Federated Learning

Strategies for Improving Communication Efficiency

Motivation

- Datasets are growing larger, and model complexity is continuing to increase
  - 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.
- Large number of low-powered unstable connected devices (phones)
- Privacy (have people store their own data)
- Training data coming from users' phones from interaction (typically non-IID, think GBoard)
- How can we collaboratively learn a shared model while keeping data on people's devices; decoupling the ability to do machine learning from the need to store the data in the cloud?

Federated Averaging Algorithm

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

2. Each client in the subset computes an updated model based on their local data.

3. The model updates are sent from the selected clients to the sever.

4. The server aggregates these models (typically by averaging) to construct an improved global model.

Federated Averaging Algorithm (paper)

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
  initialize w_0
  for each round $t = 1, 2, \dots$ **do**
    $m \leftarrow \max(C \cdot K, 1)$
    $S_t \leftarrow$ (random set of m clients)
    for each client $k \in S_t$ **in parallel do**
      $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
  for each local epoch i from 1 to E **do**
    for batch $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \eta \nabla \ell(w; b)$
  return w to server

K; Number of clients.

C; Fraction of clients to include in this round.

B; Minibatch size; B = ∞ full dataset is treated as a single batch.

E; Number of training passes client makes over local dataset.

On Averaging Weights

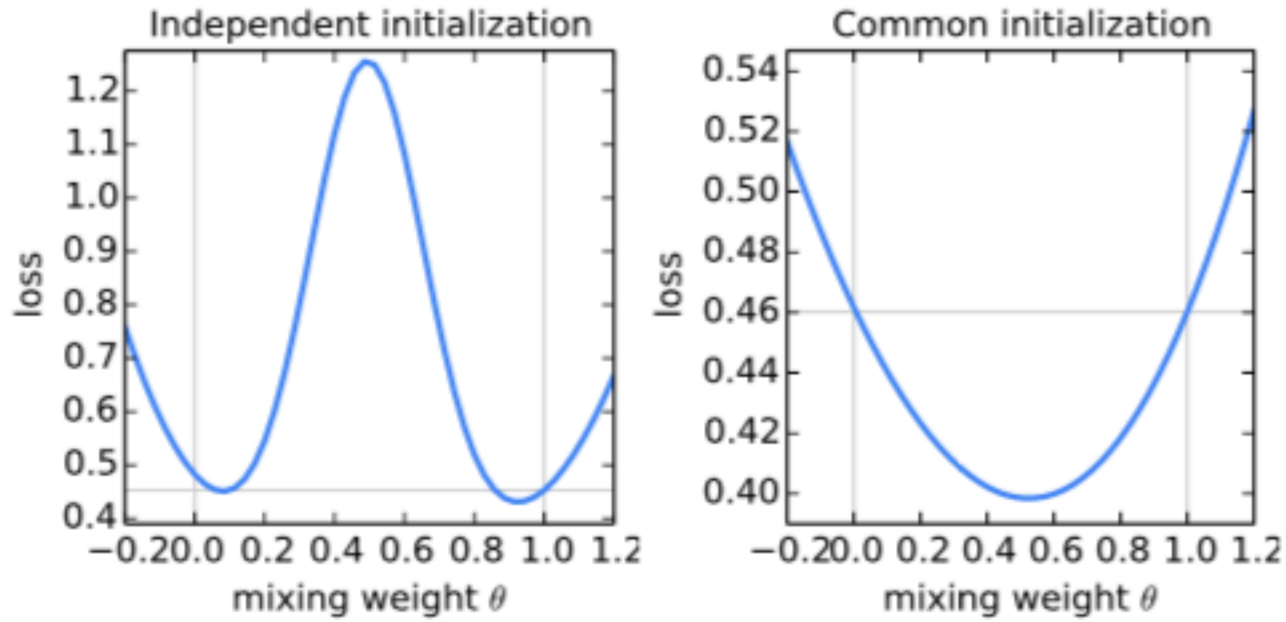

Figure 1: The loss on the full MNIST training set for models generated by averaging the parameters of two models w and w' using $\theta w + (1 - \theta)w'$ for 50 evenly spaced values $\theta \in [-0.2, 1.2]$.The models w and w' were trained using SGD on different small datasets. For the left plot, w and w' were initialized using different random seeds; for the right plot, a shared seed was used. Note the different y-axis scales. The horizontal line gives the best loss achieved by w or w' (which were quite close, corresponding to the vertical lines at $\theta = 0$ and $\theta = 1$). With shared initialization, averaging the models produces a significant reduction in the loss on the total training set (much better than the loss of either parent model).

