2023 IEEE 20th International Conference on Mobile Ad Hoc and Smart Systems (MASS) | 979-8-3503-2433-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/MASS58611.2023.00033

Dynamic Model Reduction for Decentralized Learning in Heterogeneous Mobile Networks

Sangsu Lee[†], Xi Zheng[‡], Jie Hua[†], Haoxiang Yu[†], Christine Julien[†]
[†]Department of Electrical and Computer Engineering, University of Texas at Austin
{sethlee, mich94hj, hxyu, c.julien}@utexas.edu
[‡]School of Computing, Macquarie University, james.zheng@mq.edu.au

***Abstract*—Collaborating with nearby devices to train and personalize deep learning models opens the potential to support new mobile application scenarios. In emerging decentralized learning algorithms, devices communicate over a peer-to-peer network to share knowledge obtained from local data. However, communication bandwidth, computing power, and the duration for which these connections are available are limited and heterogeneous. In this paper, we explore the feasibility and efficacy of adaptive model reduction strategies for decentralized learning algorithms (Dynamic Reduction (DR)). For this study, we use as an exemplar an existing opportunistic learning algorithm (OppCL) that relies on device-to-device model exchanges to iteratively train a local model based on encounters. In layering model reduction on OppCL, when a device encounters a potential learning partner, it dynamically constructs a model reduction suitable for given computation and communication budget by quantizing weights and building a dropout version of a neural network. We term our new approach DR-OppCL and show that DR-OppCL leads to faster convergence with minimal effort in tuning hyperparameters related to model reduction, using both simulated and real-world mobility traces. While we demonstrate our DR approaches in the context of OppCL, they are generic and can be easily applied to other decentralized learning algorithms.**

***Index Terms*—mobile computing, collaborative deep learning, distributed machine learning, decentralized learning**

I. Introduction

Mobile applications can benefit from models learned from distributed data, for instance, to perform object classification in images taken with a smartphone, perform activity recognition on wearable devices, or predict a user's interactions with social media. However, the data these models use can be very private, and it is often not desirable to share it. Federated learning (FL) [14] allows devices to maintain privacy by training updates to a global model on end nodes and aggregating them at a central server. However, relying on a coordinator maintains concerns of bottlenecks and a single point of failure.

In (fully) decentralized learning [16, 20], devices collaborate without relying on a centralized entity; decentralized FL is a subset of decentralized learning [23, 29]. *Opportunistic collaborative learning* (OppCL) [18] enables devices to rely on device-to-device communication to collaboratively train local models in an effort to personalize their models by incorporating knowledge from neighboring devices. In OppCL, each device trains a personalized local model by asking encountered devices to compute a model update using local data. OppCL is suitable for scenarios where devices (i) encounter new neighbors regularly; (ii) have limited connection to a central server; (iii) have diverse (personalized) learning goals; and (iv) hold training data that is private or large.

Support for device-to-device communication has proliferated [4, 7]. However, decentralized learning approaches are limited because they assume encountered devices are homogeneous in communication, computation, and memory. In practice, devices in these environments are heterogeneous, ranging from sophisticated edge or fog nodes, through smartphones down to resource-constrained embedded IoT devices.

In decentralized learning, model complexity is often tuned to a device's specific resource constraints. However, when devices are diverse and hosting models of different complexity, opportunities to collaborate are limited to encounters with devices that have similar capabilities and encounters that are long enough to complete the exchange of model updates. We aim to allow devices to benefit from opportunistic collaboration *even when encountered devices have diverse resource constraints or the encounters are short-lived*. We have designed an adaptive strategy (Dynamic Reduction, or DR) that dynamically determines the appropriate level of model reduction, utilizing two techniques from a classic ML toolbox: (1) *dropout*, which we use to reduce the models sent to neighbors with lower computational capabilities and (2) *quantization*, which we use to limit the amount of communication in an encounter.

Fig. 1 shows our generic DR strategy for decentralized learning. When one device (termed the learner) encounters another (termed the neighbor), the discovery module fetches information about the neighbor's resource constraints, including computational resources, such as the maximum size model it can run, and communication resources, such as the data rate at which it can send and receive. The dynamic reduction module uses this information to determine a strategy appropriate for the learning algorithm. The model is reduced, enabling collaboration even when data rates are limited, or allowing collaborative training on a resource-constrained device.

We use OppCL [18] to demonstrate and evaluate our DR strategy. Because its design relies on encountering a neighbor, sharing a model, and waiting for the neighbor to complete a round of training and return an update, the encounters that OppCL can use are limited to those of a longer duration. However, when DR is layered on top of OppCL, its ability to *quantize* model representations allows it to take advantage of shorter encounters. Further, OppCL can benefit from the

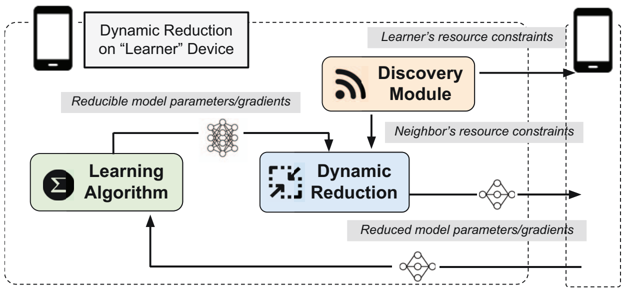

Dynamic Reduction on "Learner" Device
Learner's resource constraints
Discovery Module
Reducible model parameters/gradients
Neighbor's resource constraints
Learning Algorithm
Dynamic Reduction
Reduced model parameters/gradients

Fig. 1: Overview of Dynamic Reduction (DR). The learner dynamically adapts to the neighbor's resource constraints.

