Distributed Machine Learning Algorithms via Geometric Monitoring

Author:
Vissarion B. Konidaris

Supervisor:
Vasilis Samoladas

A thesis submitted in fulfillment of the requirements for the degree of Diploma in Electrical and Computer Engineering

January 2019

Abstract

Contemporary deep neural network architectures trained on massive datasets can provenly achieve state-of-the-art performance across a wide variety of domains, from image and speech recognition, to text processing, recommendation systems and fraud detection. With the explosion in the amount of data generated online entering its next phase, we are able to train bigger and deeper neural nets which can dramatically increase performance but also training time. What is more, most of the data is generated or received on different remote machines and its massive nature implies prohibitive communication costs if all data is to be collected at a single site. In view of these problems, much effort has been dedicated the past few years into parallelizing the training procedure of such complex models. We introduce a novel method for scaling up distributed training of deep neural networks using the Functional Geometric Monitoring (FGM) communication protocol, a well studied technique that is used to monitor complex continuous queries on high-volume, rapid distributed streams. This protocol is suitable for classic learning with stationary environment properties, as well as non-stationary ones with concept drift. Our goal is to minimize the prediction loss and network communication at the same time. We demonstrate empirically that the protocol achieves up to 95% less network communication than todays' cutting edge methods, while achieving high predictive performance.

Contents

1 Introduction

The rate by which data is generated online has taken on a surge the past few years, from traditional time series and appliances measurements, to complex unstructured information like images, videos and sound. Moreover, the data is frequently generated across multiple remote devices and computational machines with rapid pace, constituting its processing challenging, as their centralization can become practically infeasible. Statistical analysis and machine learning (ML) algorithms are key to transforming the seemingly uncorrelated data into valuable knowledge. Thus, it is of great value to adapt the state-of-the-art learning techniques into the distributed data environments in order to be able to harness their true potential.

Deep learning (DL) techniques have shown to have robust predictive performance but also a perplexing and time consuming training process. Fortunately, the training of artificial neural networks (ANN) utilize gradient descent methods for minimizing their loss, a fact that brings into being opportunities to distribute their training procedure. Many studies have focused on distributed deep learning, but very few have taken into account the overwhelming communication costs that such a procedure requires. In this work, we focus on distributing the learning process of convolutional neural networks (CNN) [32] and Extreme Learning Machines (ELM) [24, 25, 26, 27, 29, 30, 31], while minimizing the communication of the remote sites.

1.1 Related Work

1.1.1 Deep Learning

Deep learning and ANN learning in general, being a sub field of machine learning algorithms, have recently drawn the attention of research and industry alike due to their great promise in the field of data analysis. ANNs were originally developed in the late 80s, but it was only until recently that were brought back into the forefront. This is a consequence of new training techniques, that were able to alleviate some of their shortcomings, and also of the huge amount of data that is currently accessible online. The well studied approaches to parallelizing/distributing the training computation are model parallelism and data parallelism. In this study, we will focus solely on data parallelism. The most intuitive and common distributed optimization methods for training ANNs are the Synchronous and Asynchronous Distributed Stochastic Gradient Descent methods [1, 2, 3, 4, 5, 6, 7, 8] or a combination of those two.

1.1.2 Monitoring non-linear functions on Distributed Streams

The trend in the information infrastructure of modern societies induce challenges in centralizing data that is generated from multiple sources in a stream like fashion, as the massive nature of data implies prohibitive communication costs if all data is to be centralized at a single processing machine. Motivated by such needs, there has been significant research effort on monitoring complex continuous queries on high-volume, rapid distributed streams. Sharfman [9, 10], introduced the Geometric Method for monitoring non-linear functions over distributed streams by utilizing convex analysis theory. Vasilis Samoladas and Minos Garofalakis [16] generalized this idea by monitoring geometric constraints on distributed succinct summaries of streams, such as histograms, sketches, or more generally, high-dimensional vectors.

1.2 Our Contribution

On this study we aim to introduce a general framework for the problem of distributed machine learning, by utilizing the advancements in the field of distributed stream monitoring. We focus on a supervised leaning task with ANNs as learning models, but the method requires minimum, if any, alterations to be compatible with other parametric learning models.

2 Preliminaries

In this section we formally introduce the distributed prediction task and dive our selves deeper into previous research done on distributed neural network training, as well as on approximate query monitoring on distributed data streams. We recall simple batch learning algorithms for training neural networks.

2.1 Notation

We consider a distributed computing environment with a classic star network topology, comprised of k remote sites/nodes and a central hub/coordinator. Each site has a copy of the neural model, and has a subset of the whole data set or receives a data stream (for vanilla and online learning setting respectively). The training is done solely on the k remote sites, and the coordinator is responsible for synchronizing all the models into one global predictive model. Our goal is to minimize the amount of communication needed between the nodes to reach to consensus, while preserving the accuracy of the global model to competitive levels. We denote the parameters of the neural network as $w \in \mathbb{R}^D$ and the size of the mini-batch as m. We also assume that the total size of the unified data set is M. Lastly, because we are mainly concerned about the communication cost that these algorithms impose on the distributed network topology, we view this cost in terms of total bytes transmitted.

2.2 Distributed training of ANNs utilizing gradient methods

Two of the most widely used architectures for distributively training neural networks are model parallelism and data parallelism. In model parallelism, each processing machine is responsible for the computations of only a part of a single ANN. For example each local site may be assigned with the computations of a single layer. In data parallelism, each local site has a complete copy of the neural network and a subset of the whole data set. The training takes place in each individual copy and the models are combined in some way to come up with a single unified predictor. These two methods however are not mutually exclusive, as they can be combined into a hybrid approach. We however focus our study on data parallelism due to its fault tolerance properties, a property that is of high importance in distributed settings, and its simplicity to implement. The topology is completely identical to our star network topology, with k remote sites with a copy of the ANN and a central hub that orchestrates the learning procedure.

2.2.1 Gradient methods

In optimization problems, the gradient method is an algorithm that solves problems of the form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x),$$

with the search directions defined by the gradient of the function at the current point. The gradient descent (GD) algorithm, being a gradient method, is one of the most popular algorithms to perform optimization, and by far the most used algorithm to train neural networks. The training of a neural network is equivalent to minimizing a loss function $L(w)$ w.r.t the parameters $w \in \mathbb{R}^D$ of the network. This is accomplished by updating the parameters towards the opposite direction of the gradient of the loss function $\nabla_w L(w)$. GD can be be applied on the entire training set (Batch GD), or stochastically point by point (SGD) in cases where the size of the data set is too large. A common technique used to trained neural nets is a combination of those two techniques, the mini-batch GD.

For any mini-batch of m training examples the update rule for the weights of the model becomes

$$w = w - \alpha \nabla_w L(x^{(i:i+m)}, y^{(i:i+m)}; w),$$

where a is the learning rate and each pair (x, y) is a training point along with its label. Other variants of this algorithm have been implemented over the past years that are more robust in training neural networks like Momentum[], Adagrad[], Adadelta[], RMSProp[], Adam[], AdaMax[], Nadam[] and AMSGrad[]. A more general update rule that encompasses all of those algorithms is of the form

$$w = w + \Delta_w(x^{(i:i+m)}, y^{(i:i+m)}),$$

where $\Delta_w(\{x, y\})$ is the update term of each variant of the mini-batch GD algorithm that utilizes internally the learning rate a and the gradient $\nabla_w L(x^{(i:i+m)}, y^{(i:i+m)}; w)$ is some way.

2.2.2 Synchronous Distributed SGD (S-DSGD)

In the S-DSGD setting, the technique that is used to combine the k models is parameter averaging. Parameter averaging is done after each worker has fitted a mini-batch to its model, a procedure that imposes the constraint that each worker should observe the same number of examples with the rest. Assuming that the mini-batch is of size m, then the weight update rule by a single machine is given by

$$W_{t+1} = W_t - \frac{\alpha}{km} \sum_{i=1}^{km} \nabla_{W_t} L_i,$$

where L_i is the loss function suffered by the training example i. When we separate this formula to k parallel workers the update rule becomes

$$W_{t+1} = \frac{1}{k} \sum_{i=1}^{k} W_t^i = \frac{1}{k} \sum_{i=1}^{k} \left(W_t - \frac{\alpha}{m} \sum_{j=(i-1)m+1}^{im} \nabla_{W_t} L_j\right) = W_t - \frac{\alpha}{km} \sum_{i=1}^{km} \nabla_{W_t} L_j,$$

a formula that is identical to the single machine setting. Regarding a mini-batch fit for each worker as a round, then in each of these rounds the communication cost incurred by the messages that are send from the workers to the hub is $\Theta(kD)$, and the communication cost incurred by the messages send from the hub to the workers is again $\Theta(kD)$. Hence, the total communication cost of a single round is $\Theta(2kD)$ bytes. Assuming that the entire data set is of size M and that each worker fits m examples to its model per round, then the total number of rounds is $\lceil \frac{M}{km} \rceil$. Consequently, the grand total communication cost of S-DSGD algorithm in terms of bytes is $\Theta(2D\lceil \frac{M}{m} \rceil)$. The algorithm of this distributed training process proceeds as follows:

1. The hub initializes the parameters of the ANN using any initialization technique.
2. Broadcast the parameters to each worker/local site.
3. Train each worker on a mini-batch subset of the data using mini-batch SGD.
4. All workers send their parameters to the hub were they are averaged.
5. While there is more data return to step 2.

Parameter averaging is usually performed after each iteration, or else after each site has observed and fitted a single mini-batch. Although this approach improves dramatically the convergence properties of the training process, it also introduces a considerable amount of overhead to the network. Previous research on the subject [4], suggests that averaging once every 10 to 20 mini-batches per worker can still perform well, exchanging the predictive performance for communication gain. In addition, even though S-DSGD outperforms other distributed learning techniques in terms of predicting accuracy, it suffers from the so called 'last-executor' affect, meaning that a synchronous system like this will always have to wait on the slowest executor before completing

each iteration. This can be a problem when the total number of workers increases, making S-DSGD a non viable solution in large distributed settings.

2.2.3 Asynchronous Distributed SGD (A-DSGD)

A conceptually similar approach to parameter averaging is the update-based data parallelism. In this setting, only the updates of the parameters are send by the nodes instead of the parameters themselves. Additionally, each site sends its update as soon as the fitting process completes, and the hub immediately adds this term to the global weights. The hub then sends back to the site the new updated parameters. Hence, the updates are done in an asynchronous manner. This gives an update of the form

$$W_{t+1} = W_t + \frac{1}{k}\sum_{i=1}^{k} \Delta W^i,$$

with ΔW^i encompassing, as before, the update that the optimizer imposes after the observation of a mini-batch by site i (i.e. for the vanilla DG optimizer, the update term for a worker i is $-\frac{\alpha}{m}\sum_{j=1}^{m} \nabla_{W^i} L(x_j^i, y_j^i)$). The communication cost is similar to the S-DSGD method.

A-DSGD, being an asynchronous method, gets rid the 'last-executor' problem, thus gaining a big advantage against S-DSGD in terms of data throughput, and hence execution speed. Workers can also be tuned to apply their gradients after more than one mini-batch has been observed, just as in S-DSGD, providing further throughput gains. On the downside, A-DSGD on its simplest form can result to high staleness values for the gradients. This means that, as the calculation of gradients take time, by the time a worker has finished these calculations and applies its results to the global parameters, the parameters may have been updated a number of times. In realistic scenarios this problem can slow down the predictor's convergence significantly. Many variants of A-DSGD try to alleviate this problem and have been shown to improve convergence over the naive implementation of the A-DSGD algorithm, by utilizing some form of synchronization procedure [6, 7].