Challenges

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

2. Each client in the subset computes an updated model based on their local data.

3. The model updates are sent from the selected clients to the sever.

   - Bottleneck: unrealistic because of asymmetric internet bandwidth speeds (e.x. 125Mbps down vs 25Mbps up).

   - Added cryptographic protocols to conceal individual data increase bits needed to be upload.

4. The server aggregates these models (typically by averaging) to construct an improved global model.

Communication Optimizations

Approach 1: Structured Updates

- Learn an update from a restricted space that can be expressed using a smaller number of variables (e.x. low rank, or random mask)

Approach 2: Sketched Updates

- Learn a full model update, then compress it before sending it to the server

Note: Approach 1 we learn on the restricted space vs Approach 2 we do compression after a regular full model update.

Structured Update

Approach A: Low Rank

- Enforces every model update (**H**) to be a low rank matrix of rank *k*
- Express **H** = **A** x **B**
  - **A** (size: d_1 x k; random matrix, stays constant during training; compressed to random seed), **B** (size: k x d_2; learned)
  - "Given a given random reconstruction (**A**), what is the projection (**B**) that will recover most information?"

Approach B: Random Mask

- Enforces every model update (**H**) to be a sparse matrix with a random mask applied
- Mask is generated for each round, for each client; compressed to random seed
- Only need to upload random seed and non-empty values of **H**

Sketched Update

Computes **H** during local training without any constraints, then approximates update in a (lossy) compressed form before uploading. Server decodes before aggregation.

Approach A: Subsampling

- Upload random (scaled) subset of **H;** randomized for each round, for each client; compressed to random seed

Approach B: Probabilistic Quantization

- For b-bit quantization, divide $[h_{min}, h_{max}]$ into 2^b intervals
- $\tilde{h}_j = \begin{cases} h_{\max}, & \text{with probability } \frac{h_j - h_{\min}}{h_{\max} - h_{\min}} \\ h_{\min}, & \text{with probability } \frac{h_{\max} - h_j}{h_{\max} - h_{\min}} \end{cases}$
- To help when scales are not approx. equal across dimensions, apply a random rotation matrix on *h* before quantization.

