Distributed Learning of Neural Networks with One Round of Communication

Mike Izbicki[1] and Christian R. Shelton[2]

[1] Claremont McKenna College, Claremont, CA, USA
mike@izbicki.me

[2] UC Riverside, Riverside, CA, USA
cshelton@cs.ucr.edu

Abstract. The *optimal weighted average* (OWA) is an algorithm for distributed learning of linear models. It achieves statistically optimal theoretical guarantees with only a single round of communication [3]. This paper introduces the *non-linear OWA* (NOWA) algorithm, which extends the linear OWA into the non-linear setting of neural networks. Due to the difficulty of proving theoretical results in this more complex setting, NOWA loses the theoretical guarantees of the OWA algorithm. Nevertheless, we show that NOWA works well empirically. We follow an evaluation procedure introduced by McMahan et. al. [16] for federated learning and show significantly improved results on a simple MNIST baseline task.

1 Introduction

Existing distributed learning algorithms fall into one of two categories:

Interactive algorithms require many rounds of communication between machines. Representative examples include [11, 4, 13, 22, 18, 7]. The appeal of interactive algorithms is that they enjoy the same statistical performance as standard sequential algorithms. But, interactive algorithms have three main disadvantages. First, these algorithms are slow when communication latency is the bottleneck. An extreme example occurs in the *federated learning* environment proposed by [16], which uses cell phones as the computational nodes. Recent work in this setting has studied how to only communicate between nodes when doing so would proveably decrease loss [7]. Second, these algorithms require special implementations. They are not easy for non-experts to implement or use, and in particular they do not work with off-the-shelf statistics libraries provided by (for example) Python, R, and Matlab. Third, because of the many rounds of communication, any sensitive information in the data is likely to leak between machines.

Non-interactive algorithms require only a single round of communication. Each machine independently solves the learning problem on a small subset of data, then a master machine merges the solutions together. These algorithms solve all the problems of interactive ones: they are fast when communication

is the main bottleneck; they are easy to implement with off-the-shelf statistics packages; and they are robust to privacy considerations. The downside is worse statistical performance. A growing body of work analyzes the popular naive averaging merging procedure under special conditions [14, 20, 21, 17, 19], and develops more robust merging procedures [23, 12, 10, 1, 5, 6]. All of these estimators are either statistically sub-optimal or have computationally intractable merge procedures.

The *optimal weighted average* (OWA) [3, 2] is a recently proposed non-interactive estimator with statistically optimal guarantees. OWA's merge procedure uses a second round of optimization over a tiny fraction of the data. Because the fraction of data is small, it presents negligible computational burden, but OWA is still able to achieve the optimal sequential statistical error rates in the non-interactive setting. The downside of OWA is that it only works for linear models. In this paper, we develop an algorithm called NOWA that extends OWA into the nonlinear setting. The next section introduces OWA in the original linear setting, and then Section 3 describes the NOWA extension. Section 4 shows preliminary experiments with NOWA on the MNIST dataset. We see that the standard naive averaging algorithm commonly used in federated learning performs significantly worse in this simple task than NOWA.

2 Warmup: The Linear OWA

2.1 Problem Statement

Let $\mathcal{Y} \subseteq \mathbb{R}$ be the space of response variables, $\mathcal{X} \subseteq \mathbb{R}^d$ be the space of covariates, and $\mathcal{W} \subseteq \mathbb{R}^d$ be the parameter space. We assume a linear model where the loss of data point $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$ given the parameter $W \in \mathcal{W}$ is denoted by $\ell(y, \mathbf{x}^\top W)$. We define the true loss of parameter vector W to be $\mathcal{L}^*(W) = \mathbb{E}\,\ell(y; \mathbf{x}^\top W)$, and the optimal parameter vector $W^* = \arg\min_{W \in \mathcal{W}} \mathcal{L}^*(W)$. We do not require that the model be correctly specified, nor do we require that ℓ be convex with respect to W. Let $Z \subset \mathcal{X} \times \mathcal{Y}$ be a dataset of mn i.i.d. observations. Finally, let $r : \mathcal{W} \to \mathbb{R}$ be a regularization function (typically the L1 or L2 norm) and $\lambda \in \mathbb{R}$ be the regularization strength. Then the regularized empirical risk minimizer (ERM) is

$$\hat{W}^{erm} = \arg\min_{W \in \mathcal{W}} \sum_{(\mathbf{x},y) \in Z} \ell(y, \mathbf{x}^\top W) + \lambda r(W). \tag{1}$$