2.2.4 S-DSGD A-DSGD variants

The S-DSGD and A-DSGD algorithms are the most commonly used methods when it comes to distributively training neural networks, each of them having their advantages and their shortcomings. Many variants of those two algorithms have been formed over the last few years, some of them involving hybrid implementations by combining the methods in various ways. For our analysis we will stick to these two for simplicity. Concluding, S-DSGD, A-DSGD and their variants, can dramatically decrease the training time of ANNs, but they do not take into account the communication costs that bring upon to the distributed network topology. Their communication depends entirely upon the predefined number of mini-batches that a worker needs to observe before its able to communicate with the network. There is not a true mechanism that actually defines if communication is necessary. Michael Kamp [17] referred to these methods as "Static Averaging Protocols". In the upcoming sections we discuss some protocols that are used to monitor distributed streams and see how Kamp utilized one to create the first "Dynamic Averaging Protocol" for distributed learning.

2.3 Geometric monitoring for distributed data streams

Monitoring complex continuous queries on high-volume, rapid distributed streams is a complex and a non-trivial problem. The Geometric Method for monitoring non-linear functions over distributed streams, introduced by Sharfman [9, 10], is a communication protocol for distributed topologies that tackles this problem efficiently, by utilizing convex analysis theory. Vasilis Samoladas and Minos Garofalakis [12, 15, 16] generalized this method by monitoring geometric constraints on distributed succinct summaries of streams, such as histograms, sketches, or more gen-

erally, high-dimensional vectors. On this section we do some background checking on the classic Geometric Method (GM) that ignited a rich line of work in the *distributed streaming* domain.

2.3.1 Approximate Query Monitoring

We now present the Approximate Query Monitoring protocols used for monitoring complex non-linear queries over distributed streams. We adopt the standard data model for distributed streams, presented at section §2.1. At each site, a local stream is generated or received, denoted as a very high dimensional vector $V = \mathbb{R}^D$. This vector can be a frequency vector of the stream records or a linear sketch thereof. The succinct summary vector V gets updated at each site as stream updates arrive. Let $\boldsymbol{S}_i(t), i = 1...k$ denote the local state vectors. Every site communicates with the central coordinator/hub, where users pose queries on the global stream. Without loss of generality, assume that at time t the true global stream state is the average of the local stream states, i.e., $\boldsymbol{S}(t) = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{S}_i(t)$.

Typically, a continuous query on the global $Q(\boldsymbol{S}(t))$ is a highly complex, non-linear function of the global stream state $\boldsymbol{S}$. In order to reduce communication costs between the hub and the local sites, the user can tolerate some small bounded error to the query answer. More precisely, the coordinator does not possess the actual global stream state $S(t)$, but a close estimate $\boldsymbol{E}(t)$ of it, rendering an approximate query answer $Q(\boldsymbol{E}(t))$, with guarantee that, for some user defined error ϵ, at any time t it will be

$$Q(\boldsymbol{S}(t)) \in (1 \pm \epsilon)Q(\boldsymbol{E}(t)). \tag{1}$$

This guarantee is respected by the local sites that periodically publish their updated local state vectors $\boldsymbol{S}_i$ to the hub. Let $\boldsymbol{E}_i$ denote the local state vector last send by site i to the coordinator, and let $\boldsymbol{E} = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{E}_i$. Then, until the global estimate $\boldsymbol{E}$ is updated, and as long as the true unknown global stream state $\boldsymbol{S}(t)$ lies inside the *admissible region*

$$A = \{x \in V | Q(x) \in (1 \pm \epsilon)Q(\boldsymbol{E})\}, \tag{2}$$

no local site needs to communicate with the hub. Thus, the problem of approximately tracking the continuous query $Q(\boldsymbol{S}(t))$ can now be viewed as a problem of monitoring the geometric condition $\boldsymbol{S}(t) \in A$. When this condition is violated, it is necessary to update the estimate $\boldsymbol{E}$, in order to restore the system invariant of Eq. 1.

2.3.2 Geometric Monitoring Protocol

The Geometric Monitoring (GM) protocol is a general-purpose distributed algorithm for monitoring the invariant of the Eq. 1. The protocol works in rounds, where each round starts when a new estimate $\boldsymbol{E}$ is established at the hub, and lasts until the estimate is updated.

At the initial time of each round, T_{init}, the coordinator has perfect system knowledge, that is, each local state vector $\boldsymbol{S}_i$ haw been transmitted to the coordinator by all nodes. Consequently, $S(T_{init}) = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{E}_i = \boldsymbol{E}$. Then, the coordinator chooses a "good" *safe zone* $Z \subseteq A$, that is based on $\boldsymbol{E}$ and is convex subset of A, with $\boldsymbol{E} \in Z$, and transmits it to all sites.

At each local site, for every time t, the protocol maintains a drift vector $\boldsymbol{X}_i(t)$, so that the following *drift invariant* is satisfied:

$$\frac{1}{k}\sum_{i=1}^{k} \boldsymbol{X}_i(t) = \boldsymbol{S}(t). \tag{3}$$

A the beginning of each round, when $t = T_{init}$, this can be achieved by selecting $\boldsymbol{X}_i(T_{init}) = \boldsymbol{E}$ for all sites $i = 1, ..., k$. As a local stream update arrives at some remote site i at time t, the invariant is maintained by adding to $\boldsymbol{X}_i$ the vector $\boldsymbol{S}_i(t) - \boldsymbol{S}_i(t-1)$. Also, at any node i and any time t we refer to the quantity $\boldsymbol{\Delta}_i(t)$ as the delta vector, which is the actual change of the local stream state of the site i since the last round. Hence, the delta vector of each site is actually $\boldsymbol{\Delta}_i(t) = \boldsymbol{X}_i(t) - \boldsymbol{E}$. It is evident that at the start of each round the drift vector $\boldsymbol{\Delta}_i(t)$ is equal to the zero vector for all $i = 1, ..., k$.

Each local site monitors the local condition $\boldsymbol{X}_i(t) \in Z$. If this condition holds at every site, then by convexity of Z and the drift invariant, it will also be the case that $\boldsymbol{S}(t) \in Z$, and therefore, $\boldsymbol{S}(t) \in A$.

When a violation of the local condition $\boldsymbol{X}_i \in Z$ occurs, the site i signal the hub. In the simplest protocol, the round ends; the coordinator notifies sites to transmit updates collected during the round. This can be done by by shipping either every local state vector $\boldsymbol{X}_i$ or every local delta vector $\boldsymbol{\Delta}_i$ to the coordinator. Then, the coordinator updates $\boldsymbol{E}$ and starts a new round.

2.3.3 Performance

In order to analyze the communication cost, we will adopt a convenient notion of time; we assume that time is discrete and that exactly one local state vector is updated at each time step. In other words, after T time steps, the monitoring algorithm has processed exactly T local state vector updates. Since we have already assumed that the total size of the unified data set is M, we know that at each time $T \leq M$. To make the analysis more concrete we divide the communication into two parts. These parts are the *downstream cost* C^{down} of data messages from workers/local sites to the coordinator/hub and the *upstream cost* C^{up} of data and control messages from the hub to local sites. Lastly, we assume that the protocol at time T has performed n rounds, each round lasting τ_r steps, $r = 1, ..., n$.

In downstream communication cost of round r, C_r^{down}, is essentially the cost of transmitting some description of local stream data to the coordinator. This is done by flushing the new local state vector to the coordinator, constituting the transmission cost of a single site to $\Theta(D)$. The communication cost, therefore, depends on the dimensionality of the local state vector. In general, it is possible to send the actual local stream updates unencoded, if the communication cost of doing so is less. Therefore, $C_r^{down} = O(min\{\tau_r, kD\})$, and over n rounds the total downstream cost is $O(T)$. However, we will not endorse this technique on this study, as we want to utilize this protocol for machine learning problems, where in some cases sending the training examples through the network is prohibited due to security reasons (i.e. when we are dealing with private medical data). Hence, $C_r^{down} = O(kD)$ and the total downstream cost over n rounds is $C_r^{down} = O(nkD)$ for private-sensitive data in the machine learning scenario.

The upstream cost C_r^{up} of round r is essentially the cost of shipping the vector $\boldsymbol{E}$ to each site, i.e., $\Theta(D)$ bytes per site, since the local site can then both construct the safe zone according to some pre-arranged algorithm and also initiate the local drift vector. Hence, the total upstream cost of round r is $C_r^{up} = \Theta(kD)$ and over n rounds $C^{up} = O(nkD)$. Under high variability, skew between local stream data, very different local stream rates, or other adversarial conditions, the number of rounds n may grow to be large, making the communication cost quite large and heavy.

We know define the communication gain achieved by the GM protocol after the pass over the whole distributed data set. In the naive setting, each worker sends its state vector to the coordinator after every stream update, or after every mini-batch fitting, making the downstream communication cost equal to $D\lceil\frac{M}{m}\rceil$ bytes and the upstream communication cost to 0 bytes. As a result, the communication gain G of the GM protocol is defined as

$$C = D\lceil\frac{M}{m}\rceil - C^{down} - C^{up}. \tag{4}$$

When the above metric is turned out to be positive, then the GM protocol provides an advantage due to the reduced communication cost. Conversely, when it is negative then this is an indication that the protocols' use is detrimental. As Samoladas and Garofalakis mention at [], we can view the downstream gain $G_0 = D\lceil\frac{M}{m}\rceil - C^{down} \geq 0$ as some *profit*, and the C^{up} term as an *investment* whose purpose is to increase profit.

Under normal circumstances and conditions, the total cost of n rounds is approximately $O(nkD)$. Therefore, we know have an intuition and good heuristic rule about what we can increase the communication gain. Other things being equal, the communication gain of the GM protocol increases when rounds last longer. Thus, the main objective is to minimize the number of rounds of the protocol.

2.3.4 Rebalancing in the GM protocol

By observing the algorithm of the GM protocol you may have noticed that when a local violation happens at a remote site i, it is not necessarily the case that $S \notin Z$. This is because, in all other remote sites $j \neq i$, is still the case that $\boldsymbol{X}_j \in Z$. To make thing more concrete, assume that since the beginning of a round, all the stream updates have been sent to the remote site i, whose drift vector $\boldsymbol{X}_i$ does not belong in the convex set Z but it's very close to its boundary. Then, it is true that all the other drift vectors $\boldsymbol{X}_j$ are still equal to $\boldsymbol{E}$, for $j \neq i$. It is easy now to recognize that incurring a communication cost of $O(kD)$ of needlessly replacing Z at the first local violation is potentially wasteful.

An improvement of the GM protocol that addresses situations like these are the rebalancing protocols. These protocols are mostly heuristic and their goal is to adjust some of the drift vectors in order to restore the local conditions at all sites with the hope of reducing the communication cost further. The rebalancing protocol goes like this. Starting with the set $B = \{i\}$, the site with that suffered a local violation, repeatedly add each new local site index to B (using any heuristic rule), and at each step compute the mean state vector $\boldsymbol{X}_B = \frac{1}{|B|} \sum_{j \in B} \boldsymbol{X}_j$ for all nodes in B. If $\boldsymbol{X}_B \in Z$, then reset the drift vector that for all the nodes that are in B to $\boldsymbol{X}_B$ and continue the round normally. If no such set B is found, in other words if $|B| = k$, then the round finishes.

The goal of the rebalancing procedures are to extent the lifetime of each round. There is not any mathematical proof that this is always the case, but many empirical studies have shown that such heuristic methods can deliver improved performance.

2.4 Communication-Efficient Distributed Online Prediction

Michael Kamp proposed at "Communication-Efficient Distributed Online Prediction by Dynamic Model Synchronization" [17], a distributed online prediction algorithm for distributively training linear online machine learning models. We will refer to this algorithm as DMS for "Dynamic Model Synhronization". DMS utilizes the GM protocol and has been tested on a linear Passive Aggressive model [22] and a rapidly drifting two-layer neural network using Passive Aggressive updates. Experiments showed that the protocol can reduce the communication up to 90% compared to the state-of-the-art static communication protocols like S-DSGD and A-DSGD.

