# Federated Learning: Strategies for Improving Communication Efficiency

**Jakub Konečný**[†*] **H. Brendan McMahan**[‡] **Felix X. Yu**[‡]
**Peter Richtárik**[†] **Ananda Theertha Suresh**[‡] **Dave Bacon**[‡]
[†]University of Edinburgh [‡]Google
J.Konecny@sms.ed.ac.uk, {mcmahan, felixyu}@google.com
peter.richtarik@ed.ac.uk, {theertha, dabacon}@google.com

## Abstract

Federated Learning is a machine learning setting where the goal is to train a high-quality centralized model with training data distributed over a large number of clients each with unreliable and relatively slow network connections. We consider learning algorithms for this setting where on each round, each client independently computes an update to the current model based on its local data, and communicates this update to a central server, where the client-side updates are aggregated to compute a new global model. The typical clients in this setting are mobile phones, and communication efficiency is of utmost importance. In this paper, we propose two ways to reduce the uplink communication costs. The proposed methods are evaluated on the application of training a deep neural network to perform image classification. Our best approach reduces the upload communication required to train a reasonable model by two orders of magnitude.

## 1 Introduction

As datasets grow larger and models more complex, machine learning increasingly requires distributing the optimization of model parameters over multiple machines. Existing machine learning algorithms are designed for highly controlled environments (such as data centers) where the data is distributed among machines in a balanced and i.i.d. fashion, and high-throughput networks are available. Federated learning [10, 14, 9] proposes an alternative setting, where we train a shared global model under the coordination of a central server, from a federation of participating devices. The participating devices (clients) are typically large in number and have slow or unstable internet connections. A motivating example for federated optimization arises when the training data is kept locally on users' mobile devices, and the devices are used as nodes performing computation on their local data in order to update a global model. The above framework differs from conventional distributed machine learning [18, 12, 20, 23, 5, 4] due to the the large number of clients, highly unbalanced and non-i.i.d. data and unreliable network connections. Federated learning offers distinct advantages compared to performing learning in the data center. The model update is generally less privacy-sensitive than the data itself, and the server never needs to store these updates. Thus, when applicable, federated learning can significantly reduce privacy and security risks by limiting the attack surface to only the device, rather than the device and the cloud. This approach also leverages the data-locality and computational power of the large number of mobile devices.

For simplicity, we consider synchronized algorithms for federated learning [14, 3], where a typical round consists of the following steps:

1. A subset of clients is selected, each of which downloads the current model.

[*]Work performed while at Google, Inc.

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

2. Each client in the subset computes an updated model based on their local data.
3. The updated models are sent from the selected clients to the sever.
4. The server aggregates these models (typically by averaging) to construct an improved global model.

A naive implementation of the above framework requires that each client sends a full model (or a full model gradient) back to the server in each round. For large models, this step is likely to be the bottleneck of federated learning due to the asymmetric property of internet connections: the uplink is typically much slower than downlink. The US average broadband speed was 55.0Mbps download vs. 18.9Mbps upload, with some ISPs being significantly more asymmetric, e.g., Xfinity at 125Mbps down vs. 15Mbps up [1]. It is therefore important to investigate methods which can reduce the uplink communication cost. In this paper, we study two general approaches:

- Structured updates, where we learn an update from a restricted lower-dimensional space.
- Sketched updates, where we learn a full model update, but then compress it before sending to the server.

These approaches can be combined, e.g., first learning a structured update and then sketching it; however, we do not experiment with this combination in the current work.

