Theoretical Thinking I
CSCI 670 - Advanced Analysis of Algorithms
PhD Program in Computer Science

Kleanthis Avramidis
USC ID: 2680-5898-81
Date: 10/29/2021, Fall

1 Introduction

Machine Learning algorithms have managed to prevail in contemporary Computer Science research, as they are capable of providing robust models of various forms of rational intelligence. Neural Networks, in specific, show dominant performance in most experimental settings, something related to the fact that they are universal function approximators. Of course, not all architectures can efficiently approach every function. Intuitively though, one could expect that networks that are structured in a similar way to the target function, would be able to approximate it efficiently and generalize easier, which is the ultimate goal of a learning algorithm. An interesting extension of this notion is to consider Neural Network architectures that could approximate classical computer science algorithms for reasoning tasks. Indeed, Neural Networks have been successful in modeling reasoning tasks, however most of them come with a carefully configured structure. In this report we discuss the underlying principles behind this observation, as formulated theoretically and empirically by Xu et al. [1] in their ICLR 2020 publication "*What Can Neural Networks Reason About?*". I came over this while searching for applications of Graph Neural Networks (GNNs).

2 Theoretical Framework

Authors build upon the observation that reasoning processes resemble algorithms, hence if such an algorithm aligns well with the computation graph of the network, then the network only needs to learn simple functions to simulate the reasoning process. For instance, the GNN matches the structure of the Bellman-Ford algorithm, so its MLP modules only need to learn a simple update equation to simulate it. In contrast, a single MLP would need to simulate an entire for-loop, which is complex. Therefore, we expect the GNN to have better sample complexity than MLP when learning to solve shortest path problems. The above intuition is formalized using PAC Learning, which considers simplicity as sample complexity, i.e., the number of samples needed to ensure low test error with high probability:

Theorem 1 (**PAC Learning - Sample Complexity**). *Fix an error parameter $\epsilon > 0$ and failure probability $\delta \in (0, 1)$. Suppose $\{x_i, y_i\}_{i=1}^M$ are i.i.d. samples from some distribution $\mathcal{D}$, and the data satisfies $y_i = g(x_i)$ for some underlying function g. Let $f = \mathcal{A}\left(\{x_i, y_i\}_{i=1}^M\right)$ be the function generated by a learning algorithm $\mathcal{A}$. Then g is (M, ϵ, δ)-learnable with $\mathcal{A}$ if $\mathbb{P}_{x\sim\mathcal{D}}[\|f(x) - g(x)\| \leq \epsilon] \geq 1 - \delta$.*

The sample complexity $\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta)$ is the minimum M so that g is (M, ϵ, δ)-learnable with $\mathcal{A}$. By introducing the PAC learning framework, we can now define a numeric measure of algorithmic alignment:

Theorem 2 (**Algorithmic Alignment**). *Let g be a reasoning function and $\mathcal{N}$ a neural network with n modules $\mathcal{N}_i$. The module functions $f_1, \ldots, f_n$ generate g for $\mathcal{N}$ if, by replacing $\mathcal{N}_i$ with f_i, the network $\mathcal{N}$ simulates g. Then $\mathcal{N}(M, \epsilon, \delta)$-algorithmically aligns with g if a) $f_1, \ldots, f_n$ generate g and b) there are learning algorithms $\mathcal{A}_i$ for the $\mathcal{N}_i$'s such that $n \cdot \max_i C_{\mathcal{A}_i}(f_i, \epsilon, \delta) \leq M$.*

Good algorithmic alignment, i.e., small M, implies that all algorithm steps f_i to simulate the algorithm g are easily learnable. This can be further quantified by computing M. Authors indicatively describe this process for the case of MLPs, suggesting that functions that are "simple", when expressed as a polynomial, are sample efficiently learnable by an MLP. Besides that, they also provide a preliminary demonstration of sample complexity bound increasing with M, so algorithmic alignment can lead to generalization. As an illustrative example, it is proved that GNNs have a polynomial improvement in sample complexity over MLPs when learning simple relations. The Theorem states the following, relying though on three strong assumptions, namely algorithm stability, sequential type of learning and Lipschitzness:

Theorem 3. *Fix ϵ and δ. Suppose $\{S_i, y_i\}_{i=1}^M \sim \mathcal{D}$, where $|S_i| < N$, and $y_i = g(S_i)$ for some g. Suppose $\mathcal{N}_1, \ldots, \mathcal{N}_n$ are network $\mathcal{N}'$'s MLP modules in sequential order. Suppose $\mathcal{N}$ and $g(M, \epsilon, \delta)$-algorithmically align via functions $f_1, \ldots, f_n$. Under the considered assumptions, g is $(M, O(\epsilon), O(\delta))$-learnable by $\mathcal{N}$.*

3 Empirical Results

Authors apply their theoretical framework to analyze 3 types of neural networks, namely Multilayer Perceptrons (MLP), Deep Sets (DS) of the form $y = \mathrm{MLP}_2\left(\sum_{s \in S} \mathrm{MLP}_1\left(X_s\right)\right)$ and Graph Neural Networks. GNNs follow a message passing scheme, where the representation $h_s^{(k)}$ of each node s in iteration k is recursively updated by aggregating neighboring nodes. They can thus be adopted for reasoning by considering objects as nodes and assuming all object pairs are connected in a complete graph (MLP). Using algorithmic alignment, authors predict and verify whether each of these models can generalize on four increasingly complex reasoning tasks: summary statistics, relational argmax, dynamic programming, and an NP-hard problems. Below we provide an explanation of the results in Figure 3 of the paper.

- **Summary Statistics**: Deep Sets can learn to compute max or min of a feature by using smooth approximations like softmax and aggregate using their pooling layer. In contrast, an MLP must learn a complex for-loop and therefore does not align well. Empirical results on the task of finding maximum value difference verify this analysis. Interestingly, when items are sorted, the MLP tops performance, since it now learns only a trivial algorithm of a direct subtraction.
- **Relational Argmax**: Here, tasks like finding the farthest pair of same-color objects, require comparison of pairwise relationships/distances to provide an answer. As verified by the experimentation, one-iteration GNN aligns well, as it sums over all pairs of objects, whereas Deep Sets fail, since most pairwise relations cannot be encoded as a sum of individual objects.
- **Dynamic Programming**: A broad class of relational reasoning tasks can be approached using dynamic programming (DP). DP recursively breaks down a problem into simpler sub-problems:

  $$\text{Answer }[k][i] = \text{DP-Update}(\{\text{Answer }[k-1][j]\}, j = 1 \ldots n)$$

  where Answer$[k][i]$ is the solution to the sub-problem of iteration k and state i. We can interpret GNN as a DP algorithm, where node representations $h_i^{(k)}$ are Answer $[k][i]$, and the GNN aggregation step is the DP-Update. Therefore, a GNN with enough iterations can sample efficiently learn any DP algorithm with a simple DP-update function. Authors here experiment with Shortest Paths through a monster trainer game, a standard DP problem to be solved by the Bellman-Ford algorithm. Again, the theoretical framework is confirmed and with fewer iterations than Bellman-Ford would do.
- **NP-Hard Problems**: NP-hard problems cannot be solved by DP, neither by a GNN. The importance of the derived framework however is that we can design a network with a similar structure to the problem. Since most such problems are solved just using exhaustive search, authors designed the Neural Exhaustive Search (NES) network. At the subset sum problem, NES enumerates all subsets of objects and passes each subset through an LSTM followed by a pooled MLP. This network aligns well with the task, since the first MLP and LSTM only need to check whether a subset has zero sum. Indeed, NES yields 98% on the subset sum, while other models fail to approximate a solution.

4 Future Work

The particular paper is an initial step towards comprehending neural network structure for reasoning tasks and introducing modern computational tools to algorithm design. It would be interesting to utilize the theoretical framework of algorithmic alignment to model algorithmic techniques other than dynamic programming, like greedy algorithms or divide-and-conquer methods. To this end, one could build up a network architecture from scratch, based on the notion of algorithmic alignment, ore even reverse-engineer neural network structures that are empirically shown to be efficient in such tasks. A future direction that I find most appealing is that of approaching neural architectures to solve or approximate NP-Hard problems.

References

[1] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Kenichi Kawarabayashi, and Stefanie Jegelka, What can Neural Networks Reason About? In *Proceedings of ICLR*, 2020.