2.4.1 The DMS protocol

The DMS protocol uses the same distributed setting that was introduced in section §2.1. The core idea of DMS is to perform partial averaging of the models residing at the remote sites only when their parameters have deviated "enough" from their previous average. Intuitively, there is no need to perform model averaging, and suffer communication costs as a consequence of that, when all models are already approximately equal. To that end DMS uses a simple measure to quantify the affect of synchronizations. That measure is the *variance* of the current local model configuration space, i.e., the real value $Var[\boldsymbol{S}(t)] = \frac{1}{k} \sum_{i=1}^{k} ||\boldsymbol{S}(t) - E||_2^2$. The protocol accepts a user defined positive threshold Δ. As long as the *variance* of the scattered models defined by the above equation is below the the threshold Δ, then the topology remains silent. Else, synchronization is performed. This is the basis of the DMS.

The algorithm decomposes the global condition $Var[\boldsymbol{S}(t)] \leq \Delta$ into a set of local conditions that can be monitored at their respective remote sites without communication. At each time t, each node i, for i=1,...,k, checks the local condition $||\boldsymbol{X}_i(t) - \boldsymbol{E}||_2^2 \leq \Delta$, where $\boldsymbol{X}_i(t) = w_i(t)$ (see [9, 13, 18, 14] for a more general description of this method). It is easily proven that if all local conditions $||\boldsymbol{X}_i(t) - \boldsymbol{E}||_2^2$ hold, then the global variance is bounded by Δ [[17], Theorem 6], i.e.

$$\frac{1}{k} \sum_{i=1}^{k} ||\boldsymbol{X}_i(t) - \boldsymbol{S}(t)||_2^2 \leq \Delta. \tag{5}$$

This follows directly from the fact that the current average vector $\mathbf{S}(t)$ minimizes the squared distances to all $\boldsymbol{X}_i$, i.e.

$$\frac{1}{k}\sum_{i=1}^{k}||\boldsymbol{X}_i(t)-\boldsymbol{S}(t)||_2^2 \leq \frac{1}{k}\sum_{i=1}^{k}||\boldsymbol{X}_i(t)-\boldsymbol{E}||_2^2 \leq \Delta. \tag{6}$$

The DMS protocol goes exactly like the GM protocol that is discussed and analyzed in section §2.3 with the succinct summary of the stream being the parameter vector of the predictor. The algorithm of DMS is summarized in Algo. 1. We see that DMS also uses the simplest GM rebalancing protocol described in §2.3.4.

Algorithm 1 Distributed Dynamic Synchronization Protocol

Initialization:

local models $X_1(1), ..., X_k(1) \leftarrow$ random initialization technique
reference vector $E \leftarrow \frac{1}{k}\sum_{i=1}^{k} X_i(1)$
violation counter $v \leftarrow 0$

Round t at node i:

observe $x_{t,i}$ and provide service/ make a prediction
observe $y_{t,i}$ and **update** the local model
if t mod $m = 0$ **and** $||X_i(t) - E||_2^2 > \Delta$ **then**
    send $w_{t,i}$ to coordinator (violation)

At coordinator on violation:

let B be the set of nodes with violation
$v \leftarrow v + |B|$
if $v = k$ **then** $B \leftarrow [k]$, $v \leftarrow 0$
while $B \neq [k]$ **and** $\frac{1}{k}\sum_{j\in B}||X_j(t) - E||_2^2 > \Delta$ **do**
    augment B by augmentation strategy
    receive models from nodes added to B
send model $\boldsymbol{S}(t) = \frac{1}{B}\sum_{j\in B} X_j(t)$ to nodes in B
if $B = [k]$ also set new reference vector $E \leftarrow \boldsymbol{S}(t)$

2.4.2 DMS for decentralized Deep Learning

Kamp, proved on [20], both theoretically and experimentally, that DMS can be used not only on convex objectives, but also on non-convex ones as well. More specifically, DMS was used to train CNNs, a non-convex optimization problem, by distributed mini-batch SGD. Kamp proved that if the loss function that we are trying to minimize is locally convex in an $O(\Delta)$-radius around the current average, then Theorem 2 in Boley et al. [19] guarantees that for SGD, DMS has a predictive performance similar to any static/periodically communicating protocol. We later on utilize DMS as a benchmark against our implementation, that makes use of another communication protocol which is a substantial improvement on the core ideas of GM.

2.5 Functional Geometric Monitoring for distributed data streams

Most work on GM has focused on the critical problem of choosing good safe zones for various complex queries, as the protocol's performance depends heavily on the quality of the safe zone. A substantial improvement on the GM protocol is the Functional Geometric Monitoring (FGM) protocol. The algorithm's foundation on convexity provides substantial benefits in terms of performance, scalability and robustness. The focus of this subsection is to present the basic principles and protocol of FGM.

2.5.1 Safe Functions

We start by examining the notion of a safe state in a monitoring algorithm. The system is in a safe state as long as $\frac{1}{k}\sum_{i=1}^{k}\boldsymbol{X}_i = \boldsymbol{S} \in A$. In the context of the GM protocol, system safety is conservatively monitored by watching the conjunction of all local conditions $\bigwedge_{i=1}^{k}\boldsymbol{X}_i \in Z$, where Z is a convex subset of the admissible region A. When this conjunction becomes false, the system restores it, either by restarting a round or by rebalancing.

FGM on the other hand, employs a real function $\phi : V \to \mathbb{R}$. Each remote site i, for $i = 1, ..., k$, tracks its ϕ-value, or else the value of function ϕ on their state vector $\boldsymbol{X}_i$ as it gets updated. Then, system safety is guaranteed as long as the global summation of those one-dimensional projections *sum* $\psi = \sum_{i=1}^{k}\phi(\boldsymbol{X}_i)$ is non-negative.

Definition 1. *(SAFE FUNCTION). A function $\phi : V \to \mathbb{R}$ is safe for admissible region A, if, for all $\boldsymbol{X}_i \in V, i = 1, ..., k$,*

$$\sum_{i=1}^{k}\phi(\boldsymbol{X}_i) \geq 0 \Rightarrow \frac{\sum_{i=1}^{k}\boldsymbol{X}_i}{k} \in A.$$

Hence, the problem of watching a boolean conjunction in order to detect a violation has been converted into a sum-monitoring problem. The introduction of safe functions offers significant opportunities for improved distributed stream monitoring.

2.5.2 Safe Functions and convexity

The selection of a good safe function is of great importance as it can greatly affect the communication efficiency of the system under the FGM protocol. Of course, not all safe functions for a particular admissible region A is equally desirable. It should be the case that ϕ should take as large values as possible, so that ϕ remains positive longer and prolong the rounds. Let $\phi_1 \leq \phi_2$ denote pointwise-dominance, i.e.

$$\forall \boldsymbol{x}, \quad \phi_1(\boldsymbol{x}) \leq \phi_2(\boldsymbol{x}).$$

If both functions are safe for A, then ϕ_2 is to be preferred as it is more probable to prolong the rounds. The properties of *maximal* safe function such as ϕ_2 are characterized by the following theorem.

Theorem 1. *. For any set A, if ϕ is safe for A, there exists a **concave** function $\zeta \geq \phi$ which is also safe for A.*

Based on the above theorem, the FGM protocol restricts its attention to safe functions that are concave.

For any function ϕ, denote the *level set* of ϕ, as

$$L(\phi) = \{\boldsymbol{x} \in V | \phi(\boldsymbol{x}) \geq 0\}.$$

For ϕ to be safe for some A, it is necessary that $L(\phi) \subseteq A$. For a concave function ζ, this is also sufficient

Proposition 1. *. A concave function ζ is safe for A, if and only if, $L(\zeta) \subseteq A$.*

Proof. To prove sufficiency, assume $L(\phi) \subseteq A$. By the definition of concave function, for any $k \geq 1$,

$$\zeta(\frac{\sum_{i=1}^{k}\boldsymbol{X}_i}{k}) \geq \frac{1}{k}\sum_{i=1}^{k}\zeta(\boldsymbol{X}_i).$$

Then, $\sum_{i=1}^{k}\zeta(\boldsymbol{X}_i) \geq 0$ implies $\zeta(\boldsymbol{S}) \geq 0$, and thus $\boldsymbol{S} \in L(\zeta) \subseteq A$. □

Furthermore, if ζ is concave, then the set $Z = L(\zeta)$ is closed and convex. Therefore, a concave safe function ζ for an admissible region A can be thought of as a functional representation for a convex safe zone $L(\zeta) \subseteq A$. Thus, FGM is conceptually a generalization of standard GM. Below follows the definition of the safe zone function.

Definition 2. *(SAFE ZONE FUNCTION). Given an admissible region A and a reference point $\boldsymbol{E}$, a safe zone function ζ is a concave function which is safe for A, and $\zeta(A) > 0$.*

A good safe zone depends heavily on the quality of the safe zone function. Criteria for the quality of a safe zone function are discussed in [15], where safe zone functions are used on the context of a methodology for the compositional design of high-quality safe zones for complex queries. The problem of designing or composing safe zone functions for particular queries can become quite useful, especially in the context of machine learning problems.

2.5.3 The basic FGM protocol

The basic FGM protocol monitors the threshold condition

$$\sum_{i=1}^{k} \zeta(\boldsymbol{X}_i) \geq 0, \tag{7}$$

over the duration of the round.

At the beginning of a round, the coordinator/hub knows the current state of the system $\boldsymbol{E} = \boldsymbol{S}$. It selects a safe zone function ζ for A, as defined by Eq. 2 and $\boldsymbol{E}$. At each point in time, let $\psi = \sum_{i=1}^{k} \zeta(\boldsymbol{X}_i)$. The round's steps are:

1. At the beginning of a round, the coordinator ships ζ to every single one remote site/worker/local site. It many cases it is sufficient to only shit the vector $\boldsymbol{E}$. Local sites initialize their drift vectors to $\boldsymbol{E}$. Hence, at the start of a round it is the case that $\psi = k\zeta(\boldsymbol{E})$.

2. After this, the hub initiates a number of subrounds, which are described below. A the end of all subrounds, $\psi \leq \epsilon_\psi k\zeta(\boldsymbol{E})$, for some small ϵ_ψ.

3. At last, the hub ends the round by collecting all drift vectors $\boldsymbol{X}_i$ and updating $\boldsymbol{E}$.

The goal of each subround is to monitor the condition $\psi \geq 0$ coarsely, with a precision of roughly θ, performing as little communication as possible. In fact, in each subround the communication cost is at most $3k$ one-word messages, k of which are upstream messages and the rest are downstream messages. Subrounds are executed as follows:

1. A the beginning of a subround, the coordinator knows the values of ψ. It computes the subround's *quantum* $\theta = \frac{\psi}{2k}$, ans ships it to each remote/local site. Additionally, the hub initializes a counter $c = 0$. Each local site records its initial value $z_i = \zeta(\boldsymbol{X}_i)$, where $2k\theta = \sum_{i=0}^{k} z_i$, and initializes a counter $c_i = 0$.

2. Each local site i maintains its local drift vector $\boldsymbol{X}_i$, as it processes stream updates. When $\boldsymbol{X}_i$ is updated, site i updates its counter

$$c_i := max\{c_i, \lfloor \frac{z_i - \zeta(\boldsymbol{X}_i)}{\theta} \rfloor\}.$$

   If this update increases the counter, the local site sends a message to the hub, with the increase to c_i.

3. When the hub receives a message with a counter increment from a local/remote site, it adds the increment to its global counter c. If the global counter c exceeds k, the hub finishes the subround by collecting all $\zeta(\boldsymbol{X}_i)$ from all local sites, recomputing ψ. If $\psi \leq \epsilon_\psi k\zeta(\boldsymbol{E})$, the subrounds end, else another soubround begins.

The following statement guarantees the correctness of the protocol.