Experiments

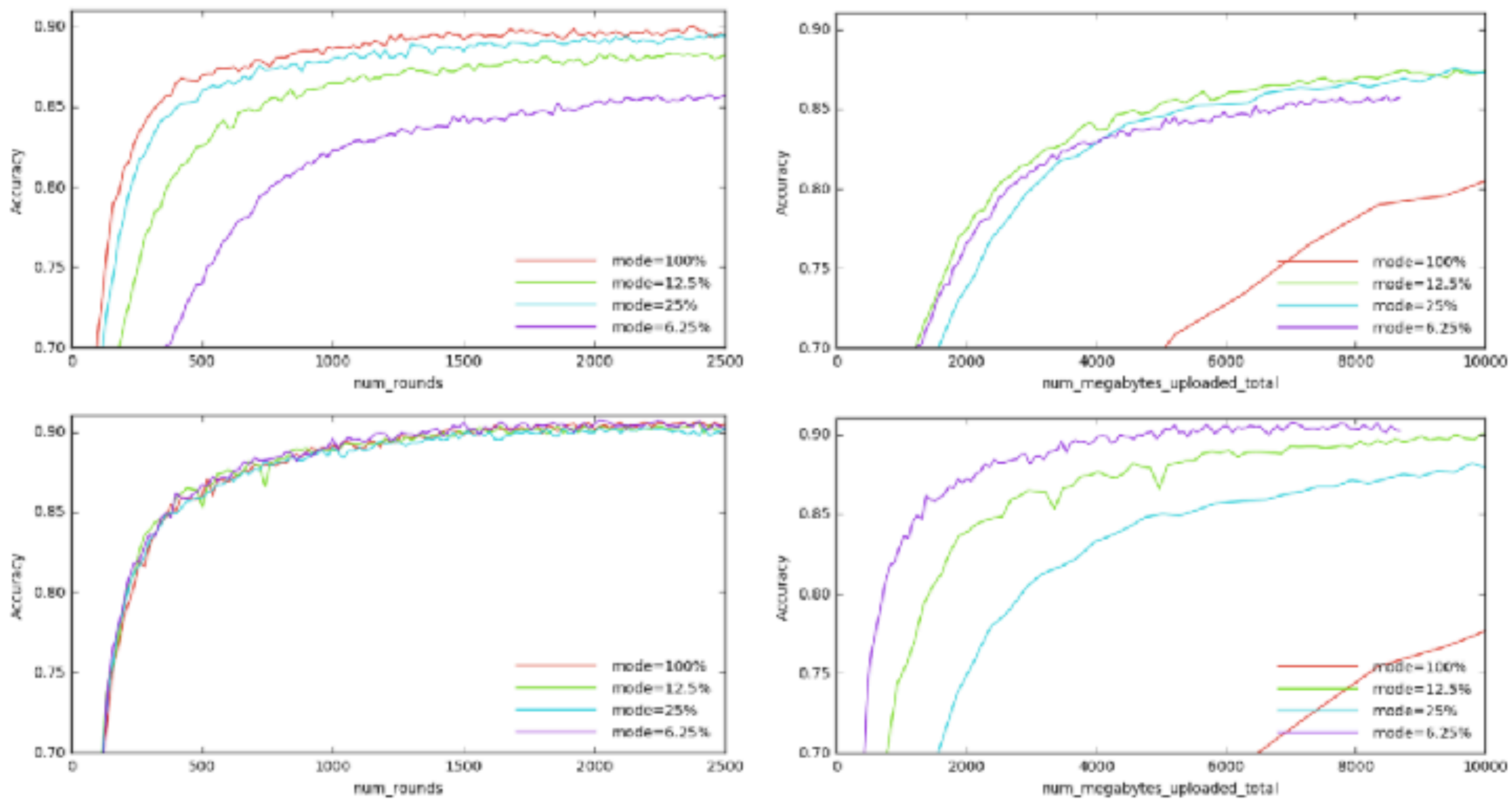

Figure 1: Structured updates with the CIFAR data for size reduction various modes. *Low rank* updates in top row, *random mask* updates in bottom row.

Experiments

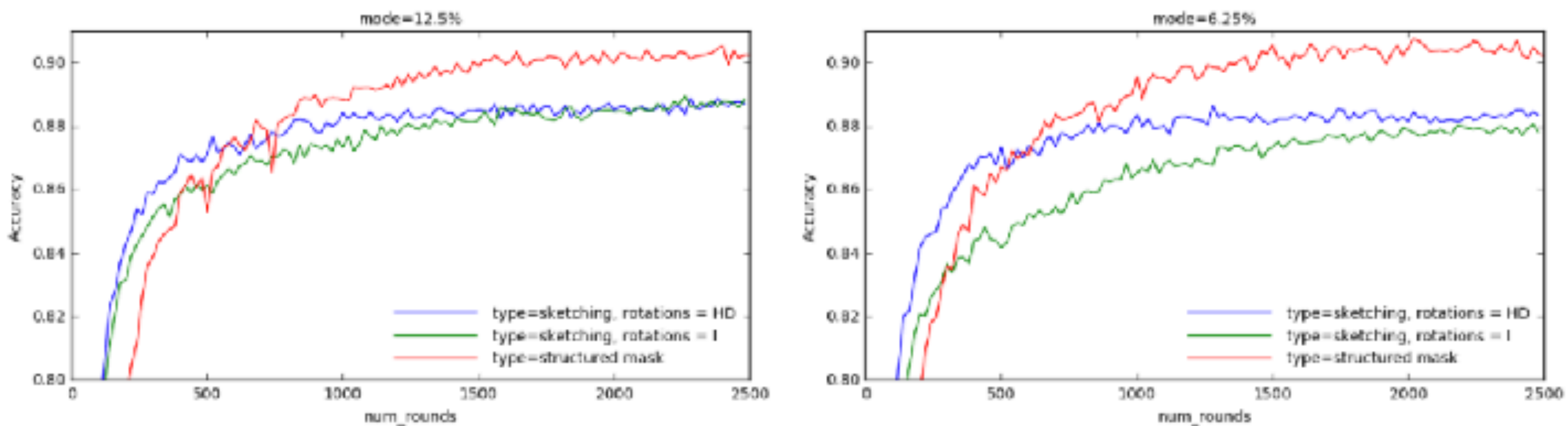

Figure 2: Comparison of structured *random mask* updates and sketched updates without quantization on the CIFAR data.