DR strategy's use of *dropout* to create a model suitable to the neighbor's resources. The creation of DR-OppCL is just one example use of the generic DR strategy. The strategy can also be combined with other decentralized learning protocols.

This paper focuses on the practical application of machine learning techniques to the resource constraints and heterogeneity that real-world mobile computing applications must overcome. Concretely, we make the following contributions:

- We propose Dynamic Reduction (DR), an adaptive model reduction strategy for decentralized learning in heterogeneously resource-constrained networks.
- We create an adaptive version of *dropout* to directly address the computational heterogeneity of the encountered collaborators in real-world mobile environments.
- We adaptively determine a quantization rate associated with communicating model parameters and gradient updates among opportunistically collaborating devices.
- We demonstrate the application of DR to an exemplar decentralized learning algorithm, OppCL.

DR preserves training accuracy on heterogeneous mobile devices in real-world mobile networks while deftly navigating tradeoffs in computation and communication overhead.

II. Related Work

We overview the state of the art in learning disciplines applied to the context of mobile computing and use them to frame our approach. We then describe efforts to adapt learning disciplines to heterogeneous environments, in particular those with communication and computation constraints.

Learning Disciplines. Interest in machine learning for dynamic mobile environments has ballooned. With respect to training on mobile devices, there are several relevant approaches, including *federated learning*, *decentralized learning*, and *opportunistic learning*. In *federated learning* (FL), the primary goal is to maintain privacy of individual users' training data; rather than shipping raw data to a centralized location and performing training on the aggregated raw data, each device performs updates to a shared model *locally*, then returns the resulting gradients, which are aggregated centrally [14].

In *decentralized learning* [16], individual devices exchange model updates in an entirely peer-to-peer fashion. While devices may learn somewhat different versions of the target model, the overall objective is still a global one, e.g., to minimize the average loss of the model across all clients.

Recognizing that different devices may have different, personalized goal models, OppCL [18] uses ephemeral encounters to support devices in training their own local models using the (private) data of encountered neighbors. Rather than exchanging gradients learned while seeking convergence to a global model, devices share their model parameters with neighbors who then use their own local data to compute an update specific to the neighbor's goal. In this work, we tackle the problem that arises when learning in a real-world environment with heterogeneously resource-constrained devices. Though we assume that neighboring devices will participate, there are works on incentivizing opportunistic collaboration [11].

Learning in Heterogeneous Environments. Existing techniques have explored supporting devices with heterogeneous capabilities in federated or decentralized learning by considering the potential for leveraging edge or fog nodes [28]. These approaches leverage edge nodes for offloading computation as part of training or to support incremental computation of gradients. Prior work generally falls in two directions: (1) reducing the size or complexity of the model to support devices with diverse computational capabilities and reduce the communication requirements of collaborative learning and (2) reducing the number of bits used to represent the model to reduce communication requirements.

There are multiple ways to reduce the complexity of a neural network (e.g., *knowledge distillation* [19], *model pruning* [6], and Dropout [26]). Dropout randomly selects neurons to omit from a training iteration to prevent the model from becoming overly reliant on a specific small subset of hidden units. In the FL space, dropout reduces both the computation and communication requirements for resource-constrained devices [1]. Given that these approaches apply the reduction techniques to a centrally coordinated and orchestrated FL task, their success supports our proposed novel application of dropout to the entirely decentralized opportunistic learning environment.

While model sparsification addresses both computation and communication constraints, other approaches focus specifically on reducing communication overhead of sharing gradients and models. *Quantization* [8] reduces the number of bits used to represent the weights in the model, lowering the communication overhead. Quantization has been explored in federated learning [3] and in decentralized learning [27].

III. DR-OppCL: Dynamic Reduction for Opportunistic Collaborative Learning

Decentralized learning algorithms largely neglect the heterogeneous capabilities of encountered devices, limiting their real-world applicability [9, 29]. Incorporating our DR strategy into decentralized learning can significantly improve effectiveness by enabling smaller devices to participate in training models that are otherwise too big to train and within shorter-duration encounters. Of particular importance is that DR is *adaptive*—it determines its operating mode on an encounter-by-encounter basis, depending on the instantaneous capabilities of the collaborators and their communication link.