Proposition 2. *During the execution of a subround, if $c \leq k$ then $\sum_{i=1}^{k} \zeta(\boldsymbol{X}_i) > 0$.*

Proof. At each point in time and for any site, it must be

$$\frac{z_i - \zeta(\boldsymbol{X}_i)}{\theta} - 1 < \lfloor \frac{z_i - \zeta(\boldsymbol{X}_i)}{\theta} \rfloor \leq c_i.$$

Summing both sides, we get $\frac{1}{\theta}(2k\theta - \psi) - k < c$, which simplifies to $\sum_{i=1}^{k} \zeta(\boldsymbol{X}_i) > (k-c)\theta \geq 0$. □

2.5.4 Performance

We know turn our attention on the performance of the FMG protocol and how it compares against the GM's. Each round r incurs an upstream cost of $C_r^{up} = \Theta(kD)$, in order to ship the reference vector $\boldsymbol{E}$ to all sites. Recall that the duration of a round is denoted as τ_r and its equal to the number of local stream updates during the round r. In general, the duration τ_r is much larger in the FGM protocol, resulting in much better performance.

Each round in FGM is consisted by a number of subrounds. Therefore, we must account for the cost of the protocol during subrounds. Each subround by itself costs only 3k+1 one-word messages, as there are k messages to broadcast the quantum θ, k messages to centralize the ζ-values to the coordinator at the end of the soubround, and up to $k+1$ downstream messages carrying counter increments. The actual number of subrounds within a round, denoted as q, is in principle unpredictable. It has been shown however in [11], that if ψ is a decreasing function of time (which is might not always be the case), then q is at most $log_2 \frac{1}{\epsilon_\psi}$. It practice though, it turns out that for $\epsilon_\psi = 0.01$ the number of subrounds per round is always at most 10 and almost always 7. In all cases, the total cost $O(kq)$ of all subrounds in a round, was dominated by the upstream cost $\Theta(kD)$. In principle, the cost $O(kq)$ was less than $\Theta(kD)$ by several orders of magnitude, even when the number of subrounds per round are high. This leads to fewer upstream messages due to the reduced number of total rounds compared to GM, making the FGM a much more communication efficient protocol for distributed stream monitoring. For more information refer to [16].

2.5.5 Rebalancing

In GM, rebalancing can prolong a round, but at substantial extra cost, which sometimes may exceed the resulting benefit. Rebalancing in FGM though we can rebalance a round without incurring additional communication cost. The motivation to rebalance in FGM stems from the fact that the boolean condition $\sum_{i=1}^{k} \zeta(\boldsymbol{X}_i) < 0$ does not necessarily imply that $\zeta(S) < 0$. This is true due the concavity properties of the concave safe zone function ζ, or more specifically because $\zeta(S) = \zeta(\frac{\sum_{i=1}^{k} \boldsymbol{X}_i}{k}) \geq \frac{1}{k} \sum_{i=1}^{k} \zeta(\boldsymbol{X}_i)$. Therefore, more often than not, the current safe zone function ζ may still be useful, and we would like to avoid the overhead of shipping a new safe zone function to the sites. FGM manages to avoid that overhead by a rebalancing protocol that changes the monitored constraint, $\psi \geq 0$, in a suitable manner.

Lets assume that the coordinator keeps a *balance vector* $\boldsymbol{B}$. At the beginning of a round the balance vector $\boldsymbol{B}$ is set to $\boldsymbol{0}$. During the duration of the round, sites update their drift vectors as local stream updates arrive. However, with rebalancing allowed, it is possible for a site to flush its current drift vector to the coordinator, during the round. When a flush occurs, the coordinator updates the balance, by adding $\Delta \boldsymbol{X_i} = \boldsymbol{X}_i - \boldsymbol{E}$ to it. After drift vector $\boldsymbol{X}_i$ is flushed, it is reset to $\boldsymbol{X}_i$. Let $B \subseteq \{1, ..., k\}$ be the set designating sites which are in *balance* mode. Sites in B will monitor a slightly different safe zone function; the only requirement on B is that B can be empty only if $\boldsymbol{B} = \boldsymbol{0}$. The invariant of Eq. 3 becomes

$$\boldsymbol{S} = \boldsymbol{E} + \frac{1}{k}\boldsymbol{B} + \frac{1}{k}\sum_{i=1}^{k} \Delta \boldsymbol{X}_i. \tag{8}$$

This can be rewritten as

$$\boldsymbol{S} = \frac{|B|}{k}(\boldsymbol{E} + \frac{1}{|B|}\boldsymbol{B} + \frac{1}{|B|}\sum_{i\in B}\Delta\boldsymbol{X}_i) \tag{9}$$

$$+ \frac{k-|B|}{k}(\frac{1}{k-|B|}\sum_{i\in B}\boldsymbol{X}_i). \tag{10}$$

To compute the safety condition in the presence of rebalance vectors, start with $k\zeta(\boldsymbol{S}) \geq 0$. Notethat, by concavity, it is $\zeta(\boldsymbol{E}+\boldsymbol{x}+\boldsymbol{y}) \geq \frac{1}{2}(\zeta(\boldsymbol{E}+2\boldsymbol{x}) + \zeta(\boldsymbol{E}+2\boldsymbol{y}))$. This property is used to separate $\boldsymbol{B}$ from $\sum\Delta\boldsymbol{X}_i$ in Eq. 9; after some manipulation, we get the safety condition

$$\frac{|B|}{2}\zeta(\boldsymbol{E} + \frac{2\boldsymbol{B}}{|B|}) \tag{11}$$

$$+\sum_{i\in B}\frac{1}{2}\zeta(\boldsymbol{E} + 2\Delta\boldsymbol{X}_i) \tag{12}$$

$$+\sum_{i\notin B}\zeta(\boldsymbol{X}_i) \tag{13}$$

$$\geq 0. \tag{14}$$

The inequality depicted above can be though of as the analog representation to the basic FGM monitoring condition Eq. 7; notice that when B = $\emptyset$, it is identical to Eq. 7. For general B, the left-hand inequality is consisted by the following three parts:

1. The item $\psi_B = \frac{|B|}{2}\zeta(\boldsymbol{E} + \frac{2\boldsymbol{B}}{|B|})$ is an information term that is known only to the coordinator, and it is initially 0. The coordinator updates this term, when a site flushes its local drift vector.

2. For each site that belongs to the set B, or else $\forall i$ such that $i \in B$, the site's contribution is changed to
$$\frac{1}{2}\zeta(\boldsymbol{E} + 2\Delta\boldsymbol{X}_i)$$
compared to Eq. 7.

3. For each site that does not belong in to set B, or else $\forall i$ such that $i \notin B$, the site's contribution is the same as in Eq. 7, meaning that is equal to $\zeta(\boldsymbol{X}_i)$.

If we define ψ analogous ti its previous definition, then we can write ψ as

$$\psi = \sum_{i\in B}\frac{1}{2}\zeta(\boldsymbol{E} + 2\Delta\boldsymbol{X}_i) \quad + \sum_{i\notin B}\zeta(\boldsymbol{X}_i).$$

Then the global condition can summarized into the condition

$$\psi_B + \psi \geq 0.$$

n order to monitor this condition, the only modification needed to FGM is in the selection of a specific purposed value of the quantum θ at the beginning of each subround, so that

$$2k\theta = \psi + \psi_B. \tag{15}$$

2.5.6 The rebalancing FGM protocol

The new protocol begins in the same manner as in the basic FGM protocol that was described in subsection §2.5.3, with $B = 0$. Therefore $\psi + \psi_B = \psi = k\zeta(\boldsymbol{E})$. At the end of all subrounds, it is $\psi < \epsilon_\psi k\zeta(\boldsymbol{E})$. Where the basic protocol would start a new round, the rebalancing protocol restores the invariant $\psi + \psi_B \geq 0$ as follows:

1. The coordinator asks all sites to send their local drift vectors $\boldsymbol{X}_i$, and adds all sites to set B.
2. When the coordinator receives each $\boldsymbol{X}_i$, it updates $\boldsymbol{B}$ by adding each sites delta vector $\boldsymbol{X}_i - \boldsymbol{E}$ to it.
3. After $\boldsymbol{X}_i$ is sent, the site i resets it to $\boldsymbol{E}$, or else it makes its delta vector $\Delta\boldsymbol{X}_i$ equal to $\boldsymbol{0}$, and the new contribution of the site to ψ is $\zeta(\boldsymbol{E})/2$.
4. As soon as all drift vectors have been received, the coordinator recalculates ψ and ψ_B to check if the condition $\psi + \psi_B \geq k\zeta(\boldsymbol{E})$ is restored. If the condition holds, then it starts a subround with quantum $\theta = (\psi + \psi_B)/(2k)$.
5. If on the other the hand the condition does not hold, then it starts a new round by computing and then shipping the new reference vector $\boldsymbol{E}$ to all sites.

2.5.7 Rebalancing protocol performance

The rebalancing manifested as above will cause additional downstream communication, but incurs no upstream cost. In the simple distributed stream monitoring, the FGM rebalancing protocol is considered to be conservative, since the total downstream cost can never exceed the size of the streamed data. In the machine learning scenario this can be done by allowing the sites to send the actual data points to the coordinator, if the data are not private-sensitive, instead of the actual parameters when the communication of doing so is less. Even if the round's duration is not extended by much, there is no added overhead to overcome; therefore, the *communication gain* G of a round can never decrease because of rebalancing. Quite the contrary, it is quite likely that it will increase. For more information about the performance of the rebalancing protocol refer to [16].

2.6 Data Stream and Concept Drift

We now present the two different online leaning scenarios that we will use in our experiments for this study. In traditional centralized off-line machine learning we have a single stationary dataset that we want to mine or built a model on it that describes it well. On the other hand, in online machine learning the data is generated on the fly by some unknown distribution. In this case, the trained model is updated incrementally whenever new data arrives, but it is usually the case that the continuously generated data will stem from a dynamic environment, causing a change to the characteristics of the data stream which may lead to the degradation of the predictor's performance. This challenging issue is known as concept drift (CD), in which statistical properties of the input features and target classes or values may shift over time. We assume two learning environments; a stationary one without CD (where the characteristics of the data points that are continuously generated are always the same), and a dynamic one with CD.

According to Gama et al. [23], the basic concept drift based on Bayesian decision theory in the classification problem for class output c and incoming data $\boldsymbol{\mathcal{X}}$ is

$$P(c|\boldsymbol{\mathcal{X}}) = P(c)\frac{P(\boldsymbol{\mathcal{X}}|c)}{P(\boldsymbol{\mathcal{X}})}. \tag{16}$$

Concept drift occurs when $P(c|\boldsymbol{\mathcal{X}})$ changes; for instance, $\exists\boldsymbol{\mathcal{X}} : P_{(0)}(\boldsymbol{\mathcal{X}}, c) \neq P_{(1)}(\boldsymbol{\mathcal{X}}, c)$, where $P_{(0)}$ and $P_{(1)}$ are the joint distributions at times $t_{(0)}$ and $t_{(1)}$ respectively. Gama et al. categorized the concept drift types as follows:

1. *Real Drift* (RD) refers to the changes in the $P(c|\boldsymbol{\mathcal{X}})$ term. The change in $P(c|\boldsymbol{\mathcal{X}})$ may be caused by a change in the class boundary (the actual number of classes) or the class conditional probabilities (likelihood) $P(\boldsymbol{\mathcal{X}}|c)$. The number of classes expanded and different class of data that may come alternately are both known as recurrent context. A drift, where new conditional probabilities replace the old ones while the number of classes remain the same, is known as sudden drift.

2. *Virtual Drift* (VD) refers to the changes in the distribution of the continuously generated data. This is caused by changes happening to $P(\boldsymbol{\mathcal{X}})$. These changes may be due to incomplete or partial feature representation of the current data distribution. The predictor is built with additional data from the same environment without overlapping the true class boundaries.