In the following, we formally describe the problem. The goal of federated learning is to learn a model with parameters embodied in a real matrix $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ from data stored across a large number of clients. We first provide a communication-naive version of the federated learning. In round $t \geq 0$, the server distributes the current model $\mathbf{W}_t$ to a subset $S_t$ of $n_t$ clients (for example, to a selected subset of clients whose devices are plugged into power, have access to broadband, and are idle). These clients independently update the model based on their local data. Let the updated local models be $\mathbf{W}_t^1, \mathbf{W}_t^2, \ldots, \mathbf{W}_t^{n_t}$, so the updates can be written as $\mathbf{H}_t^i := \mathbf{W}_t^i - \mathbf{W}_t$, for $i \in S_t$. Each selected client then sends the update back to the sever, where the global update is computed by aggregating[2] all the client-side updates:

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \eta_t \mathbf{H}_t, \qquad \mathbf{H}_t := \frac{1}{n_t} \sum_{i \in S_t} \mathbf{H}_t^i.$$

The sever chooses the learning rate $\eta_t$ (for simplicity, we choose $\eta_t = 1$). Recent works show that a careful choice of the server-side learning rate can lead to faster convergence [13, 12, 10].

In this paper, we describe federated learning for neural networks, where we use a separate 2D matrix $\mathbf{W}$ to represent the parameters of each layer. We suppose that $\mathbf{W}$ gets right-multiplied, i.e., $d_1$ and $d_2$ represent the output and input dimensions respectively. Note that the parameters of a fully connected layer are naturally represented as 2D matrices. However, the kernel of a convolutional layer is a 4D tensor of the shape #input $\times$ width $\times$ height $\times$ #output. In such a case, $\mathbf{W}$ is reshaped from the kernel to the shape (#input $\times$ width $\times$ height) $\times$ #output.

The goal of increasing communication efficiency of federated learning is to reduce the cost of sending $\mathbf{H}_t^i$ to the server. In this paper, we propose two general strategies of achieving this, discussed next.

## 2 Structured Update

The first type of communication efficient update restricts the updates $\mathbf{H}_t^i$ to have a pre-specified *structure*. Two types of structures are considered in the paper:

**Low rank.** We enforce $\mathbf{H}_t^i \in \mathbb{R}^{d_1 \times d_2}$ to be low-rank matrices of rank at most $k$, where $k$ is a fixed number. We express $\mathbf{H}_t^i$ as the product of two matrices: $\mathbf{H}_t^i = \mathbf{A}_t^i \mathbf{B}_t^i$, where $\mathbf{A}_t^i \in \mathbb{R}^{d_1 \times k}$, $\mathbf{B}_t^i \in \mathbb{R}^{k \times d_2}$, and $\mathbf{A}_t^i$ is generated randomly and fixed, and only $\mathbf{B}_t^i$ is optimized. Note that $\mathbf{A}_t^i$ can then be compressed in the form of a random seed and the clients only need to send $\mathbf{B}_t^i$ to the server. We also tried fixing $\mathbf{B}_t^i$ and training $\mathbf{A}_t^i$, as well as training both $\mathbf{A}_t^i$ and $\mathbf{B}_t^i$; neither performed as well. Our approach seems to perform as well as the best techniques considered in [6],

**Random mask.** We restrict the update $\mathbf{H}_t^i$ to be sparse matrices, following a pre-defined random sparsity pattern (i.e., a random mask). The pattern is generated afresh in each round and for each client. Similar to the low-rank approach, the sparse pattern can be fully specified by a random seed, and therefore it is only required to send the values of the non-zeros entries of $\mathbf{H}_t^i$. This strategy can be seen as the combination of the master training method and a randomized block coordinate minimization approach [16, 15].

[2] A weighted sum might be used to replace the average based on specific implementations.