Algorithm 1: OppCL [18]

```
1  $\mathcal{G}_i$: goal label distribution for $\mathcal{C}_i$'s learning task
2  $\mathcal{L}_j$: data distribution advertised by the neighbor $\mathcal{C}_j$
3  $\mathcal{B}_j$: batch of $\mathcal{C}_j$'s data used for training
4  Function onDiscover: ($\mathcal{L}_j$)
5      if decideOppCL($\mathcal{G}_i, \mathcal{L}_j$) then
6          ask $j$ to compute $g \leftarrow \nabla\ell(w_i^t; \mathcal{B}_j)$
7          receive $g$
8          $w_i^{t+1} \leftarrow$ updateModel($g$)
9      end
10 end
```

A. OppCL and its Limitations

Because we apply DR to OppCL, we next overview the behavior of OppCL [18]. When a device ("the learner") encounters another device ("the neighbor"), it may request the neighbor to perform a round of training on the learner's model using the neighbor's data. The learner sends its model parameters to the neighbor; the neighbor trains the model with its local data and returns the gradients, which the learner incorporates into its personal model. Each data item has a label (e.g., a photo may be labeled with whether it was shared on social media; a sequence of words may be labeled with the emoji that follows them). OppCL captures device $\mathcal{C}_i$'s *data label distribution* ($\mathcal{L}_i$) as the relative frequency of each label within the local data set. Each device aims to learn a local model for some personalized task represented by a *goal distribution* ($\mathcal{G}_i$) of labels that it desires for its model to be successful at classifying. It is common for a device's goal and data label distributions to differ.

Algorithm 1 shows OppCL as presented in [18]. The decideOppCL function abstracts the learner's decision logic that determines whether to request training from a discovered neighbor (line 5). In OppCL, this function simply checks whether the learner's goal distribution $\mathcal{G}_i$ is sufficiently similar to the neighbor's local data distribution $\mathcal{L}_j$. If so, the learner requests that the neighbor use the learner's model weights to compute new gradients on a batch of the neighbor's local data, and return the gradients to the learner. Using the received gradients, the learner updates the local model (line 8).

Two challenges arise in directly applying OppCL. First, due to limitations in computational power, a neighbor may be unable to compute a model update. Second, the duration or bandwidth of an encounter may not always be sufficient to support sending w_i^t or receiving g (lines 6 and 7). Applying DR to OppCL tackles both problems.

B. Overview of DR-OppCL

Fig. 2 depicts an opportunistic encounter in DR-OppCL. As part of a continuous discovery process [10] the learner discovers the availability of the neighbor, which also advertises a summary of its locally available data. The learner uses this information to determine whether the encountered neighbor has had experiences from which the learner seeks to learn. If so, the learner uses the neighbor's advertised capabilities to determine an appropriate dropout rate, uses the expected duration and quality of the communication link [2] to compute

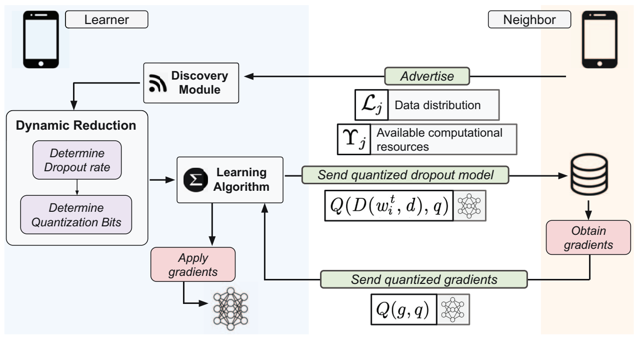


Fig. 2: An Opportunistic Encounter in DR-OppCL.

Algorithm 2: DR-OppCL

```
1  $\Upsilon_j$: available computational resources advertised by $\mathcal{C}_j$
2  getDropoutRate($\Upsilon_j$): determines dropout rate based on
     the neighbor's computational resources.
3  getQuantizationBits($d, \mathcal{C}_j$): determine quantization bits
     based on dropout rate $d$ and predicted encounter duration.
4  Function onDiscover: ($\mathcal{L}_j$, $\Upsilon_j$)
5      if decideOppCL($\mathcal{G}_i, \mathcal{L}_j, \Upsilon_j$) then
6          $d \leftarrow$ getDropoutRate($\Upsilon_j$)
7          $q \leftarrow$ getQuantizationBits($d, \mathcal{C}_j$)
8          if $d \neq$ nil $\wedge$ $q \neq$ nil then
9              ask $j$ to compute $g \leftarrow \nabla\ell(Q(D(w_i^t, d), q); \mathcal{B}_j))$
10             receive $G_j^t \leftarrow Q(g, q)$
11             $w_i^{t+1} \leftarrow$ updateModel($G_j^t$)
12         end
13     end
14 end
```

the appropriate number of quantization bits, then sends the model parameters to the neighbor. The neighbor trains the model using the neighbor's own local data. When training is complete, the neighbor returns a quantized version of the updated gradients, which the learner applies to its model.