3. *Hybrid Drift* (HD) happens when RD and VD happens at the same time.

We assume the training data come from S different concepts which we denote as C_S. To express a concept drift event, we use the symbol $\underset{VD}{\Longrightarrow}$, where the subscript font shows the drift type. For example, Concept 1 has virtual drift event to be replaced by Concept 2: $\boldsymbol{C}_1 \underset{VD}{\Longrightarrow} \boldsymbol{C}_2$. Concept 1 has real drift event to be replaced by Concept 1 and Concept 2 recurrently in the shuffled composition $\boldsymbol{C}_1 \underset{RD}{\Longrightarrow} shuffled(\boldsymbol{C}_1, \boldsymbol{C}_2)$.

Not all machine learning algorithms are built to handle such dynamic environments. In fact, most of them are build having a stationary one in mind. For the stationary distributed environment we will train a Convolutional Neural Network (CNN) and for the dynamic one we will train an Extreme Learning Machine (ELM), a one layer feedforward neural network that is able to adapt on all the aforementioned drift scenarios. We now proceed on the next sections by describing those two learners.

2.7 Convolutional Neural Networks (CNNs)

CNNs are an extension of deep artificial neural networks that are primarily used to classify, cluster and recognize objects in images and video. The first successful applications of Convolutional Networks were developed by Yann LeCun [32] in 1990's. One of the best known CNN architectures is the LeNet architecture that was used to read zip codes and digits among others. We will address the problem of distributively training a LeNet architecture using an Adam optimizer. Because Adam is a variation of the classic GD optimization methods, the training of a LeNet network can be done by the S-DSGD A-DSGD algorithms. For the stationary distributed learning environment we will train a LeNet network.

2.8 ELMs

One of the popular machine learning methods is Extreme Learning Machine (ELM), a Single-Layer Feedforward Neural Network introduced by Huang [24]. ELM has gained popularity because of its simplicity, learning speed, good generalization and its ability to adapt on many non-stationary online learning scenarios.

2.8.1 ELM

Assuming that the classes are represented as C and the number of neurons of the single hidden layer are L, then $\boldsymbol{\mathcal{X}_{M\times d}}$ and $\boldsymbol{Y_{M\times |C|}}$ are the input matrix and one-hot labels of M examples respectively, $\boldsymbol{A_{d\times L}}$ and $\boldsymbol{b_{1\times L}}$ are the weights and biases of the hidden layer, $\boldsymbol{g}$ is any non-linear activation function, $\boldsymbol{\beta_{L\times |C|}}$ the learnable output weight matrix and $\boldsymbol{H_{M\times L}}$ is the hidden layer matrix. The output function of the ELM is formulated as

$$f_L = \sum_{i=1}^{L} \beta_i H(a_i, b_i, x), \tag{17}$$

where $\boldsymbol{H} = g(\boldsymbol{\mathcal{X}A} + \boldsymbol{b})$. The parameters $\boldsymbol{A}$ and $\boldsymbol{b}$ of the hidden layer are initialized uniformly at random within the range $[-1, 1]$ and remain unchanged throughout the hole learning process. The

solution of ELM training with the smallest error can be obtained when the output weight $\boldsymbol{\beta}$ is approximated by

$$\boldsymbol{\beta} = \boldsymbol{H}^{\dagger}\boldsymbol{Y}, \tag{18}$$

where $\boldsymbol{H}^{\dagger}$ is the pseudoinverse of $\boldsymbol{H}$, also known as Moore-Penrose generalized inverse of $\boldsymbol{H}$. The ELM learning is simply equivalent to finding the smallest least-squares solution to $\boldsymbol{\beta}$ of the linear system $\boldsymbol{H}\boldsymbol{\beta} = \boldsymbol{Y}$. Hence, ELM does not require using the backpropagation algorithm for training which can sometimes be computationally demanding.

$\boldsymbol{H}^{\dagger}$ can be approximated by left pseudoinverse of $\boldsymbol{H}$ as

$$\beta = (\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T\boldsymbol{Y}. \tag{19}$$

The above equation is called the *least squares solution* to $\boldsymbol{H}\boldsymbol{\beta} = \boldsymbol{Y}$. This method though cannot can not address incremental learning in this simple form. For this reason, Liang et al. [30] proposed an online learning adaptation of ELM named OS-ELM. The OS-ELM is capable of incremental learning by using mini-batch fitting, with dynamic mini-batch size. More precisely, in OS-ELM, if we have $\boldsymbol{\beta}_{(t-1)}$ from $\boldsymbol{H}_{(t-1)}$ filled by the m_1 training data, and $\boldsymbol{H}_{(t)}$ filled by m_2 incremental training data, then the output weights $\boldsymbol{\beta}_{(t)}$ can be approximated by solving

$$\boldsymbol{\beta}_{(t)} = \left(\begin{bmatrix} \boldsymbol{H}_{(t-1)} \\ \boldsymbol{H}_{(t)} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{H}_{(t-1)} \\ \boldsymbol{H}_{(t)} \end{bmatrix} \right)^{-1} \begin{bmatrix} \boldsymbol{H}_{(t-1)} \\ \boldsymbol{H}_{(t)} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{Y}_{(t-1)} \\ \boldsymbol{Y}_{(t)} \end{bmatrix}. \tag{20}$$

OS-ELM sequentially updates the least squares solution using the update rule

$$\boldsymbol{K}_{(t)} = \boldsymbol{K}_{(t-1)} + \boldsymbol{H}_{(t)}^T\boldsymbol{H}_{(t)} \tag{21}$$

$$\boldsymbol{\beta}_{(t)} = \boldsymbol{\beta}_{(t-1)} + \boldsymbol{K}_{(t)}^{-1}\boldsymbol{H}_{(t)}^T(\boldsymbol{Y}_{(t)} - \boldsymbol{H}_{(t)}\boldsymbol{\beta}_{(t-1)}), \tag{22}$$

which can be shown to be identical to the Eq. 20.

Algorithm 2 OS-ELM

Require: $\boldsymbol{\mathcal{X}}_{(t)} \in [-1,1]\mathbb{R}^{m_t \times d}$, $\boldsymbol{Y}_{(t)} \in [0,1]\mathbb{R}^{m_t \times |C|}$, $\boldsymbol{A}_{(t)}$, $\boldsymbol{b}_L$, $\boldsymbol{K}_{(t-1)}$, $\boldsymbol{\beta}_{(t-1)}$
Ensure: $\boldsymbol{\beta}_{(t)}$, $\boldsymbol{K}_{(t)}$

Compute $\boldsymbol{H}_{(t)} = g(\boldsymbol{\mathcal{X}}_{(t)}\boldsymbol{A}_{(t)} + \boldsymbol{b}_L)$
if $IncreaseHiddenNodes == \textbf{\textit{True}}$ **then**
  $\Delta\boldsymbol{A}_{d\times\delta L} = RandomNumbers([-1,1], \mathbb{R}^{\delta\times\delta L})$
  $\Delta\boldsymbol{b}_{\delta L} = RandomNumbers([-1,1], \mathbb{R}^{\delta L})$
  $\boldsymbol{A}_{(t)} = [\boldsymbol{A}_t \quad \Delta\boldsymbol{A}_{\delta\times\delta L}]$
  $\boldsymbol{b}_L = [\boldsymbol{b}_L \quad \Delta\boldsymbol{b}_{\delta L}]$
  $\Delta\boldsymbol{H}_{(t)} = g(\boldsymbol{X}_{(t)}\Delta\boldsymbol{A}_{d\times\delta L} + \Delta\boldsymbol{b}_{\delta L})$
  $\boldsymbol{K}_{(t)} = \begin{bmatrix} \boldsymbol{H}_{(t-1)} & \boldsymbol{0} \\ \boldsymbol{H}_{(t)} & \Delta\boldsymbol{H}_{(t)} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{H}_{(t-1)} & \boldsymbol{0} \\ \boldsymbol{H}_{(t)} & \Delta\boldsymbol{H}_{(t)} \end{bmatrix}$
  $\boldsymbol{\beta}_{(t)} = \boldsymbol{K}_{(t)}^{-1} \begin{bmatrix} \boldsymbol{H}_{(t-1)} & \boldsymbol{0} \\ \boldsymbol{H}_{(t)} & \Delta\boldsymbol{H}_{(t)} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{Y}_{(t-1)} \\ \boldsymbol{Y}_{(t)} \end{bmatrix}$
else
  $\boldsymbol{K}_{(t)} = \boldsymbol{K}_{(t-1)} + \boldsymbol{H}_{(t)}^T\boldsymbol{H}_{(t)}$
  $\boldsymbol{\beta}_{(t)} = \boldsymbol{\beta}_{(t-1)} + \boldsymbol{K}_{(t)}^{-1}\boldsymbol{H}_{(t)}^T(\boldsymbol{Y}_{(t)} - \boldsymbol{H}_{(t)}\boldsymbol{\beta}_{(t-1)})$
end if
return $\boldsymbol{\beta}_{(t)}$, $\boldsymbol{K}_{(t)}$

Although OS-ELM is capable of learning online, it does not address to online learning environments with concept drift. On the other hand, the AOS-ELM algorithm introduced by et al.[31] utilizes the OS-ELM method and is capable of handling virtual, real and hybrid concept drifts. AOS-ELM will be the method we are going to be using for our dynamic distribute learning environment.

AOS-ELM handles concepts drifts in a very simple and efficient way. If the data stream suddenly suffers from VD, AOS-ELM assigns the random coordinates in the new concept space, or more simply, it extends the parameters of each neuron with the appropriate random values. According to interpolation theory from ELM point of view and Learning Principle I of ELM Theory [26], the input weight and bias are independent of the training samples and their learning environment through randomization. Their independence is not only in initial training stage, but also in any sequential training stages. Thus, the method can adapt the input weights and biases on any sequential stages in order to handle additional feature inputs. In case of RD, the method assigns the equivalent projection coordinates in the new design space, or else it extends the output weight vector $\boldsymbol{\beta}$ by the total number of the newly observed classes. According to universal approximation theory [28], the AOS-ELM has the capability of handling real drift by modifying the output matrix with zero block matrix concatenation to change the size matrix dimension without changing the norm value. Zero block matrix means that the previous $\boldsymbol{\beta}_{(t-1)}$ has no knowledge about the new concept. ELM can approximate any complex decision boundary, as long as the output weights $\boldsymbol{\beta}_t$ are kept minimum when the number of output classes are increased. Lastly, AOS-ELM can extend the number of neurons in the hidden layer on the fly, for the purpose of increasing its predicting accuracy, but will not be using this attribute in our experiments.

Algorithm 3 AOS-ELMVD

Require: $\boldsymbol{\mathcal{X}}_{(t)} \in [-1,1]\mathbb{R}^{m_t \times d}$,
$\boldsymbol{A}_{(t-1)} = Model_{(t-1)} \cdot \boldsymbol{A}$
Ensure: $\boldsymbol{A}_{(t)}$

$d_{(t-1)} = SizeOfAttributes(Model_{(t-1)})$
$d_{(t)} = SizeOfAttributes(\boldsymbol{\mathcal{X}}_{(t)})$
if $d_{(t)} > d_{(t-1)}$ **then**
    $\delta d = d_{(t)} - d_{(t-1)}$
    $\Delta \boldsymbol{A}_{\delta d \times L} = RandomNumbers([-1,1], \mathbb{R}^{\delta d \times L})$
    $\boldsymbol{A}_{(t)} = \begin{bmatrix} \boldsymbol{A}_{(t-1)} \\ \Delta \boldsymbol{A}_{\delta d \times L} \end{bmatrix}$
else
    $\boldsymbol{A}_{(t)} = \boldsymbol{A}_{(t-1)}$
end if
return $\boldsymbol{A}_{(t)}$

Algorithm 4 AOS-ELMRD

Require: $\boldsymbol{Y}_{(t)} \in [0,1]\mathbb{R}^{m_t \times |C|}$,
$\boldsymbol{\beta}_{(t-1)} = Model_{(t-1)} \cdot \boldsymbol{\beta}$
Ensure: $\boldsymbol{\beta}_{(t-1)}$

$m_{(t-1)} = SizeOfClasses(Model_{(t-1)})$
$m_{(t)} = SizeOfClasses(\boldsymbol{Y}_{(t)})$
if $m_{(t)} > m_{(t-1)}$ **then**
    $\delta m = m_{(t)} - m_{(t-1)}$
    $\Delta \boldsymbol{\beta}_{L \times \delta m} = \boldsymbol{0}$
    $\boldsymbol{\beta}_{(t-1)} = \begin{bmatrix} \boldsymbol{\beta}_{(t-1)} & \Delta \boldsymbol{\beta} \end{bmatrix}$
end if
return $\boldsymbol{\beta}_{(t-1)}$

The AOS-ELM algorithm can be summarized in the following four pseudocodes. The first one, Alg. 2, depicts the OS-ELM learning algorithm, and the pseudoalgorithms 3, 4 depict the AOS-ELMVD, AOS-ELMRD algorithms that address the VD and RD scenarios respectively. By putting together of these algorithms to Alg. 5 we get the AOS-ELM method that we will be using to our experimental distributed environment.

