Tensorflow

Dreycey Albin

September 2019

The key need for Tensorflow was a system that allowed for training and deploying neural networks. While there were already other systems available at the time Tensorflow was released, the current systems available at the time did not deliver both an easy to use interface as well as control over the neural network structure [1].

The key need for Tensorflow was a system that overcame the limitations of a *parameter server* architecture, such as in Disbelief. The parameters in a *parameter server* are weight matrices that linearly transform input vectors from each node, and thereafter, the output vectors are sent through a non-linear function (such as a sigmoid). The authors note three ways in which they hoped to improve upon the previous Distbelief system: (1) Defining new layers; (2) Refining the training algorithms; (3) Defining new training algorithms. These challenges were overcome in Tensorflow, and a distributed computing abstraction was implemented using a dataflow graph. This also gives users the ability to compose novel layers through a high level scripting interface, overcoming the specific challenge listed in (1) [1].

The authors mention a limitation that earlier users found: the static representation of a dataflow graph is limiting. This brought forward some drawbacks when it came to deep reinforcement learning (again, as noted by the beta version users). In addition, the comparison to other neural network abstraction systems in Table 1 shows mixed results. Without considering usability, or flexibility, it looks as though Neon outperforms both Torch and Tensorflow. The authors attribute this to the cuDNN library used by both Torch and Tensorflow (https://developer.nvidia.com/cudnn), as most of Neon was written in house using assembly language. Therefore, it is not clear what system is the best to use for training and deploying neural networks [1]. (Neon is also mentioned in the separate paper citing Tensorflow [2])

An area of the paper that interests me is the dataflow graph. I do not quite fully understand this yet, but it is a very interesting concept. I find two things interesting about the dataflow graph: (1) how edges represent tensors across the nodes, and (2) How this representation is converted into GPU OR CPU level instructions. As for the first the first interest, this model creates a simple representation for a neural networks, and it overall makes sense for the edges to represent tensors (and I think this was also used in Distbelief). As for the system level operations, I think it's interesting how "engineered" Tensorflow was before delivery- where it can be used on clusters or smart phones, on CPUs or GPUs, ect. [1].

The paper by Min et al. describes different ways that deep learning is being used in bioinformatics. This review article was comprehensive, going over several of the background concepts of deep learning before jumping into the applications. One interesting concept I learned from reading this review was how many different neural network architectures exist, and current ways that deep neural networks are being applied to bioinformatics. The article described three different areas where deep neural networks have made a major push in

bioinformatics: (1) Omics; (2) Biomedical Imaging; and (3) Biomedical Signal Processing. In addition, one area of the paper that caught my eye is the idea of neural networks being a "black box", as the review paper described methods that are being used to make sense of the models for the biomedical field [2].

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics. *Briefings in bioinformatics*, 18(5):851–869, 2017.