Assume that the dataset Z has been partitioned onto m machines so that each machine i has dataset Z_i of size n, and all the Z_i are disjoint. Then each machine calculates the local ERM

$$\hat{W}_i^{erm} = \arg\min_{W \in \mathcal{W}} \sum_{(\mathbf{x},y) \in Z_i} \ell(y, \mathbf{x}^\top W) + \lambda r(W). \tag{2}$$

Notice that computing $\hat{W}_i^{erm}$ requires no communication with other machines. Our goal is to merge the $\hat{W}_i^{erm}$s into a single improved estimate.

To motivate our OWA merge procedure, we briefly describe a baseline procedure called *naive averaging*:

$$W^{ave} = \frac{1}{m}\sum_{i=1}^{m} \hat{W}_i^{erm}. \tag{3}$$

Naive averaging is simple to compute but has only limited theoretical guarantees. Recall that the quality of an estimator $\hat{W}$ can be measured by the estimation error $\|\hat{W} - W^*\|_2$, and we can use the triangle inequality to decompose this error as

$$\|\hat{W} - W^*\|_2 \leq \|\hat{W} - \mathbb{E}\,\hat{W}\|_2 + \|\mathbb{E}\,\hat{W} - W^*\|_2. \tag{4}$$

We refer to $\|\hat{W} - \mathbb{E}\,\hat{W}\|_2$ as the variance of the estimator and $\|\mathbb{E}\,\hat{W} - W^*\|_2$ as the bias. McDonald et al. [14] show that the W^{ave} estimator has lower variance than the estimator $\hat{W}_i^{erm}$ trained on a single machine, but the same bias. Zhang et al. [20] extend this analysis to show that if $\hat{W}_i^{erm}$ is a "nearly unbiased estimator," then naive averaging is optimal. But Rosenblatt and Nadler [17] show that in high dimensional regimes, all models are heavily biased, and so naive averaging is suboptimal. All three results require ℓ to be convex in addition to other technical assumptions. The OWA estimator relaxes these assumptions and achieves better error bounds.

2.2 The Full OWA

To motivate the OWA estimator, we first present a less efficient estimator that uses the full dataset for the second round of optimization. Define the matrix $\hat{W} : \mathbb{R}^{d\times m}$ to have its ith column equal to $\hat{W}_i^{erm}$. Now consider the estimator

$$\hat{W}^{owa,full} = \hat{W}\hat{V}^{owa,full}, \tag{5}$$

where

$$\hat{V}^{owa,full} = \operatorname*{arg\,min}_{V\in\mathbb{R}^m} \sum_{(\mathbf{x},y)\in Z} \ell\left(y, \mathbf{x}^\top \hat{W}V\right) + \lambda r(\hat{W}V). \tag{6}$$

Notice that $\hat{W}^{owa,full}$ is just the empirical risk minimizer when the parameter space $\mathcal{W}$ is restricted to the subspace $\hat{\mathcal{W}}^{owa} = \text{span}\{\hat{W}_i^{erm}\}_{i=1}^m$. In other words, the $\hat{V}^{owa,full}$ vector contains the optimal weights to apply to each $\hat{W}_i^{erm}$ when averaging. Figure 1 shows graphically that no other estimator in $\hat{\mathcal{W}}^{owa}$ can have lower regularized empirical loss than $\hat{W}^{owa,full}$.

2.3 The OWA Estimator

The OWA estimator uses fewer data points in the second round of optimization. Recall that in a linear model, the amount of data needed is proportional to the problem's dimension. Since the dimension of the second round is a fraction m/d smaller than the first round, only an m/d fraction of data is needed for the same accuracy.

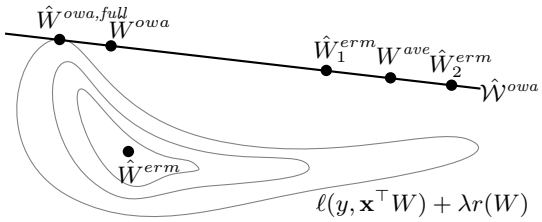

Fig. 1. $\hat{W}^{owa,full}$ is the estimator with best loss in $\hat{\mathcal{W}}^{owa}$, and $\hat{W}^{owa}$ is close with high probability.