## 3 Sketched Update

The second type of communication-efficient update, which we call *sketched*, first computes the full unconstrained $\mathbf{H}_t^i$, and then encodes the update in a (lossy) compressed form before sending to the server. The server decodes the updates before doing the aggregation. Such sketching methods have application in many domains [21]. We propose two ways of performing the sketching:

**Subsampling.** Instead of sending $\mathbf{H}_t^i$, each client only communicates matrix $\hat{\mathbf{H}}_t^i$ which is formed from a random subset of the (scaled) values of $\mathbf{H}_t^i$. The server then averages the sampled updates, producing the global update $\hat{\mathbf{H}}_t$. This can be done so that the average of the sampled updates is an unbiased estimator of the true average: $\mathbb{E}[\hat{\mathbf{H}}_t] = \mathbf{H}_t$. Similar to the random mask structured update, the mask is randomized independently for each client in each round, and the mask itself is stored as a synchronized seed. It was recently shown that, in a certain setting, the *expected iterates* of SGD converge to the optimal point [19]. Perturbing the iterates by a random matrix of zero mean, which is what our subsampling strategy would do, does not affect this type of convergence.

**Probabilistic quantization.** Another way of compressing the updates is by *quantizing* the weights. We first describe the algorithm of quantizing each scalar to one bit. Consider the update $\mathbf{H}_t^i$, let $h = (h_1, \ldots, h_{d_1 \times d_2}) = \text{vec}(\mathbf{H}_t^i)$, and let $h_{\max} = \max_j(h_j)$, $h_{\min} = \min_j(h_j)$. The compressed update of $h$, denoted by $\tilde{h}$, is generated as follows:

$$\tilde{h}_j = \begin{cases} h_{\max}, & \text{with probability } \frac{h_j - h_{\min}}{h_{\max} - h_{\min}} \\ h_{\min}, & \text{with probability } \frac{h_{\max} - h_j}{h_{\max} - h_{\min}} \end{cases}.$$

It is easy to show that $\tilde{h}$ is an unbiased estimator of $h$. This method provides $32\times$ of compression compared to a 4 byte float. One can also generalize the above to more than 1 bit for each scalar. For $b$-bit quantization, we first equally divide $[h_{\min}, h_{\max}]$ into $2^b$ intervals. Suppose $h_i$ falls in the interval bounded by $h'$ and $h''$. The quantization operates by replacing $h_{\min}$ and $h_{\max}$ of the above equation by $h'$ and $h''$, respectively. Incremental, randomized and distributed optimization algorithms can be similarly analyzed in a quantized updates setting [17, 8, 7].

**Improving the quantization by structured random rotations.** The above 1-bit and multi-bit quantization approaches work best when the scales are approximately equal across different dimensions. For example, when $\max = 100$ and $\min = -100$ and most of values are 0, the 1-bit quantization will lead to large quantization error. We note that performing a random rotation of $h$ before the quantization (multiplying $h$ by an orthogonal matrix) will resolve this issue. In the decoding phase, the server needs to perform the inverse rotation before aggregating all the updates. Note that in practice, the dimension of $h$ can be as high as $d = 1e6$, and it is computationally prohibitive to generate ($\mathcal{O}(d^3)$) and apply ($\mathcal{O}(d^2)$) a rotation matrix. In this work, we use a type of structured rotation matrix which is the product of a Walsh-Hadamard matrix and a binary diagonal matrix, motivated by the recent advance in this topic [22]. This reduces the computational complexity of generating and applying the matrix to $\mathcal{O}(d)$ and $\mathcal{O}(d \log d)$.

## 4 Experiments

We conducted the experiments using federated learning to train deep neural networks for the CIFAR-10 image classification task [11]. There are 50,000 training examples, which we partitioned into 100 clients each containing 500 training examples. The model architecture was taken from the TensorFlow tutorial [2], which consists of two convolutional layers followed by two fully connected layers and then a linear transformation layer to produce logits, for a total of over 1e6 parameters. While this model is not the state-of-the-art, it is sufficient for our needs, as our goal is to evaluate our compression methods, not achieve the best possible accuracy on this task.

We employ the Federated Averaging algorithm [14], which significantly decreases the number of rounds of communication required to train a good model. However, we expect our techniques will show a similar reduction in communication costs when applied to synchronized SGD. For Federated Averaging, on each round we select 10 clients at random, each of which performs 10 epochs of SGD with a learning rate of $\eta$ on their local dataset using minibatches of 50 images, for a total of 100 local updates. From this updated model we compute the deltas for each layer $\mathbf{H}_t^i$.