Algorithm 2 details a learner $\mathcal{C}_i$'s encounter with neighbor $\mathcal{C}_j$. The neighbor ($\mathcal{C}_j$) continuously advertises its data label distribution ($\mathcal{L}_j$) and the computational resources it has available for learning on behalf of neighbors (Υ_j). When $\mathcal{C}_i$ discovers $\mathcal{C}_j$, DR-OppCL's decideOppCL first determines whether $\mathcal{C}_j$'s label distribution is sufficiently similar to $\mathcal{C}_i$'s learning goal (line 4; identical to OppCL). If so, the learner computes the best possible dropout rate, given $\mathcal{C}_j$'s available resources (line 8; Section III-C) and the optimal number of quantization bits achievable, given the network context (line 9; Section III-D). In both cases, we try to make as much use as possible of the available resources: we choose the smallest amount of dropout that $\mathcal{C}_j$ is capable of computing and the optimal number of bits that will succeed in being communicated. If DR-OppCL finds feasible levels of dropout and quantization for the encounter, then $\mathcal{C}_i$ requests that $\mathcal{C}_j$ compute a model update using $\mathcal{C}_j$'s local data. In Algorithm 2, line 9, D references a function that computes a dropout model from the learner's local model using the dropout rate d, while Q computes a quantized version of this model's parameters using only q bits. Once the neighbor has completed training, it returns the quantized updated gradients to the learner (line 12).

Finally, the learner incorporates the received gradients into the local model (w_i), using an update procedure adapted to accommodate dropout versions of the model (Section III-E).

C. Handling Computational Heterogeneity

In selecting the degree of dropout for a given neighbor, DR-OppCL is greedy—it seeks to use the largest model the neighbor can accommodate, computationally. When characterizing dropout, we use the rate to refer to the percentage of units removed from the model, i.e., a dropout rate of 0.1 indicates that 10% of the units are omitted from the shared model. We apply the dropout to each layer, similar to Federated Dropout [1]. Therefore, model architectures resulting from the same dropout rate are the same. DR-OppCL adapts the rate to each encounter, based on that encounter's available resources.

Dropout models can still converge even when many units are dropped because continuously updating models compensates for noise that arises in updating them sparsely [26]. However, a DR-OppCL model mixes diverse dropout rates, which increases the risk for the gradient updates to result in noise and hinder convergence. To combat this, DR-OppCL dynamically computes a maximum allowable dropout rate, d^e_{max} for each encounter, as the first step of GETDROPOUTRATE (line 8 in Algorithm 2). The learner then computes the minimum supportable dropout rate, based on the neighbor's computational resources. If $d_{\mathcal{C}_j} > d^e_{max}$, GETDROPOUTRATE returns NIL, indicating that no usable dropout rate exists for this encounter.

To compute d^e_{max}, we estimate the distribution of $d_{\mathcal{C}_j}$ of the devices in the learner's surroundings. We keep a recent history of the computed minimum dropout rates that encountered devices can support. We rely on a window W_e of size w that contains the dropout rates computed for the w encounters immediately preceding an encounter e. Based on W_e, we compute the max allowable dropout rate for encounter e as:

$$d^e_{max} = \max\left\{d \in W_e : \mathbb{Q}\left(\sigma_d^{-1} \cdot \left(\frac{|\theta_i|}{|D(\theta_i, d)|} - \mu_d\right)\right) \geq \tau\right\} \quad (1)$$

Intuitively, the learner uses d^e_{max} to skip encounters that require a dropout rate higher than a statistic computed on the window of dropout rates for recent encounters. When the learner is in the presence of more resource-poor neighbors, the value of d^e_{max} will decrease, instructing the learner to take advantage of opportunities requiring higher dropout rates.

$\mathbb{Q}$ measures the probability that a normal random variable will exhibit a value larger than a given number of standard deviations; μ_d and σ_d are the mean and standard deviation of the w samples for d in W_e; $|\theta_i|$ is the number of parameters in the learner's full model; and $|D(\theta_i, d)|$ is the number of parameters remaining given dropout rate d. This function seeks a dropout rate d^e_{max} for which it can be expected that a device capable of d^e_{max} will be encountered at least $|\theta_i|/|D(\theta_i, d)|$ times in the next w encounters with a confidence of τ. τ is in the range $[0.1, 0.9]$; it should be larger when an application seeks to save resources and smaller when the application seeks to promote faster training. In this paper, we used $\tau = 0.5$.

Selecting d^e_{max} with Gaussian approximation works well even with few samples (e.g., $w = 30$). Further, it is desirable to choose a $w < |\theta_i|/|D(\theta_i, d_{max})|$, where d_{max} is a global upper bound of the dropout rate; d_{max} is specific to the learning task and represents the point at which training diverges.

D. Handling Communication Heterogeneity

To handle heterogeneity in communication, we compute q^e, the number of quantization bits to use for the model and gradient exchanges for encounter e. This computation relies on a global constant q_{min} that is an absolute minimum number of quantization bits to be used for this model. As with d_{max}, this is dependent on the learning task, and represents the point at which further quantization is detrimental to training.

To compute q^e, we rely on the predicted encounter duration (t_{enc}) [2]; the predicted time for training (t_{train}), based on the neighbor's computational resources (Υ_j); and the selected dropout rate (d^e). We use the same number of bits to quantize the model parameters the learner sends to the neighbor and the gradients the neighbor returns because we assume the communication link is symmetric. Specifically, assuming $t_{train} < t_{enc}$, we compute q^e as:

$$q^e = \max\left\{\left\lfloor \log_2\left(\frac{(t_{enc} - t_{train}) \cdot R_b}{2 \cdot |D(\theta_i, d^e)|}\right)\right\rfloor, q_{min}\right\} \quad (2)$$

To quantize model parameters and gradients, we map the values of each weight onto a uniformly discretized grid of range $[\theta^k_{min}, \theta^k_{max}]$. Then we project each parameter or gradient value to the closest point in the grid. For the model parameters and gradients to be reconstructed, we also transmit θ^k_{min} and θ^k_{max}, each represented as a 64 bit float, which yields $2 \cdot 64 + |\theta| \cdot q^e$ bits that must be transmitted in each direction.