Formally, let Z^{owa} be a set of m^2n/d additional data points sampled i.i.d. from the original data distribution. Thus the total amount of data the OWA estimator requires is $mn+m^2n/d$. Whenever $m/d \leq 1$, this expression simplifies to $O(mn)$, which is the same order of magnitude of data in the original problem. The OWA estimator is then defined as

$$\hat{W}^{owa} = \hat{W}\hat{V}^{owa}, \tag{7}$$

where

$$\hat{V}^{owa} = \operatorname*{arg\,min}_{V \in \mathbb{R}^m} \sum_{(\mathbf{x},y) \in Z^{owa}} \ell\left(y, \mathbf{x}^\top \hat{W} V\right) + \lambda r(\hat{W}V). \tag{8}$$

OWA's merge procedure is more complicated than the naive averaging merge procedure, but still very fast. Notice that the projected data points $\mathbf{x}^\top \hat{W}$ have dimensionality $m \ll d$, and there are only m^2n/d of them. Because the optimization uses a smaller dimension and fewer data points, it takes a negligible amount of time. Izbicki and Shelton [3] show an experiment where the first round of optimizations takes about a day, and the second optimization takes about a minute.

3 The Non-linear OWA (NOWA)

The intuition of the NOWA algorithm is that we apply the OWA algorithm to each layer of a neural network independently. Unfortunately, the notation is much messier in this scenario due to the need to keep track of many indices.

3.1 Problem Setting

We now extend our notation to include neural networks with multiple hidden layers. In particular, we continue to use subscripts to denote different machines (and let i range over the machines), but we also introduce superscripts to denote different network layers (and let j range over the layers).

Formally, assume our network architecture has p layers. For each layer $j \in \{1, ..., p\}$, there is an associated dimension $d^{(j)} \in \mathbb{N}$, activation function $\sigma^{(j)} : \mathbb{R}^{d^{(j)}} \to \mathbb{R}^{d^{(j)}}$, and weight matrix $W^{(j)} : \mathbb{R}^{d^{(j)} \times d^{(j-1)}}$. The input to the network is a vector $\mathbf{x} \in \mathbb{R}^{d^{(0)}}$. The output of layer j is then recursively given by

$$f^{(j)}(\mathbf{x}) : \mathbb{R}^{d^{(j)}} = \begin{cases} \mathbf{x} & j = 0 \\ \sigma^{(j)}(W^{(j)} f^{(j-1)}(\mathbf{x})) & j > 0 \end{cases} \tag{9}$$

and $f^{(p)}(\mathbf{x})$ is the final output of the network. In supervised learning problems, we are given a dataset $Z \subset \mathbb{R}^{d^{(0)}} \times \mathbb{R}^{d^{(p)}}$ with mn data points, and our goal is to solve

$$\hat{W}^{erm} = \arg\min_{W} \sum_{(\mathbf{x},y) \in Z} \ell(y, f^{(p)}(\mathbf{x})) + \lambda r(W), \tag{10}$$

where ℓ is the loss function and r is the regularization function. We divide Z into m disjoint smaller datasets $\{Z_1, ..., Z_m\}$ each with n points. Each dataset Z_i is transfered to processor i, which solves the local learning problem

$$\hat{W}_i^{erm} = \arg\min_{W} \sum_{(\mathbf{x},y) \in Z_i} \ell(y, f^{(p)}(\mathbf{x})) + \lambda r(W). \tag{11}$$

Each machine solves (11) without communicating with other machines using any optimizer appropriate for the network architecture and data. Our goal is to develop a merge procedure that combines the W_a local parameter estimates into a single global parameter estimate with small loss.

3.2 The Merge Procedures

In this non-linear setting, the naive averaging merge procedure for the jth layer is given by

$$W^{ave,(j)} = \frac{1}{m} \sum_{i=1}^{m} \hat{W}^{erm,(j)}{}_i. \tag{12}$$

Google's recent federated learning architecture uses naive averaging to merge models together that have been independently trained on users' cellphones [16].