Algorithm 5 AOS-EML

Require: $\boldsymbol{\mathcal{X}}_{(t)} \in [-1,1]\mathbb{R}^{m_t \times d}$, $\boldsymbol{Y}_{(t)} \in [0,1]\mathbb{R}^{m_t \times |C|}$,
$\boldsymbol{A}_{(t-1)} = Model_{(t-1)} \cdot \boldsymbol{A}$
$\boldsymbol{\beta}_L = Model_{(t-1)} \cdot \boldsymbol{\beta}$
$\boldsymbol{K}_{(t-1)} = Model_{(t-1)} \cdot \boldsymbol{K}$
$\boldsymbol{\beta}_{(t-1)} = Model_{(t-1)} \cdot \boldsymbol{\beta}$
Ensure: $Model_{(t)}$
1: $\boldsymbol{\beta}_{(t-1)}$ =AOS-ELMRD($\boldsymbol{Y}_{(t)}, Model_{(t-1)}$)
2: $\boldsymbol{A}(t)$ =AOS-ELMVD($\boldsymbol{\mathcal{X}}_{(t)}, Model_{(t-1)}$)
3: $(\beta_{(t)}, \boldsymbol{K}_{(t)})$ =OS-ELM($\boldsymbol{\mathcal{X}}_{(t)}, \boldsymbol{Y}_{(t)}, \boldsymbol{A}(t), \boldsymbol{b}_L, \boldsymbol{K}_{(t-1)}, \boldsymbol{\beta}_{(t-1)}, IncreaseHiddenNodes$)
4: $Model_{(t)} = SaveModel(\boldsymbol{A}_{(t)}, \boldsymbol{b}_L, \boldsymbol{K}_{(t)}, \beta_{(t)})$
5: **return** $Model_{(t)}$

3 Distributed ANN training via the FGM protocol

We now present our approach to distributed ANN training that utilizes the FGM protocol. Our method is a "Dynamic Averaging Protocol" that encourages communication between the workers and the coordinator only when it deems necessary. Once again, we adopt the distributed setting that we discussed in section §2.1, with the succinct summaries of the stream at each site being the parameters of the local predictor, or else $\boldsymbol{X}_i(t) = w_i(t)$. The distributed protocol is identical to the FGM protocol with some adjustments to the safe zone function, the averaging procedure and the rebalancing method. We proceed by introducing the ML-FGM protocol for distributed ANN training and ML aglorithms in general.

3.1 Admissible region and safe function for Learning

The DMS algorithm utilizes the safe zone condition of Eq. 6 to dynamically synchronize the learners. We now borrow this safe zone and we modify it to a real safe zone function $\zeta : V \to \mathbb{R}$, in order to be applicable to the FGM protocol.The admissible region is the convex level set

$$A = \{\boldsymbol{S} \in \mathbb{R}^D \ \mid \ T - ||\boldsymbol{S} - \boldsymbol{E}||_2^2 \geq 0\}, \tag{23}$$

hence we construct the function $\phi : V \to \mathbb{R}$ that is safe for A as

$$\phi(\boldsymbol{X}_i) = \frac{1}{k}\Big(T - ||\boldsymbol{X}_i - \boldsymbol{E}||_2^2\Big). \tag{24}$$

Therefore, the coordinator monitors the condition

$$\sum_{i=1}^{k} \phi(\boldsymbol{X}_i) = \frac{1}{k}\sum_{i=1}^{k}\Big[T - ||\boldsymbol{X}_i - \boldsymbol{E}||_2^2\Big] \geq 0. \tag{25}$$

The safe function $\phi : V \to \mathbb{R}$ is safe for A because

$$\begin{aligned} 0 \leq \sum_{i=1}^{k} \phi(\boldsymbol{X}_i) &= T - \frac{1}{k} \sum_{i=1}^{k} ||\boldsymbol{X}_i - \boldsymbol{E}||_2^2 \\ &\leq T - \frac{1}{k} || \sum_{i=1}^{k} \boldsymbol{X}_i - \boldsymbol{E}||_2^2 \\ &= T - || \frac{1}{k} \sum_{i=1}^{k} \boldsymbol{X}_i - \boldsymbol{E}||_2^2 \\ &= T - ||\boldsymbol{S} - \boldsymbol{E}||_2^2 \Longrightarrow \frac{1}{k} \sum_{i=1}^{k} \boldsymbol{X}_i \in A. \end{aligned}$$

3.2 Safe zone function for Learning

In order to ensure that the variance of the distributed parameters $\boldsymbol{X}_i$ is bounded by T, we need to ensure the condition shown in Eq.25. We can ignore the positive scalar $\frac{1}{k}$, as it does not alter the level set. Consequently, the protocol will monitor the sign of the summation

$$\sum_{i=1}^{k} \zeta(\boldsymbol{X}_i(t)) = \sum_{i=1}^{k} (T - ||\boldsymbol{X}_i(t) - \boldsymbol{E}||_2^2).$$

It is easy to see that the function $\zeta(x) = T - ||\boldsymbol{x} - \boldsymbol{E}||_2^2$ is concave, positive for $\boldsymbol{E}$ and $\zeta > \phi$. Function ζ is also safe for the admissible region A. Recall that a function ζ is safe for A, if and only if, $L(\zeta) \subseteq A$. To prove the sufficiency, assume $L(\zeta) \subseteq A$. Because ζ is concave we have that

$$\begin{aligned} \zeta(\boldsymbol{S}) &= \zeta(\frac{1}{k} \sum_{i=1}^{k} \boldsymbol{X}_i) \\ &= T - || \frac{1}{k} \sum_{i=1}^{k} \boldsymbol{X}_i - \boldsymbol{E}||_2^2 \\ &\leq T - \frac{1}{k} \sum_{i=1}^{k} ||\boldsymbol{X}_i - \boldsymbol{E}||_2^2 \\ &= \frac{1}{k} \sum_{i=1}^{k} \zeta(X_i). \end{aligned}$$

Then, $\sum_{i=1}^{k} \zeta(\boldsymbol{X}_i) \geq 0$ implies that $\zeta(\boldsymbol{S}) \geq 0$, and thus $\boldsymbol{S} \in L(\zeta) \subseteq A$. Moreover, in this specific case it is true that $Z = L(\zeta) = A$.

The local safe zone functions $\zeta(\boldsymbol{X}_i(t))$ are quadratic functions, which means that small variations in the input may cause large variations to the output of the function. To ameliorate this situation, we transform these functions to eikonal. An function $\zeta : V \to \mathbb{R}$ is eikonal, if and only if, $||\nabla \zeta|| = 1$ at every point where it is differentiable. Fortunately, in our case these this can be done quite easily by square rooting the two terms of our local safe zone functions $\zeta(\boldsymbol{X}_i(t))$. Thus, we are left with the following monitoring problem

$$\sum_{i=1}^{k} \zeta(\boldsymbol{X}_i(t)) \geq 0, \tag{26}$$

where

$$\zeta(\boldsymbol{X}_i(t)) = \sqrt{T} - ||\boldsymbol{X}_i(t) - \boldsymbol{E}||_2. \tag{27}$$

Observe that the level set of the safe zone functions have not been altered, and the second norm of their partial derivatives w.r.t $\boldsymbol{X}_i(t)$ is 1. What is more, a nice consequence of ζ being eikonal, is that it provides sufficient distance information from the convex subset Z of the admissible region A. More precisely, the eikonal function $\zeta(\boldsymbol{X}_i(t))$ is the signed distance function for the set Z. The maximum value of ζ is $\zeta(\boldsymbol{E})$, meaning that as soon as X_i starts to deviate from $\boldsymbol{E}$, towards any direction, the value of ζ decreases and the variance between the models increases. The convex set Z is considered to be maximal, according to Definition 11 in [15], as there is no convex subset of A that is a superset of Z. Z is also maximum distance, according to Definition 5 in [15], as $dist(\boldsymbol{E}, \bar{Z}) = dist(\boldsymbol{E}, \bar{A})$, where $\bar{\cdot} = V - \cdot$. This is obvious here as $Z = A$. For more information about composing good safe zone functions you may refer to [15].

3.3 The basic FGM protocol for learning

We now introduce our approach for distributively training ANNs, and parametric machine learning algorithms in general, in an online manner by using the basic FGM protocol without rebalancing. In the beginning, the coordinator warms up the predictor by fitting some data to it. These data could be centralized by the workers to the coordinator. After the warm-up, the coordinator ships the parameters to all the sites. This procedure provides the same starting point for all the workers. The size of the warm-up dataset can be tuned by the user, and its size can affect the communication cost of the early rounds. If the warm-up dataset is "too" small, then the communication cost of the early stages of the stream is likely to be big. Conversely, if the warm-up dataset is somewhat "large", then the communication cost is expected to be quite small, as the starting parameters may be closer to the optimal solution, thus reducing the variability of their updates. In this study, we start evaluating the algorithm right after the warm-up procedure, at the beginning of the first round, and we do not investigate for any further strategies for choosing the optimal warm-up dataset size.

The basic FGM protocol for distributed machine learning is very similar to the one mentioned in §2.5.3. In this particular case, the protocol monitors the threshold condition $\sum_{i=1}^{k} \zeta(\boldsymbol{X}_i(t)) \geq 0$, with $\zeta(\boldsymbol{X}_i(t)) = \sqrt{T} - ||\boldsymbol{X}_i(t) - \boldsymbol{E}||_2 = \sqrt{T} - ||\boldsymbol{\Delta}_i(t)||_2$, where $\boldsymbol{\Delta}_i(t)$ is the local delta vector of the parameters. This delta vector is nothing more than the summation of all the updates that were incurred by the local optimizer, or any other learning update rule, since the beginning of the round. In general, we could pick any safe zone function that we may think is good to monitor. In this example, we picked this one to extend and compare against the work of Kamp in [17] and [20]. The basic protocol for distributed training with arbitrary safe zone function and arbitrary learning algorithm with update rule of the form $w_t = w_{t-1} + step * update$, can be summarized in Alg. 6.

3.4 The averaging procedure for learning

In the "Static Averaging Protocols" like S-DSGD, the averaging procedure is quite simple. The parameters of all the workers are added up and divided by the total number of workers at the end of each round. In our approach though this may not always be the right course of action. In a stream like setting, the rate by which each site receives data may be completely different than the rest of them. In some adversarial conditions, it may the case that a site has not received any example. In an even more extreme case, only one site may receive examples. To counter this problem, the protocol divides the summation of the delta vectors, provided by all the sites, by the number of sites that have a non-zero delta vector $\boldsymbol{\Delta X}_i$. This can also be observed in Alg. 6. Observe that $\boldsymbol{\Delta X}_i = \boldsymbol{X}_i - \boldsymbol{E}$.