E. Incorporating Neighbor Gradients

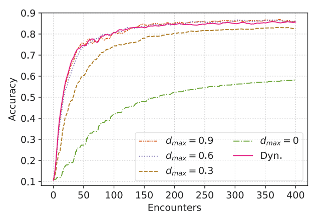
To incorporate gradients trained with different dropout rates, we introduce a modified version of the Adam optimizer [12], AdamMD (Adam for Mixed Dropout). Briefly, Adam maintains a learning rate for each model parameter, which it adapts based on statistics computed over the history of the gradient. These adaptations rely on both the first moment (i.e., the gradient's mean) and the second moment (i.e., the gradient's uncentered variance). Adam has been shown to be a good choice as an optimizer in settings similar to ours [25].

To adapt Adam to mixed dropout, we make a slight adjustment to the update function for the two moment estimates:

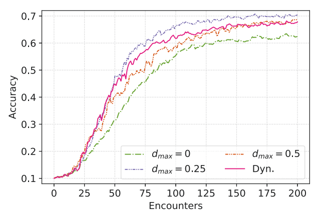
$$m^k_t \leftarrow \begin{cases} \beta_1 \cdot m^k_{t-1} + (1 - \beta_1) \cdot g^k & \text{if } \theta^k \text{ in } D(\theta, d) \\ m^k_{t-1} & \text{otherwise} \end{cases} \quad (3)$$

$$v^k_t \leftarrow \begin{cases} \beta_2 \cdot v^k_{t-1} + (1 - \beta_2) \cdot (g^k_t)^2 & \text{if } \theta^k \text{ in } D(\theta, d) \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

θ^k is the k-th value of the model parameters, and g^k is the corresponding value in the gradient. If θ^k is present in the dropout model, we update the moment according to the standard Adam update. If θ^k is *not* in the dropout model, we maintain the previous value of the first moment and set the second moment to 0. Therefore, AdamMD amplifies the next gradient that is applied to these units, thereby rectifying error



(a) MNIST, uniform



(b) SVHN, uniform

Fig. 3: Comparing different dropout rates.

induced from the reused first moment. We inherit β_1 and β_2, the moment estimates' exponential decay rates, from the Adam optimizer and use a fixed stepsize of 0.001.

IV. Evaluation

We evaluate DR-OppCL using (1) controlled experiments of the impacts of diverse resource heterogeneity and (2) simulation studies that explore performance in real-world mobility.

We rely on three learning tasks; MNIST [17], SVHN [22], and CIFAR-10 [15]. For MNIST, we use a model similar to the "2NN" model in [21]. We report results with four dropout rates: 0 (no dropout), 0.3, 0.6, and 0.9. SVHN and CIFAR-10 use a CNN model. To apply different dropout rates, we apply different filter sizes to the layers depending on their proximity to the input layer; filters near the input are dropped less than others. We use cross-categorical entropy loss with L2 regularization, which prevents gradients that exhibit large differences and cause quantization to lose more information.

A. Controlled Experiments

We evaluate how the accuracy of the trained model changes when it encounters neighbors with different computational capabilities and encounter durations. In these experiments, the learner encounters 30 unique devices repeatedly. For MNIST, each device is allocated 50 images as its local data and uses all 50 as a batch when asked to train on behalf of a neighbor. In SVHN, each device is allocated 100 images and uses all 100 as a batch for each round of training. All experiments are averaged over 10 runs with different random seeds.

Computational heterogeneity. Because DR adapts to the real-time capabilities of devices, we report experiments with different distributions of device capabilities. Each device is assigned a computational capability that ranges from $\{0, 0.1, 0.2, ..., 0.9\}$. For convenience, these numbers represent the dropout rates the device can support for training(e.g., a device with computational capability 0.4 can compute gradients from models with $d \geq 0.4$). We report experiments with two distributions of device capabilities: *uniform* (i.e., all device capabilities are equally likely to occur) and *inverse uniform* (i.e., devices with higher capabilities are less likely to occur).

Fig. 3 shows the results for the uniform distribution considering only dropout and no quantization. Each figure compares statically assigned dropout rates to DR, computed according to Equation 1. In the static condition, only neighbors with computational capabilities that meet d_{max} are used to collaboratively train the model. For instance, when $d_{max} = 0.3$, only neighbors with capability levels 0.3 or smaller are used.