We will define an improved merge procedure based on a weighted average of the local parameter estimates. This requires some tensor notation. for each layer j in the network, we define the 3rd-order tensor $W^{stacked(j)} : \mathbb{R}^m \times \mathbb{R}^{d^{(j)}} \times \mathbb{R}^{d^{(j-1)}}$, where the (a, b, c)th component of $W^{stacked(j)}$ is defined to be the (b, c)th component of $\hat{W}^{erm,(j)}{}_i$. In words, $W^{stacked(j)}$ is the 3rd-order tensor constructed by stacking the local parameter estimates $\hat{W}^{erm,(j)}{}_a$ along a new axis. We also define the function $\texttt{contract} : (\mathbb{R}^m, \mathbb{R}^m \times \mathbb{R}^{d^{(j)}} \times \mathbb{R}^{d^{(j-1)}}) \to \mathbb{R}^{d^{(i)}} \times \mathbb{R}^{d^{(j-1)}}$ to be the tensor contraction along the first dimension. That is, if $V : \mathbb{R}^m$, then the (b, c)th component of $\texttt{contract}(V, W^{stacked(j)})$ is equal to $\sum_{a=1}^{m} V(a) \hat{W}^{erm,(j)}{}_a(b, c)$. In particular, if each component of V equals $1/m$, then $\texttt{contract}(V, W^{stacked(j)}) = \frac{1}{m} \sum_{a=1}^{m} \hat{W}^{erm,(j)}{}_a = W^{ave,(j)}$.

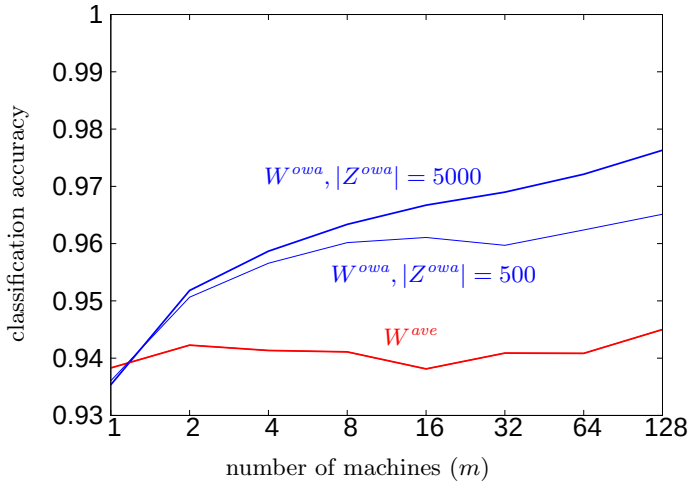

Fig. 2. The performance of the naive averaging estimator used in McMahan et al. [16] is constant as we add more machines, but the performance of the NOWA estimator increases.

In our *non-linear optimal weighted average* (NOWA) merge procedure, we first construct the modified neural network

$$f^{mod,(j)}(\mathbf{x}) : \mathbb{R}^{d^{(j)}} = \begin{cases} \mathbf{x} & j = 0 \\ \sigma^{(j)}(W^{mod,(j)} f^{mod,(j-1)}(\mathbf{x})) & j > 0 \end{cases} \tag{13}$$

where

$$W^{mod,(j)} = \texttt{contract}(V^{(j)}, W^{stacked^{(j)}}). \tag{14}$$

We then select a small subset of the data Z^{owa} (i.e. $|Z^{owa}| \ll |Z|$) and train the network f^{mod} over only the parameters $V^{(j)}$. That is, we solve the optimization problem

$$V^{owa} = \arg\min_{V} \sum_{(\mathbf{x},y) \in Z^{owa}} \ell(y, f^{mod,(p)}(\mathbf{x})) + \lambda r(V). \tag{15}$$

The parameter matrices $W_i^{owa} = \texttt{contract}(V^{owa,(j)}, W^{stacked^{(j)}})$ can then be used in the original neural network. Intuitively, we need only a small number of data points in the optimization of (15) because the number of parameters is significantly smaller than in the original optimization (10). That is, the dimension of V^{owa} is much less than the dimension of $\hat{W}^{erm}$. When the network contains no hidden layers, then the NOWA procedure reduces to the OWA procedure described above.

4 Experiments

McMahan et al. [16] evaluated the naive averaging merge procedure on the MNIST dataset, and we perform a similar experiment here. We train the LeNet neural network [9] provided by TensorFlow's standard tutorial using the Adam optimizer and dropout. We performed no hyperparameter tuning and simply used the default hyperparameters provided by TensorFlow.

We perform our experiment using a cluster of 128 machines. MNIST contains a training set 55,000 data points, and each machine receives a subset of the data containing either 429 or 430 data points. The 10 class labels are evenly distributed throughout the original training set, but we made no effort to ensure they were evenly distributed throughout the subsets. That means on average, each machine has access to only 43 examples from each class, but most machines will have significantly fewer examples for some classes. Under such an extreme paucity of data, it is unlikely for a single machine to be achieve high classification accuracy.