Algorithm 6 ML-FGM

Require: ζ, ϵ_ψ, m, $BatchSize$

Initialization at the coordinator:

Warming up the global learner and end up with parameters w_{init}
Set containing the nodes that have updated their local parameters: $U \leftarrow \emptyset$
$E \leftarrow w_{init}$, $c \leftarrow 0$, $\psi \leftarrow k\zeta(E)$, $\theta \leftarrow \frac{\psi}{2k}$
send E and θ to all sites and start the first round

A. Site i on **receiving** E and θ at the start of a new round:

update the local model: $\boldsymbol{X}_i \leftarrow E$
$quantum \leftarrow \theta$, $c_i \leftarrow 0$, $z_i \leftarrow \zeta(E)$

B. Site i on **receiving** θ at the start of a new subround:

$c_i \leftarrow 0$, $quantum \leftarrow \theta$, $z_i \leftarrow \zeta(\boldsymbol{X}_i)$

C. Site i on **observing** data at at time t:

observe $\{x_{t,i}, y_{t,i}\}$
Augment $\mathbf{batch}_i$ with the data point $\{x_{t,i}, y_{t,i}\}$
if $size(\mathbf{batch}_i) == BatchSize$ **then**
  update the local model $\boldsymbol{X}_i$ by fitting to it the $\boldsymbol{batch}_i$
  $BatchesObserved_i \leftarrow BatchesObserved_i + 1$
  clear the mini-batch $\mathbf{batch}_i$
  if $BatchesObserved_i \bmod m = 0$ **and** $\lfloor \frac{z_i - \zeta(\boldsymbol{X}_i)}{quantum} \rfloor > c_i$ **then**
    $Increment_i \leftarrow \lfloor \frac{z_i - \zeta(\boldsymbol{X}_i)}{quantum} \rfloor - c_i$
    $c_i \leftarrow \lfloor \frac{z_i - \zeta(\boldsymbol{X}_i)}{quantum} \rfloor$
    send $Increment_i$ to the coordinator

D. Coordinator on **receiving** an increment:

$c \leftarrow c + Increment_i$
if $c > k$
  request and **collect** all $\zeta(\boldsymbol{X}_i)$ from all sites
  $\psi \leftarrow \sum_{i=1}^{k} \zeta(\boldsymbol{X}_i)$
  if $\psi \leq \epsilon_\psi k\zeta(\boldsymbol{E})$
    request and **collect** all $\boldsymbol{\Delta X}_i$ from all sites
    for each $\boldsymbol{\Delta X}_i$
      if $\boldsymbol{\Delta X}_i$!= $\mathbf{0}$ **and** $i \notin U$ **then**
        augment U with the site i
    $E \leftarrow E + \frac{1}{size(U)} \sum_{i=1}^{k} \boldsymbol{\Delta X}_i$
    $U \leftarrow \emptyset$, $c \leftarrow 0$, $\psi \leftarrow k\zeta(E)$, $\theta \leftarrow \frac{\psi}{2k}$
    send E and θ to all sites and start a new round (code A)
  else
    $c \leftarrow 0$, $\theta \leftarrow \frac{\psi}{2k}$
    send θ to all sites to start a new subround (code B)

3.5 The rebalancing protocol for learning

Although the basic FGM protocol can perform much better than the GM, the gains can be further increased by utilizing rebalancing methods. In this section we extend the basic FGM protocol for machine learning with a simple rebalancing procedure.

The algorithm for rebalancing a round in the FGM protocol that is presented in §2.5.6, is a valid rebalancing method for monitoring streams that are summarized into linear sketches. In the machine learning case though, a slight modification is needed. In the vanilla rebalancing method, each site sends its drift vector $\boldsymbol{X}_i$ to the coordinator and resets it to $\boldsymbol{E}$, or equivalently it sends its delta vector $\boldsymbol{\Delta X}_i$ to the coordinator and then sets it to the zero vector. When we are working with predictors though, where $\boldsymbol{X_i}$ represents the parameters, it would not make sense to revert $\boldsymbol{X}_i$ back to $\boldsymbol{E}$, as we would discard the progress that was achieved so far and retrain the predictor from $\boldsymbol{E}$ all over again.

To overcome this obstacle, we need to take a closer look at the safe zone function at hand. For $\zeta(\boldsymbol{X}_i) = \sqrt{T} - ||\boldsymbol{X}_i - E||_2$ we have that

$$\begin{aligned}\zeta(\boldsymbol{X}_i) &= \sqrt{T} - ||\boldsymbol{X}_i - \boldsymbol{E}||_2 \\ &= \sqrt{T} - ||\boldsymbol{E} + \boldsymbol{\Delta X}_i - \boldsymbol{E}||_2 \\ &= \sqrt{T} - ||\boldsymbol{\Delta X}_i||_2.\end{aligned}$$

Suppose now that $\boldsymbol{\Delta X}_i^j$ are the updates incurred to sites' i parameter vector from the beginning of the round up until the j^{th} rebalance. Also, let $\boldsymbol{\Delta X}_i^{jt}$ be the updates incurred to sites' i parameter vector from the beginning of the j^{th} rebalance up until the current time t. Hence, at each time t it is true that $\boldsymbol{X}_i = \boldsymbol{E} + \boldsymbol{\Delta X}_i^j + \boldsymbol{\Delta X}_i^{jt}$. After rebalance j, we would ideally like to monitor only the updates that happen from that time on wards at each site, meaning that for each site we would like to measure the quantity $\sqrt{T} - ||\boldsymbol{\Delta X}_i^{jt}||_2$. By analyzing this relation get

$$\begin{aligned}\sqrt{T} - ||\boldsymbol{\Delta X}_i^{jt}||_2 &= \sqrt{T} - ||\boldsymbol{E} + \boldsymbol{\Delta X}_i^{jt} - \boldsymbol{E}||_2 \\ &= \sqrt{T} - ||\boldsymbol{E} + \boldsymbol{\Delta X}_i^{j} + \boldsymbol{\Delta X}_i^{jt} - \boldsymbol{E} - \boldsymbol{\Delta X}_i^{j}||_2 \\ &= \sqrt{T} - ||\boldsymbol{X}_i - (\boldsymbol{E} + \boldsymbol{\Delta X}_i^{j})||_2.\end{aligned}$$

Therefore, each site needs to be equipped with an extra parameter vector $E_i = \boldsymbol{E} + \boldsymbol{\Delta X}_i^j$. At the start of the round, all sites set their E_i to the new $\boldsymbol{E}$. When the coordinator needs the delta vectors to rebalance the round, then every site sends the vector $\boldsymbol{X}_i - E_i$ as its delta vector, and then sets its E_i equal to $\boldsymbol{X}_i$. With this simple trick, the sites do set their delta vectors to $\boldsymbol{0}$, without actually changing their drift vector. This stabilizes the learning process in each site.

The only thing that remains is to form the $\psi + \psi_B$ term to calculate the new quantum for $\theta = \frac{\psi + \psi_B}{2k}$. Considering our safe zone function and the fact that some remote sites may not receive any examples the term when the rebalancing happens becomes

$$\begin{aligned}\psi + \psi_B &= \frac{|B|}{2}\zeta(\boldsymbol{E} + \frac{2\boldsymbol{B}}{|B|}) + \sum_{i \in B}\frac{1}{2}\zeta(\boldsymbol{E} + 2\boldsymbol{\Delta X}_i) + \sum_{i \notin B}\zeta(\boldsymbol{X}_i) \\ &= \frac{|B|}{2}\zeta(\boldsymbol{E} + \frac{2\boldsymbol{B}}{|B|}) + \frac{1}{2}\sum_{i \in B}\zeta(\boldsymbol{E}) + (k - |B|)\sum_{i \notin B}\zeta(\boldsymbol{X}_i) \\ &= \frac{|B|}{2}\zeta(\boldsymbol{E} + \frac{2\boldsymbol{B}}{|B|}) + \frac{|B|}{2}\zeta(\boldsymbol{E}) + (k - |B|)\zeta(\boldsymbol{E}) \\ &= \frac{|B|}{2}\zeta(\boldsymbol{E} + \frac{2\boldsymbol{B}}{|B|}) + (\frac{|B|}{2} + k - |B|)\zeta(\boldsymbol{E}) \\ &= \frac{|B|}{2}(\sqrt{T} - \left\|\frac{2\boldsymbol{B}}{|B|}\right\|_2) + (\frac{|B|}{2} + k - |B|)\sqrt{T}.\end{aligned}$$

The final approach to our protocol that uses rebalancing methods for online distributed machine learning, is depicted in Alg. 7.

Algorithm 7 ML-FGM with rebalancing for $\zeta(X_i) = \sqrt{T} - ||X_i - E||_2$

Require: T, ϵ_ψ, ϵ_{reb}, m, $BatchSize$

Initialization at the coordinator:

Warming up the global learner and end up with parameters w_{init}
Set containing the nodes that have updated their local parameters: $U \leftarrow \emptyset$
$E \leftarrow w_{init}$, $c \leftarrow 0$, $\psi \leftarrow k\sqrt{T}$, $\theta \leftarrow \frac{\psi}{2k}$, $B \leftarrow \mathbf{0}$, $reb \leftarrow 0$
send E and θ to all sites and start the first round

A. Site i on **receiving** E and θ at the start of a new round:

update the local model: $\boldsymbol{X}_i \leftarrow E$
$quantum \leftarrow \theta$, $c_i \leftarrow 0$, $z_i \leftarrow \sqrt{T}$, $E_i \leftarrow E$

B. Site i on **receiving** θ at the start of a new subround:

$c_i \leftarrow 0$, $quantum \leftarrow \theta$, $z_i \leftarrow \sqrt{T} - ||\boldsymbol{X}_i - E_i||_2$

C. Site i on **receiving** θ for rebalancing the current round:

$c_i \leftarrow 0$, $quantum \leftarrow \theta$, $E_i \leftarrow \boldsymbol{X}_i$, $z_i \leftarrow \sqrt{T}$

D. Site i on **sending** $\boldsymbol{\Delta X}_i$ to the coordinator to rebalance the current round:

$\boldsymbol{\Delta X}_i \leftarrow \boldsymbol{X}_i - E_i$
send $\boldsymbol{\Delta X}_i$ to the coordinator

E. Site i on **observing** data at at time t:

observe $\{x_{t,i}, y_{t,i}\}$
Augment $\mathbf{batch}_i$ with the data point $\{x_{t,i}, y_{t,i}\}$
if $size(\mathbf{batch}_i) == BatchSize$ **then**
  update the local model $\boldsymbol{X}_i$ by fitting to it the $\boldsymbol{batch}_i$
  $BatchesObserved_i \leftarrow BatchesObserved_i + 1$
  clear the mini-batch $\mathbf{batch}_i$
  if $BatchesObserved_i \bmod m = 0$ **and** $\lfloor \frac{z_i - \sqrt{T} + ||\boldsymbol{X}_i - E_i||_2}{quantum} \rfloor > c_i$ **then**
    $Increment_i \leftarrow \lfloor \frac{z_i - \sqrt{T} + ||\boldsymbol{X}_i - E_i||_2}{quantum} \rfloor - c_i$
    $c_i \leftarrow \lfloor \frac{z_i - \sqrt{T} + ||\boldsymbol{X}_i - E_i||_2}{quantum} \rfloor$
    send $Increment_i$ to the coordinator

