WebNN: A Distributed Framework for Deep Learning

Aaron Goin
School of Engineering and Computer Science
Washington State University
aaron.goin@wsu.edu

Ronald Cotton
School of Engineering and Computer Science
Washington State University
ronald.cotton@wsu.edu

Xinghui Zhao
School of Engineering and Computer Science
Washington State University
x.zhao@wsu.edu

ABSTRACT

Distributed computing technologies have been used to facilitate machine learning applications, so that high-end hardware resources, such as clusters, GPUs, etc., can be leveraged to achieve high performance. In this paper, take a different approach to make deep learning framework more accessible. We present WebNN, a web-based distributed framework for deep learning. WebNN enables users to configure, distribute, and train neural networks asynchronously over the Internet, utilizing peer-owned resources with increased flexibility. Experiments have been carried out to evaluate WebNN, and the results show that WebNN is a effective solution for distributed training of models using pre-labeled batches, or client-provided data.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Distributed algorithms**;

KEYWORDS

Deep Learning, Neural Networks, Distributed Computing

ACM Reference Format:
Aaron Goin, Ronald Cotton, and Xinghui Zhao. 2018. WebNN: A Distributed Framework for Deep Learning. In *Proceedings of 47th International Conference on Parallel Processing (ICPP'18).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

There are a myriad of tools and frameworks available for training neural networks, among which Google's TensorFlow [1] has rapidly become the most well known and widely used framework. Most recently, Google added TensorFlow.js to its framework–bringing machine learning to the browser. TensorFlow.js is inspired by TensorFlow's Keras in its API, and it leverages WebGL to accelerate execution of neural networks on the client's GPU.

While TensorFlow.js does run in the browser, it's a client-side technology that requires additional services on the back-end to distribute trained models to clients, or to train a central model that all clients can execute, train, and share. To this end, we have created a system called WebNN to facilitate distributing and training models in the web browser. WebNN also allows users to deploy their models in real web applications, and have their clients train the model with their own private data, eliminating the need for creating large, pre-labeled training data. WebNN is open sources on Github[1].

ICPP'18, August 2018, Eugene, OR, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 WEBNN ARCHITECTURE

To make it easy to interface with the front-end, we employ JavaScript (ES6+) through NodeJS. WebNN Server builds off of NodeJS to serve the model over http. Also supported by NodeJS is a small tool "wnn.js" which provides some convenient command-line controls for WebNN. WebNN can be deployed as a standalone server, or applied to an existing application as a service. Figure 1 shows the architecture of the WebNN framework.

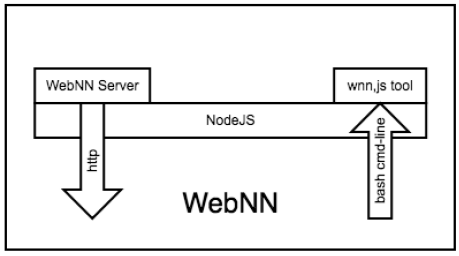

Figure 1: WebNN Architecture

Users create their models in a JSON format, configure it training and validation properties, and create a JavaScript module which the server will use to get training and validation data. The server will hand the model off to clients for training, along with a set of weights, and training data upon request. Clients can send back their modified weights to the server, and receive a new set of weights (from another client) to merge into their own.

A challenge in WebNN is how to merge the client generated weights in an effective way. Distributed TensorFlow handles these so-called stale weights by simply dropping them, and not merging them at all. This is an acceptable

[1]WebNN is open sourced at https://github.com/aarongoin/WebNN

strategy when training over a central dataset, but this is unacceptable when clients are potentially training with their own data that a central model cannot redistribute and train on. Inspared by [3] to we take a different approach to scale stale weights based on how stale they are.