Figure 2 shows the classification accuracy as the number of machines used varies from 2 to 128. (Each experiment is repeated 5 times, and the average is shown.) Since the number of data points per machine is fixed, adding more machines adds more data, so we should expect the classification accuracy to increase for a good merge procedure. We see that the NOWA algorithm significantly outperforms naive averaging. The NOWA algorithm does not perform as well as the oracle network trained on all the data (which has > 0.99 accuracy). This is because of the difficulty of the local learning problems, which average only 42 instances of each class.

5 Discussion

The original papers on federated learning [8, 15, 16] perform several rounds of naive averaging to improve performance. In each round, the average from the previous round is used to initialize the optimization of each worker node. This procedure can easily be extended to use the NOWA merge procedure instead of naive averaging. Since NOWA's weighted averaging procedure performs better than naive averaging in a single round, a multi-round version of NOWA will likely perform better than a multi-round version of naive averaging. The second round of optimization used in NOWA is particularly negligible in the federated setting because this optimization can be performed in the data center on dedicated machines. Therefore, using NOWA in a federated setup would provide no additional burden to the node machines, which are typically severely computationally limited devices like cell phones.

References

1. Jun Han and Qiang Liu. Bootstrap model aggregation for distributed statistical learning. *NeurIPS*, 2016.

2. Mike Izbicki. *Divide and Conquer Algorithms for Faster Machine Learning.* PhD thesis, UC Riverside, 2017.
3. Mike Izbicki and Christian R. Shelton. Distributed learning of non-convex linear models with one round of communication. *ECML-PKDD*, 2019.
4. Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *NeurIPS*, 2014.
5. Michael I. Jordan, Jason D. Lee, and Yun Yang. Communication-efficient distributed statistical inference. *arXiv preprint arXiv:1605.07689*, 2016.
6. Michael Kamp, Mario Boley, Olana Missura, and Thomas Gärtner. Effective parallelisation for machine learning. In *NeurIPS*, 2017.
7. Michael Kamp, Linara Adilova, Joachim Sicking, Fabian Hüger, Peter Schlicht, Tim Wirtz, and Stefan Wrobel. Efficient decentralized deep learning by dynamic model averaging. In *ECML-PKDD*. Springer, 2018.
8. Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.
9. Yann LeCun et al. Lenet-5, convolutional neural networks.
10. Jason D Lee, Qiang Liu, Yuekai Sun, and Jonathan E Taylor. Communication-efficient sparse regression. *JMLR*, 18(5), 2017.
11. Mu Li, David G Andersen, and Jun Woo Park. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014.
12. Qiang Liu and Alexander T Ihler. Distributed estimation, information loss and exponential families. In *NeurIPS*, 2014.
13. Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I Jordan, Peter Richtárik, and Martin Takáč. Adding vs. averaging in distributed primal-dual optimization. *ICML*, 2015.
14. Ryan McDonald, Mehryar Mohri, Nathan Silberman, Dan Walker, and Gideon S Mann. Efficient large-scale distributed training of conditional maximum entropy models. In *NeurIPS*, 2009.
15. H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *CoRR*, 2016.
16. H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Samson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. 2017.
17. Jonathan D Rosenblatt and Boaz Nadler. On the optimality of averaging in distributed statistical learning. *Information and Inference*, 5(4), 2016.
18. Virginia Smith, Simone Forte, Ma Chenxin, Martin Takáč, Michael I Jordan, and Martin Jaggi. Cocoa: A general framework for communication-efficient distributed optimization. *JMLR*, 18, 2018.
19. Shusen Wang. A sharper generalization bound for divide-and-conquer ridge regression. *AAAI*, 2019.
20. Yuchen Zhang, Martin J Wainwright, and John C Duchi. Communication-efficient algorithms for statistical optimization. In *NeurIPS*, 2012.
21. Yuchen Zhang, John C Duchi, and Martin J Wainwright. Divide and conquer kernel ridge regression. In *COLT*, 2013.
22. Shen-Yi Zhao, Ru Xiang, Ying-Hao Shi, Peng Gao, and Wu-Jun Li. Scope: Scalable composite optimization for learning on spark. *AAAI*, 2017.
23. Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *NeurIPS*, 2010.