Experiments

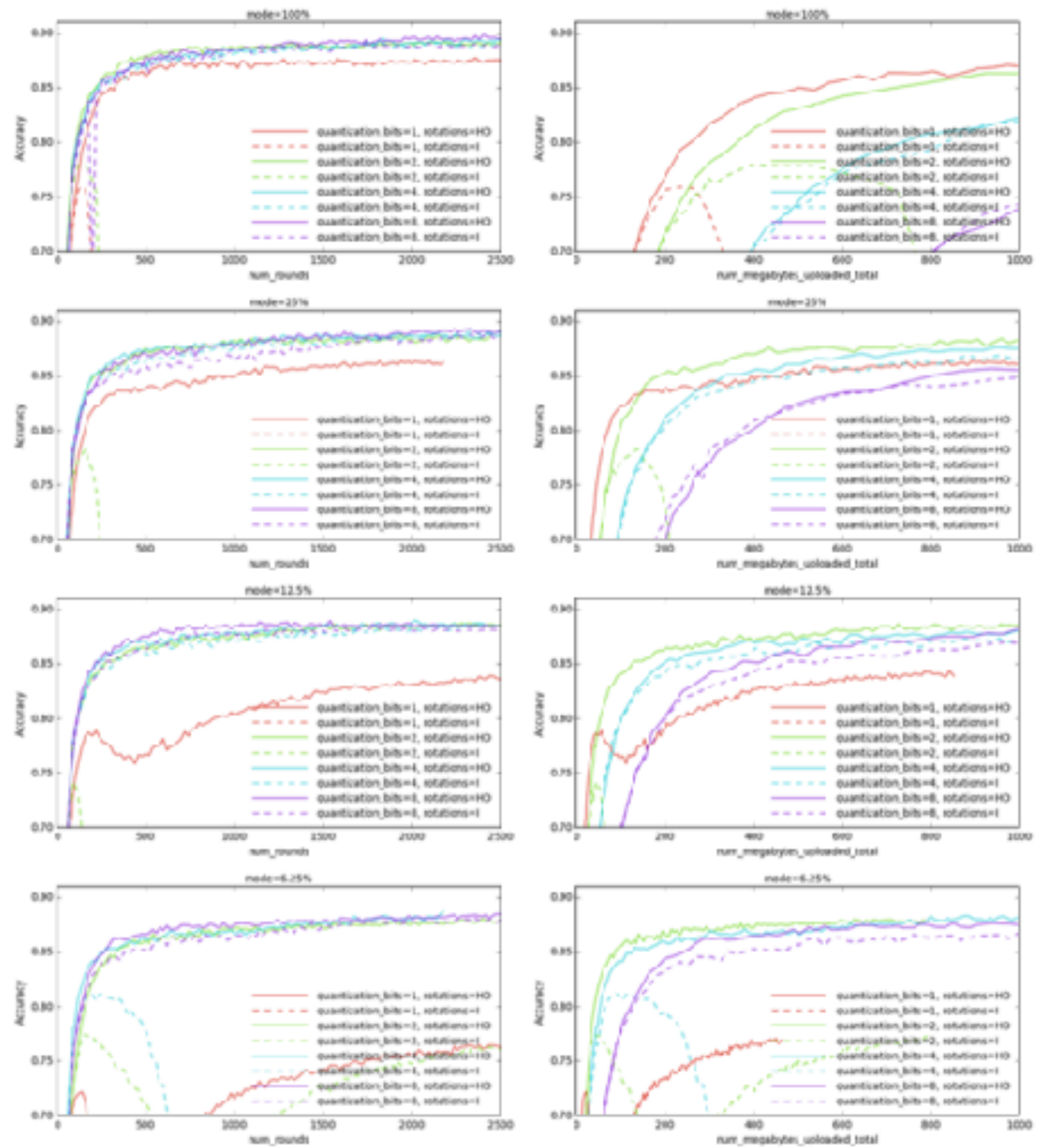

Figure 3: Comparison of sketched updates, combining preprocessing the updates with rotations, quantization and subsampling on the CIFAR data.