To enable web-scale applications, we offload merging to our clients, and the server simply facilitates swapping weights between peers. The server still retains a copy of all outstanding weights (weights that have been sent to a client who has not yet updated the server), so weights are never in jeopardy of being lost. And the server can be made to merge all weights together before sending them to be validated if needed. To this end, we implemented three different methods a client could use to merge the incoming weights with their own:

- **Average merge:** weights are simply averaged together regardless of potential staleness;
- **Weighted merge:** every time a client trains, its time step variable is incremented. This step variable is always kept with the weights, and weighted merge scales all weights by their time step to favor less stale weights.
- **Mimic merge:** uses the same information as the weighted merge, but handles it differently. Any average is going to be bounded between the values themselves. Rather than averaging the weights, the mimic merge determines which set of weights is less stale, determines the difference between those weights and the other set, and then moves the current weights in the same direction. To avoid erratic oscillations, the weight adjustment is scaled according to how far apart the two weights are.

3 EXPERIMENTAL RESULTS

We have evaluated our merge functions using MNIST dataset[2]. Specifically, we keep training on the model until the 5-second running average of the validation accuracy is greater than or equal to 98%. We set our model to validate once every second, set our mini-batch size to 64 samples, and set our learning rate to 0.2. To compare between merge functions, we trained our model with each merge function and 4 clients. We did this 4 times for each merge function. Both the server and the clients are run on iMacs with 2.5GHz i5 CPU, 8GB RAM and an AMD Radeon 6750M GPU.

In addition to the three merge functions in WebNN, in our experiments we added Copy merge as a baseline. This is a method which is effectively equivalent to training with a single client and is used as a benchmark. This merging method has the client pick whichever set is most accurate, and discards the other. When used in an environment where clients provide their own data: clients using copy merge would discard weights and lose valuable trained weights.

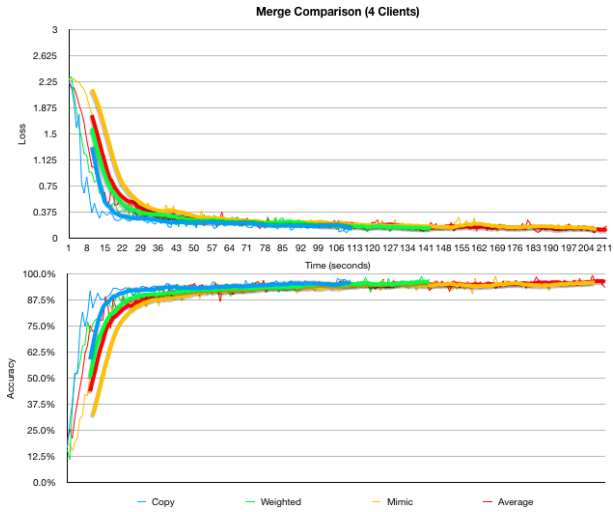

Figure 2: Comparison of Merge Functions

Figure 2 shows the loss and accuracy over time for each of the four merge functions. The weighted merge results in more accurate training in less time than the naive merge, demonstrating the potentials of the peer-based weight merge mechanism for distributed training.

4 CONCLUSION

WebNN is a web-based framework for distributing and training a centralized neural network in the browser. It enables users to easily deploy and train a neural network over a distributed system, therefore utilize peer-owned resources. Our experimental results show that a peer-based weight merge system works best with a weighted average favoring weights with more training iterations behind them. The peer-based merging we implemented can be improved to promote less variance between clients [2], which can potentially further improve the training performance.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. "TensorFlow: A system for large-scale machine learning". In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA.*

[2] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormándi, George E. Dahl, and Geoffrey E. Hinton. 2018. Large scale distributed neural network training through online distillation. *CoRR* abs/1804.03235 (2018). arXiv:1804.03235 http://arxiv.org/abs/1804.03235

[3] Ji Liu Xiangru Lian Suyog Gupta Wei Zhang. 2016. Staleness-Aware Async-SGD for Distributed Deep Learning. *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* 25 (2016), 2350–2356.

[2]Dataset is availalbe at https://js.tensorflow.org/tutorials/mnist.html