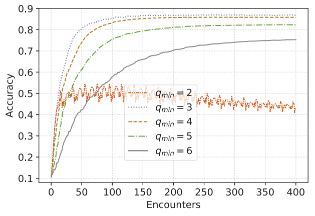
TABLE I: Comparison of d_{max}

		uniform		inv. uniform	
	d_{max}	MFLOPs	Accuracy(%)	MFLOPs	Accuracy(%)
MNIST	0.9	34.05	86.21	20.29	82.68
	0.6	31.16	86.15	12.79	80.07
	0.3	18.45	82.46	6.57	70.02
	0	4.45	58.14	2.22	38.41
	Dyn.	27.01	85.71	15.63	82.49
SVHN	0.5	287.78	68.64	211.30	50.76
	0.25	242.50	70.39	141.22	62.64
	0	170.01	62.40	67.51	36.37
	dyn.	249.07	67.81	151.76	58.89

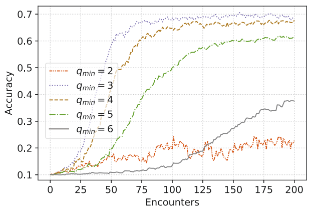
There is not a significant difference in Fig. 3(a) between d_{max} of 0.3, 0.6, and 0.9. However $d_{max} = 0$ performs quite poorly because the learner can only collaborate with similarly capable devices. For the inverse uniform distribution (not shown), the performance of $d_{max} \leq 0.3$ falters for the same reason. Table I compares the accuracy of each model to the total number of computational steps required to train it. For MNIST, while $d_{max} = 0.9$ achieves the highest accuracy, the dynamic approach achieves nearly the same accuracy with significantly less computation. These same general trends hold for SVHN. Notably for SVHN, the dynamic approach learns more quickly at the outset, indicating its flexibility to adapt. Because the drop out rate is highly dependent on the encountered devices (thus can not be set statically), the dynamic approach is more feasible with nearly optimal results.

Communication Heterogeneity. To assess DR-OppCL with different quantizations, we use uniform and inverse uniform distributions of encounter durations rather than dropout model size. Fig. 4 shows the results for both MNIST and SVHN. Using a smaller number of quantization bits increases performance up to a point because the learner can take advantage of very short encounters when using a smaller number of bits to represent the model. At some point , the quantization is too much and the model diverges. DR-OppCL's approach to preventing the quantization from straying too close to (or past) q_{min} allows it to take advantage of heterogeneous communication opportunities without sacrificing accuracy. Table II displays this tradeoff more concretely, comparing the number of bits communicated for each q_{min} versus the achievable accuracy. For instance, $q_{min} = 4$ bits achieves a good balance between accuracy and communication overhead for both models. It is future work to further validate whether this setting can be generalized across diverse datasets and models. The result also shows that the setting of minimum quantization level can be based on application-specific requirements (e.g., computation vs. accuracy). Irrespective of the setting, the dynamically set quantization bits for each encounter (q^e in Equation 2) balances communication constraints and accuracy.

Combining Dropout and Quantization. Next, we combine dropout and quantization using uniform distributions of device computational capabilities and encounter lengths. Fig. 5 shows five approaches for comparison: baseline, which assumes no dropout and no quantization; three static approaches that fix the values of dropout and quantization; and the dynamic DR-OppCL approach. Table III compares the top-performing static configuration with DR-OppCL. For each model, we chose



(a) MNIST, uniform

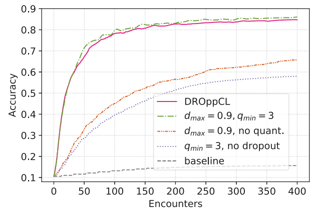


(b) SVHN, uniform

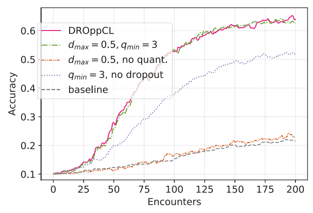
Fig. 4: Comparing different q values in mixed quantization.

TABLE II: Comparison of q_{min}

		uniform		inverse uniform	
	q_{min}	Bit(MB)	Accuracy(%)	Bit(MB)	Accuracy(%)
MNIST	2	642.18	42.44	362.06	20.27
	3	569.77	86.85	262.97	83.64
	4	446.86	85.81	197.23	79.50
	5	298.23	82.33	121.01	73.48
	6	160.07	75.29	40.02	43.43
SVHN	2	1164.40	22.74	746.52	19.00
	3	1042.52	68.83	620.28	65.23
	4	911.93	67.73	496.23	58.63
	5	694.29	60.98	330.82	42.20
	6	378.70	37.53	156.70	18.78



(a) MNIST



(b) SVHN

Fig. 5: DR-OppCL vs. best d_{max} and q_{min}.

a statically configured setting that used the empirically best quantization and dropout rates as the baseline.

DR-OppCL matches the performance of the best statically configured settings. As shown in Table III, DR-OppCL achieves nearly the same accuracy for MNIST at a dramatically lower overall computational and communication cost. For SVHN, the overhead values are very nearly identical, but the difference here is that DR-OppCL does not need to be configured *a priori* to the optimal levels of dropout and quantization, and was able to discover both on its own.