Experiments

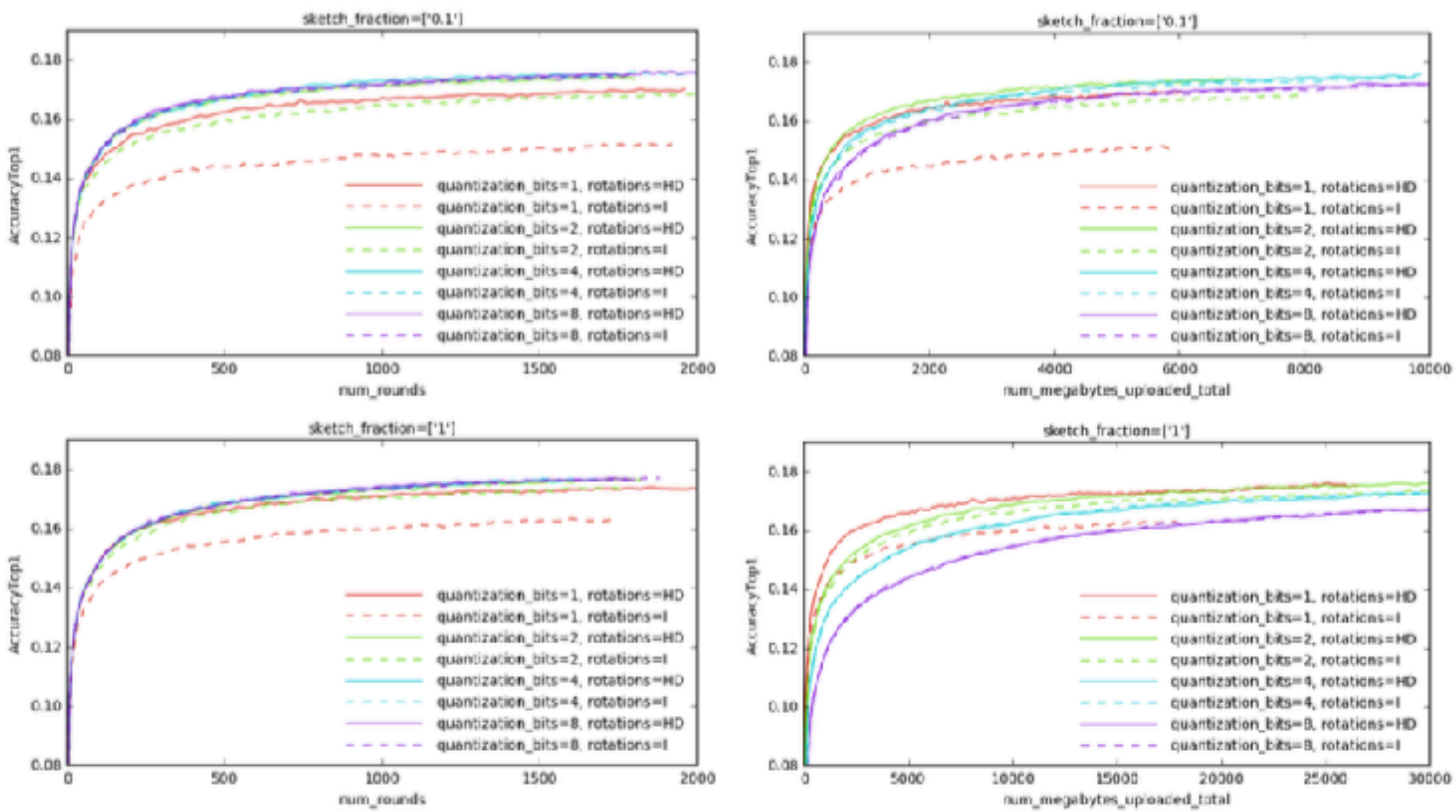

Figure 4: Comparison of sketched updates, training a recurrent model on the Reddit data, randomly sampling 50 clients per round.