Table 1: Low rank and sampling parameters for the CIFAR experiments. The Sampling Probabilities column gives the fraction of elements uploaded for the two convolutional layers and the two fully-connected layers, respectively; these parameters are used by `StructMask`, `SketchMask`, and `SketchRotMask`. The Low Rank column gives the rank restriction $k$ for these four layers. The final softmax layer is small, so we do not compress updates to it.

| | (Low) Rank | Sampling Probabilities | model size | reduction |
|---|---|---|---|---|
| Full Model (baseline) | 64, 64, 384, 192 | 1, 1, 1, 1 | 4.075 MB | — |
| Medium subsampling | 64, 64, 12, 6 | 1, 1, 0.03125, 0.03125 | 0.533 MB | 7.6× |
| High subsampling | 8, 8, 12, 6 | 0.125, 0.125, 0.03125, 0.03125 | 0.175 MB | 23.3× |

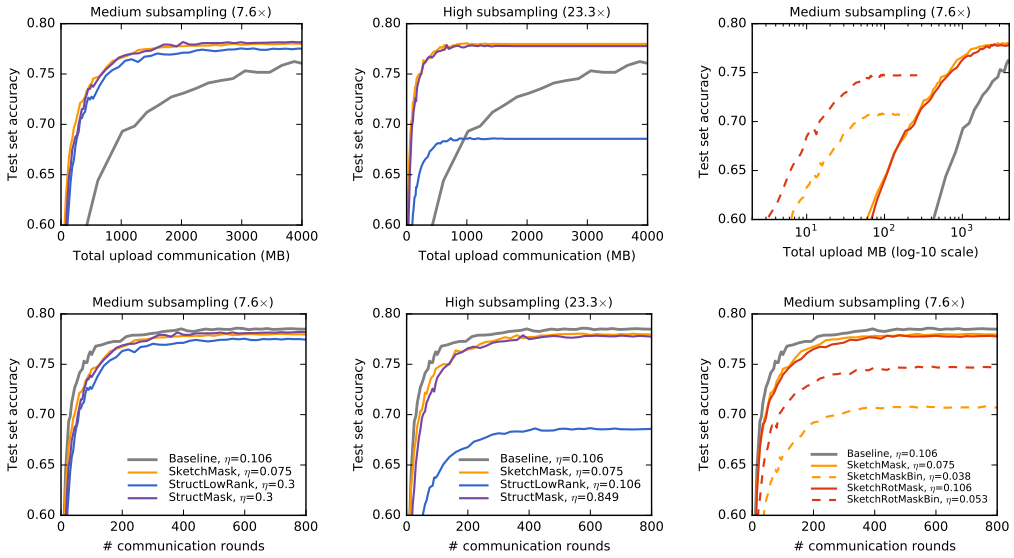


Figure 1: Non-quantized results (left and middle columns), and results including binary quantization (dashed lines `SketchRotMaskBin` and `SketchMaskBin`, right column). Note the $x$-axis is on a log scale for top-right plot. We achieve over 70% accuracy with fewer than 100MB of communication.

We define medium and high low-rank/sampling parameterizations that result in the same compression rates for both approaches, as given in Table 1. The left and center columns of Figure 1 present non-quantized results for test-set accuracy, both as a function of the number of rounds of the algorithm, and the total number of megabytes uploaded. For all experiments, learning rates were tuned using a multiplicative grid of resolution $\sqrt{2}$ centered at 0.15; we plot results for the learning rate with the best median accuracy over rounds 400 – 800. We used a multiplicative learning-rate decay of 0.988, which we selected by tuning only for the baseline algorithm.