B. Real-world Scenarios

For our real-world scenarios, we used both generated and real-world mobility traces. For the former, we used the Levy walk mobility model [24], following the same setup in [18] with 45 devices (which act as both learner and neighbor) and 10 episodes where each episode has 200 timesteps. We also used HYCCUPS [5] to evaluate DR-OppCL under more realistic encounter patterns. HYCCUPS comprises traces from 72 participants over 63 days. We selected the 20 devices with the largest number of encounters and included encounters only for these devices. In the resulting traces, there were a total of 4,376 encounters for HYCCUPS (approximately 200 per device). We used the real durations of encounters from the traces as input to Equation 2. We assumed a Bluetooth connection with datarate of 2Mbps.

TABLE III: Comparison of top-performers from Fig. 5

		MFLOPs	Bit(MB)	Accuracy(%)
MNIST	$d_{max} = 0.9, q_{min} = 3$	24.13	1132.67	86.06
	DR-OppCL	19.84	754.96	84.76
SVHN	$d_{max} = 0.5, q_{min} = 3$	154.20	794.40	62.89
	DR-OppCL	154.20	794.40	63.95

TABLE IV: Types of devices.

	Edge Devices	Mobile Phones 1	Mobile Phones 2	IoT Devices	IoT Sensors
d^{infer}_{2NN}	0	0	0.2	0.4	0.6
d^{train}_{2NN}	0	0.2	0.4	0.6	0.8
d^{infer}_{CNN}	0	0	0.5	0.75	-
d^{train}_{CNN}	0	0.5	0.75	-	-

We modeled the computational capabilities of five types of devices (Table IV) and assigned a corresponding dropout rate to each device. These devices range from powerful edge devices that can train full models to lightweight IoT devices and sensors that are so resource constrained they cannot train even the most dropped-out version of the CNN model (though they can host a small model for inference and still encounter neighboring devices that help train their models). Depending on the device's capabilities, each type started with a different model, with weights initialized independently. For instance, a "Mobile Phone 2" device's personalized model was already 20% or 50% smaller than the largest possible 2NN or CNN model, respectively. The independence of the learners' models is an important aspect of the DR approach.

Figure 6 shows the final accuracy of all participating devices as a box plot under four different conditions: (1) *baseline* (no quantization or dropout); (2) *quantize* ($q_{min} = 3$; similar to [13, 27]); (3) *dropout* (choosing the dropout model dynamically); and (4) *DR-OppCL*. Table V shows the percentage of encounters a device uses compared to baseline. DR-OppCL tends to find more encounters to take advantage of because they enable collaborating with resource-constrained devices. For instance, in the SVHN-HYCCUPS experiment, DROppCL was able to utilize 239.5% of encounters to baseline, which was realized by requesting more resource-constrained devices and making use of encounters with short duration.

Overall, DR-OppCL achieves the highest average accuracy across all scenarios and reduced the variance of the accuracy across devices. Although the dropout and quantize strategies improve performance relative to the baseline, they often result in high variance, and their effectiveness is not always guaranteed. For instance, in the SVHN-Levy scenario, dropout led to greater performance improvement than quantize, enabling collaboration with resource-constrained devices. In contrast, in the MNIST-Levy scenario, utilizing shorter encounter durations appears to be critical for performance improvement, as quantize brought more improvement than dropout.

The HYCCUPS mobility trace exhibits a very skewed distribution of encounters per device—seven devices had fewer than five successful collaborations in the baseline for SVHN-HYCCUPS. However, because DR enables the learning algorithm to take advantage of a larger fraction of the encounters, in the case of DR-OppCL, we did not observe *any* device

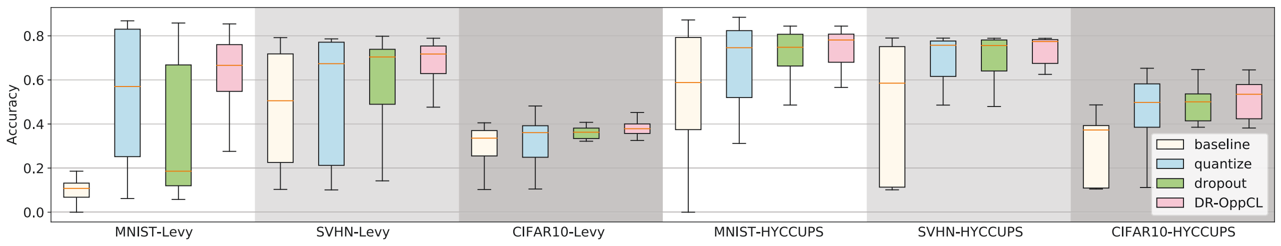


Fig. 6: Accuracies from simulated (Levy) and real-world (HYCCUPS) mobility traces.

TABLE V: Encounters used in comparison to baseline (%).

		Quantize	Dropout	DR-OppCL
Levy	MNIST	182.5	261.1	449.2
	SVHN/CIFAR-10	199.3	147.6	239.5
HYC	MNIST	123.2	198.2	198.5
	SVHN/CIFAR-10	358.2	262.7	333.3

with fewer than five collaborations, indicating that DR-OppCL enables the utilization of resource-constrained neighbors and encounters with shorter durations.

