms that we would eventually run into an optimization algorithm that would look at it's inputs, analyze it's properties, and simply prove the intended result mathematically rather than expending computation. In this sense, we see that math may not be worth applying to sufficiently complex systems because these systems develop all the incentive for developing math themselves. This is especially true if the systems have more computational resources than the human mind, and the "general intelligence" necessary to use these resources effectively.

5 Conclusion

We have summarized some of the intuition for the idea that mathematics may be less important to advances in intelligence than it has in the past. What we are seeing in deep learning is not the end of mathematics, but rather the passing of the responsibility for understanding the system away from the optimization process of the human user and towards the optimization process of the machine.