For medium subsampling, all three approaches provide a dramatic improvement in test set accuracy after a fixed amount of bandwidth usage; the lower row of plots shows little loss in accuracy as a function of the number of update rounds. The exception is that the `StructLowRank` approach performs poorly for the high subsampling parameters. This may suggest that requiring a low-rank update structure for the convolution layers works poorly. Also, perhaps surprisingly, we see no advantage for `StructMask`, which optimizes for a random sparse set of coefficients, as compared to `SketchMask`, which chooses a sparse set of parameters to update *after* a full update is learned.

The right two plots in Figure 1 give results for `SketchMask` and `SketchRotMask`, with and without binary quantization; we consider only the medium subsampling regime which is representative. We observe that (as expected) introducing the random rotation without quantization has essentially no effect (solid red and orange lines). However, binary quantization dramatically decreases the total communication cost, and further introducing the random rotation significantly speeds convergence, and also allows us to converge to a higher level of accuracy. We are able to learn a reasonable model (70% accuracy) in only ∼100MB of communication, two orders of magnitude less than the baseline.

## References


[1] Speedtest market report. http://www.speedtest.net/reports/united-states/, August 2016.

[2] Tensorflow convolutional neural networks tutorial. http://www.tensorflow.org/tutorials/deep_cnn, 2016.

[3] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous SGD. In *ICLR Workshop Track*, 2016.

[4] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 571–582, 2014.

[5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *NIPS*, pages 1223–1231, 2012.

[6] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *NIPS*, pages 2148–2156, 2013.

[7] Mostafa El Gamal and Lifeng Lai. On randomized distributed coordinate descent with quantized updates. *arXiv:1609.05539*, 2016.

[8] Daniel Golovin, D. Sculley, H. Brendan McMahan, and Michael Young. Large-scale learning with less ram via randomization. In *ICML*, 2013.

[9] Jakub Konečný, H. Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv:1511.03575*, 2015.

[10] Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[11] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[12] Chenxin Ma, Jakub Konečný, Martin Jaggi, Virginia Smith, Michael I. Jordan, Peter Richtárik, and Martin Takáč. Distributed optimization with arbitrary local solvers. *arXiv:1412.6293*, 2014.

[13] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael Jordan, Peter Richtárik, and Martin Takáč. Adding vs. averaging in distributed primal-dual optimization. In *ICML*, pages 1973–1982, 2015.

[14] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Aguera y Arcas. Federated learning of deep networks using model averaging. *arXiv:1602.05629*, 2016.

[15] Zheng Qu and Peter Richtárik. Coordinate descent with arbitrary sampling I: algorithms and complexity. *Optimization Methods and Software*, 31(5):829–857, 2016.

[16] Zheng Qu, Peter Richtárik, and Tong Zhang. Quartz: Randomized dual coordinate ascent with arbitrary sampling. In *NIPS*, volume 28, pages 865–873, 2015.

[17] M.G. Rabbat and R.D. Nowak. Quantized incremental algorithms for distributed optimization. *IEEE Journal on Selected Areas in Communications*, 23(4):798–808, 2005.

[18] Sashank J Reddi, Jakub Konečný, Peter Richtárik, Barnabás Póczós, and Alex Smola. AIDE: Fast and communication efficient distributed optimization. *arXiv:1608.06879*, 2016.

[19] Peter Richtárik and Martin Takáč. Stochastic reformulation of linear systems and fast stochastic iterative methods. Technical report, 2016.

[20] Ohad Shamir, Nathan Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate Newton-type method. In *ICML*, pages 1000–1008, 2014.

[21] David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

[22] Felix X Yu, Ananda Theertha Suresh, Krzysztof Choromanski, Daniel Holtmann-Rice, and Sanjiv Kumar. Orthogonal random features. In *NIPS*, 2016.

[23] Yuchen Zhang and Xiao Lin. DiSCO: Distributed optimization for self-concordant empirical loss. In *ICML*, pages 362–370, 2015.