V. Conclusions and Future Work

This paper presented a generic Dynamic Reduction strategy for decentralized learning and an implementation of the strategy for OppCL (DR-OppCL). DR-OppCL extends OppCL to a more complex learning environment where devices and networks are heterogeneously resource-constrained. In contrast to opportunistic collaborative learning, our approach (1) removes the need for a central coordinator for initialization, making the approach truly decentralized; (2) addresses the heterogeneity of mobile computing devices' computational capabilities by using dropout models to reduce the complexity of the task demanded of collaborating neighbors; and (3) addresses the heterogeneity of mobile computing communication links by adaptively determining the best level of quantization used to communicate model parameters and gradient updates. Alongside these contributions, we derive a simple extension to a classic machine learning optimizer that allows integrating gradients learned using diverse dropout models. Our evaluation shows that DR-OppCL strikes a balance between maintaining high accuracy and optimally leveraging the resources in the neighbors and the network, reducing computation and communication resources contributed by nearby devices.

Acknowledgements

This work was funded in part by the National Science Foundation under grant CNS-1909221, Australian Research Council Project DP210102447, and ARC Linkage Project LP190100676. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and not necessarily reflect the views of the funding agencies.

References

[1] S. Caldas et al. "Expanding the reach of federated learning by reducing client resource requirements". In: *arXiv preprint 1812.07210* (2018).
[2] H. Chen and W. Lou. "Contact expectation based routing for delay tolerant networks". In: *Ad Hoc Networks* 36 (2016), pp. 244–257.
[3] S. Chen et al. "Dynamic Aggregation for Heterogeneous Quantization in Federated Learning". In: *IEEE Trans. on Wireless Comms.* (2021).
[4] S. Cho and C. Julien. "Chitchat: Navigating tradeoffs in device-to-device context sharing". In: *Proc. of PerCom.* 2016, pp. 1–10.
[5] R. I. Ciobanu and C. Dobre. *CRAWDAD dataset upb/hyccups (v. 2016-10-17)*. Downloaded from https://crawdad.org/upb/hyccups/20161017. Oct. 2016. DOI: 10.15783/C7TG7K.
[6] X. Dong, S. Chen, and S. Pan. "Learning to Prune Deep Neural Networks via Layer-wise Optimal Brain Surgeon". In: *NIPS*. 2017.
[7] Y. Du, D. Sailhan, and V. Issarny. "Let Opportunistic Crowdsensors Work Together for Resource-efficient, Quality-aware Observations". In: *Proc. of PerCom*. 2020.
[8] S. Han, H. Mao, and W. J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *Proc. of ICLR*. 2015.
[9] C. Hu, J. Jiang, and Z. Wang. "Decentralized federated learning: A segmented gossip approach". In: *arXiv preprint 1908.07782* (2019).
[10] C. Julien et al. "BLEnd: Practical Continuous Neighbor Discovery for Bluetooth Low Energy". In: *Proc. of IPSN*. 2017.
[11] J. Kang et al. "Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory". In: *IEEE Internet of Things Journal* 6.6 (2019).
[12] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint 1412.6980* (2014).
[13] A. Koloskova, S. Stich, and M. Jaggi. "Decentralized stochastic optimization and gossip algorithms with compressed communication". In: *Proc. of ICML*. 2019, pp. 3478–3487.
[14] J. Konečný et al. "Federated Learning: Strategies for Improving Communication Efficiency". In: *arXiv preprint 1610.05492* (2017).
[15] A. Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. University of Toronto, 2009.
[16] A. Lalitha et al. "Fully decentralized federated learning". In: *Third workshop on Bayesian Deep Learning (NeurIPS)*. 2018.
[17] Y. LeCun et al. "Gradient-based learning applied to document recognition". In: *Proc. of the IEEE* 86.11 (1998), pp. 2278–2324.
[18] S. Lee et al. "Opportunistic Federated Learning: An Exploration of Egocentric Collaboration for Pervasive Computing Applications". In: *Proc. of PerCom*. 2021.
[19] D. Li and J. Wang. "Fedmd: Heterogenous federated learning via model distillation". In: *arXiv preprint 1910.03581* (2019).
[20] X. Li et al. "Communication efficient decentralized training with multiple local updates". In: *arXiv preprint 1910.09126* 5 (2019).
[21] B. McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Proc. of AISTATS*. 2017.
[22] Y. Netzer et al. "Reading Digits in Natural Images with Unsupervised Feature Learning". In: *Proc. of NIPS* (Jan. 2011).
[23] C. Pappas et al. "Ipls: A framework for decentralized federated learning". In: *Proc. of IFIP Networking*. 2021, pp. 1–6.
[24] I. Rhee et al. "On the Levy-Walk Nature of Human Mobility". In: *IEEE/ACM Transactions on Networking* 19.3 (2011), pp. 630–643.
[25] S. Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint 1609.04747* (2016).
[26] N. Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.
[27] H. Taheri et al. "Quantized decentralized stochastic learning over directed graphs". In: *Proc. of ICML*. 2020.
[28] Y. Ye et al. "EdgeFed: Optimized federated learning based on edge computing". In: *IEEE Access* 8 (2020), pp. 209191–209198.
[29] X. Zhang et al. "Net-fleet: Achieving linear convergence speedup for fully decentralized federated learning with heterogeneous data". In: *Proc. of MOBIHOC*. 2022, pp. 71–80.