ECE472 – Quiz 10

Jonathan Lam

December 2, 2020

1. *What do you think is preventing main stream usage of these types of architectures?* (arXiv:1410.5401, arXiv:1808.00508, *Nature*, 538(7626):471–476)

   I believe the reason for this is a fairly straightforward one: there is no practical domain for NTMs (DNCs) or NALUs at the current moment where they perform better than any other method (deep architecture or otherwise). These models' relevance is mostly one of theoretical interest; in particular, the NTM's ability to turn discrete memory operations into differentiable operations (by some fuzzy attention mechanism), the DNC's more advanced memory operations, and the generalizing ability for NTMs (by hardcoding in some common numerical functions).

   Then, what could be some practical use-cases for these systems? The authors of the NTM and DNC models use their models to evaluate common simple problems, such as copying memory, sorting, and using graph structures. Many of the mainstream models we look at achieve SOTA performance in some domain, as we've seen with ResNets (training deep networks), transformers (speech/text models like GPT-3), and MobileNets (small computational and size resources). In the case of the NTM, this would either mean SOTA performance in existing algorithms, or some efficient algorithm for previously unsolved problems.

   The problem I see with this is by attempting to recreate a more general problem-solving machine, we lose the ability to perform very well in any given task. (The opposite is the case for NALUs, which I discuss below.) When many of these simple algorithms require speed, accuracy, and have memory or hardware constraints, a neural network is hardly the best tool for the job. If we have errors in even simple tasks like copying and repeating a piece of data a given number of times, when the corresponding algorithm is trivially simple and optimally fast, then it is not suitable for a task. One potential solution would be to encode some information about the problem in the network, but this clearly does not generalize to other tasks. Another potential "solution" is to use NTMs on higher-level problems, e.g., distributed computing or a logic constraint checker; however, given the accuracy of the NTM on low-level problems, we can hardly expect anything out of accuracy, speed (computational performance), or memory efficiency. At which point the best we might be able to do is

equip the NTM "controller" with higher-level differentiable operations in addition to read/write heads: differentiable data structures, differentiable API calls, etc. This would require a very tiresome method to produce what would probably be at best heuristic results, by coding some human knowledge into the program (efficient algorithms); but this is the price of attempting to make a more generalized problem-solving machine.

(Of course, this is assuming that the end goal is to create some sort of generalized problem-solving machine. There may be some interesting and practical use-case for differentiable memory operations that I can't fathom.)

On the other hand, NALUs are the opposite – they encode human knowledge of a problem into a structure dedicated to the problem at hand. While this allows for generalization in the domain of arithmetic operations, which is the authors' stated goal, these NALUs are likely unsuitable for any other task. Moreover, there is no task that the NALU can perform that a regular ALU cannot perform much more efficiently. Thus it is difficult to find a suitable use case for NALUs (it is unfortunate that the authors also do not provide any sort of motivation for NALUs other than generalization for learning arithmetic operations).

tl;dr: These can't approach SOTA (e.g., hardware and handcoded low-level algorithms) in their domains, even if they can beat other deep neural architectures, and it doesn't seem that any other practical applications of these architectures have been discovered yet.