F. Coordinator on **receiving** an increment:

$c \leftarrow c + Increment_i$
if $c > k$
  request and **collect** $\zeta_i = \sqrt{T} - ||\boldsymbol{X}_i - E_i||_2 \forall i \notin U$ and $\zeta_i = \frac{1}{2}(\sqrt{T} - ||2(\boldsymbol{X}_i - E_i)||_2) \forall i \in U$
  $\psi \leftarrow \sum_{i=1}^{k} \zeta_i$
  if $reb == 1$
    $\psi \leftarrow \psi + \frac{size(U)}{2}\left(\sqrt{T} - ||\frac{2B}{size(U)}||_2\right)$
  if $\psi \leq \epsilon_\psi k\sqrt{T}$
    request and **collect** all $\boldsymbol{\Delta X}_i$ from all sites (code D)
    for each $\boldsymbol{\Delta X}_i$
      $B \leftarrow B + \boldsymbol{\Delta X}_i$
      if $\boldsymbol{\Delta X}_i$!= $\mathbf{0}$ **and** $i \notin U$ **then**
        augment U with the site i
    $\psi \leftarrow \frac{size(U)}{2}\left(\sqrt{T} - ||\frac{2B}{size(U)}||_2\right) + \left(\frac{size(U)}{2} + k - size(U)\right)\sqrt{T}$
    if $\psi \geq \epsilon_{reb}\, \epsilon_\psi\, k\sqrt{T}$ **then**
      $c \leftarrow 0$, $\theta \leftarrow \frac{\psi}{2k}$, $reb \leftarrow 1$
      send θ to all sites in order to rebalance the current round (code C)
    else
      $E \leftarrow E + \frac{1}{size(U)}B$
      $B \leftarrow \mathbf{0}$, $U \leftarrow \emptyset$, $c \leftarrow 0$, $\psi \leftarrow k\sqrt{T}$, $\theta \leftarrow \frac{\psi}{2k}$, $reb \leftarrow 0$
      send E and θ to all sites and start a new round (code A)
  else
    $c \leftarrow 0$, $\theta \leftarrow \frac{\psi}{2k}$
    send θ to all sites to start a new subround (code B)

Algorithm 8 AOS-ELM Synchronization Module

Initialization at the coordinator:

$\boldsymbol{A} \leftarrow RandomNumbers([-1, 1], \mathbb{R}^{d \times L})$
$\boldsymbol{b} \leftarrow RandomNumbers([-1, 1], \mathbb{R}^{L})$
send $\boldsymbol{A}$ and $\boldsymbol{b}$ to all sites (code A)

A. Site i on **receiving** $\boldsymbol{A}$ and $\boldsymbol{b}$ at the before the start of the first round:

update the local model: $\boldsymbol{A}_i \leftarrow \boldsymbol{A}$
update the local model: $\boldsymbol{b}_i \leftarrow \boldsymbol{b}$

B. Site i on **observing** data data with concept drift at time t:

observe $\{x_{t,i}, y_{t,i}\}$
if $NumOfAttr(x_{t,i}) > NumOfRows(\boldsymbol{A}_i)$ **and** $SizeOfClasses(y_{t,i}) > NumOfCols(\boldsymbol{\beta}_i)$
  $d1 \leftarrow NumOfAttr(x_{t,i})$, $d2 \leftarrow SizeOfClasses(y_{t,i})$
  send to the coordinator the increment to the attributes and classes $d1$, $d2$ (code D)
  wait for the coordinator to send the new parameters $\Delta\boldsymbol{A}_{d1 \times L}$ in one or more messages
  update the model when the coordinator responds: $\boldsymbol{A}_i \leftarrow \begin{bmatrix} \boldsymbol{A}_i \\ \Delta\boldsymbol{A}_{d1 \times L} \end{bmatrix}$, $\boldsymbol{\beta}_i \leftarrow [\boldsymbol{\beta}_i \quad \boldsymbol{0}_{L \times d2}]$
else if $NumOfAttr(x_{t,i}) > NumOfRows(\boldsymbol{A}_i)$
  $d \leftarrow NumOfAttr(x_{t,i})$
  send to the coordinator the increment to the attributes d (code D)
  wait for the coordinator to send the new parameters $\Delta\boldsymbol{A}_{d \times L}$ in one or more messages
  update the model when the coordinator responds: $\boldsymbol{A}_i \leftarrow \begin{bmatrix} \boldsymbol{A}_i \\ \Delta\boldsymbol{A}_{d \times L} \end{bmatrix}$
else if $SizeOfClasses(y_{t,i}) > NumOfCols(\boldsymbol{\beta}_i)$
  $d \leftarrow SizeOfClasses(y_{t,i})$
  send to the coordinator the increment to classes d (code D)
  wait for the coordinator for confirmation
  update the model when the coordinator responds: $\boldsymbol{\beta}_i \leftarrow [\boldsymbol{\beta}_i \quad \boldsymbol{0}_{L \times d}]$
resuming normal execution

C. Site i on **receiving** new attributes and classes increment the coordinator:

receive $\{\Delta\boldsymbol{A}_{d1 \times L}, d2\}$
if $\Delta\boldsymbol{A}_{d1 \times L}$ **not** *empty*
  $\boldsymbol{A}_i \leftarrow \begin{bmatrix} \boldsymbol{A}_i \\ \Delta\boldsymbol{A}_{d1 \times L} \end{bmatrix}$
if $d2 > 0$
  $\boldsymbol{\beta}_i \leftarrow [\boldsymbol{\beta}_i \quad \boldsymbol{0}_{L \times d2}]$

D. Coordinator on **receiving** attribute and classes increments by site i:

receive $\{d1, d2\}$
if $d1 > NumOfRows(\boldsymbol{A})$ **and** $d2 > NumOfCols(\boldsymbol{\beta})$
  $d1 \leftarrow d1 - NumOfRows(\boldsymbol{A})$, $d2 \leftarrow d2 - NumOfCols(\boldsymbol{\beta})$
  $\Delta\boldsymbol{A}_{d1 \times L} \leftarrow RandomNumbers([-1, 1], \mathbb{R}^{d1 \times L})$
  $\boldsymbol{A} \leftarrow \begin{bmatrix} \boldsymbol{A} \\ \Delta\boldsymbol{A}_{d1 \times L} \end{bmatrix}$, $\boldsymbol{\beta} \leftarrow [\boldsymbol{\beta} \quad \boldsymbol{0}_{L \times d2}]$
  send to all sites $j \neq i$ the tuple $\{\Delta\boldsymbol{A}_{d1 \times L}, d2\}$ to update their models (code C)
  send a response to site i with the tuple $\{\Delta\boldsymbol{A}_{d1 \times L}, d2\}$ to update its model
else if $d1 > NumOfRows(\boldsymbol{A})$
  $d1 \leftarrow d1 - NumOfRows(\boldsymbol{A})$
  $\Delta\boldsymbol{A}_{d1 \times L} \leftarrow RandomNumbers([-1, 1], \mathbb{R}^{d1 \times L})$
  $\boldsymbol{A} \leftarrow \begin{bmatrix} \boldsymbol{A} \\ \Delta\boldsymbol{A}_{d1 \times L} \end{bmatrix}$
  send to all sites $j \neq i$ the new weights $\Delta\boldsymbol{A}_{d1 \times L}$ to update their models (code C)
  send a response to site i with the new weights $\Delta\boldsymbol{A}_{d1 \times L}$ to update its model
else if $d2 > NumOfCols(\boldsymbol{\beta})$
  $d2 \leftarrow d2 - NumOfCols(\boldsymbol{\beta})$
  $\boldsymbol{\beta} \leftarrow [\boldsymbol{\beta} \quad \boldsymbol{0}_{L \times d2}]$
  send to all sites the increment $d2$ to update their models (code C)
  send a response to site i with the increment $d2$ to update its model

3.6 The applicability of FGM in machine learning

The proposed framework is suited for any parametric machine learning algorithm that has an update rule of some kind. Our approach provides the ability to incorporate any of the aforementioned optimizers in §2.2.1, as well as more general update rules like 21 and 22. However, we need to slightly modify the distributed algorithms in order to be able to account for the internal changes of AOS-ELM that happen during the detection of CD. DMS and ML-FGM stay exactly the same, with the addition of the Alg. 8. We assume, for simplicity, that after a single point with CD has been observed, all the next points respect this change. Finally, the parameters that need to be tracked are the matrix $\boldsymbol{K}$ and the vector $\boldsymbol{\beta}$. Hence, the algorithm tracks the vector $\hat{\boldsymbol{W}} \in \mathbb{R}^{L^2+L|C|}$ with $\hat{\boldsymbol{W}} = \hat{\boldsymbol{K}} \oplus \hat{\boldsymbol{\beta}}$, where $\hat{\boldsymbol{K}}$ and $\hat{\boldsymbol{\beta}}$ are the flattened vectors of matrix $\boldsymbol{K}$ and vector $\boldsymbol{\beta}$ respectively.

4 Experimental Results

In this section we investigate the practical performance of distributed machine learning via the FGM protocol against the state-of-the-art dynamic learning protocols. The DMS algorithm has been experimentally proven to be much more communication efficient than any static averaging protocol like S-DSGD and A-DSGD. Thus, our main goal is to empirically confirm the communication gains against DMS. Our experiments will be conducted on two distinct online learning scenarios, a distributed stream with no concept drift where we will train a CNN classifier, and a distributed stream with concept drift where we will train an AOS-ELM classifier.

4.1 CNN training on static distributed streams

We start with the problem of tracking a static and rapid distributed stream, for the purpose of classifying the data using a CNN. The dataset that we used for this experiment is the MNIST data set. MNIST data set is a balanced data set that contains numeric handwriting (digits from 0 to 9) with size 28×28 pixels in a gray scale image. The data set has been divided to 60000 examples for training, and 10000 examples for testing the accuracy of the classifier. However, the regular MNIST dataset is not big enough to simulate a big data stream. For that reason, we developed an extended version of the MNIST, with larger number of examples, by using a very common technique in deep learning called data augmentation.

It is generally the case, especially in biomedical problems, that the size of the data set is not big enough for a predictor to learn any valuable patterns or underlined information in the data. To overcome this problem, a widely used technique on image data sets is Data Augmentation (DA), where you randomly apply transformation on the images to come up with a much bigger number of training examples. A CNN can robustly classify objects even if they are placed in different orientations, and for this reason is said to have the property called *invariance*. More specifically, a CNN can be invariant to translation, rotation, illumination, image noise or distortion, size and viewpoint or even a combination of the above. Utilizing this technique, we augmented the MNIST data set, calling it AMNIST, by using random distortion, image cropping, shear and image skew in any random combination. The resulting training set finally has 2220000 training examples. For the testing we use the regular test data set of MNIST.

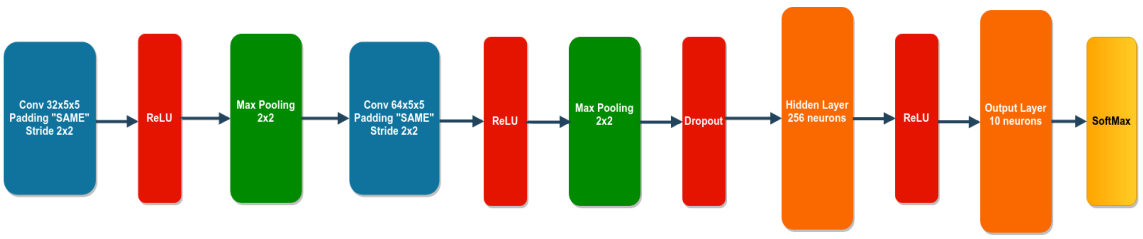

Figure 1: CNN architecture.

For the CNN we used a modified LeNet architecture with two convolutional layers and one hidden layer. The first convolutional layer is consisted of 32 5×5 filters with padding "SAME" and stride 2×2. The outputs are passed through ReLU activation functions and then straight to a max pooling layer. The second convolutional layer has the same architecture, with the difference that it uses 64 5×5 filters. The hidden layer has 256 neurons with ReLU activation functions and a dropout layer. Finally, the output layer has 10 outputs, one of each class, with a softmax output function. The architecture of the CNN used in our experiments can be seen on figure 1.

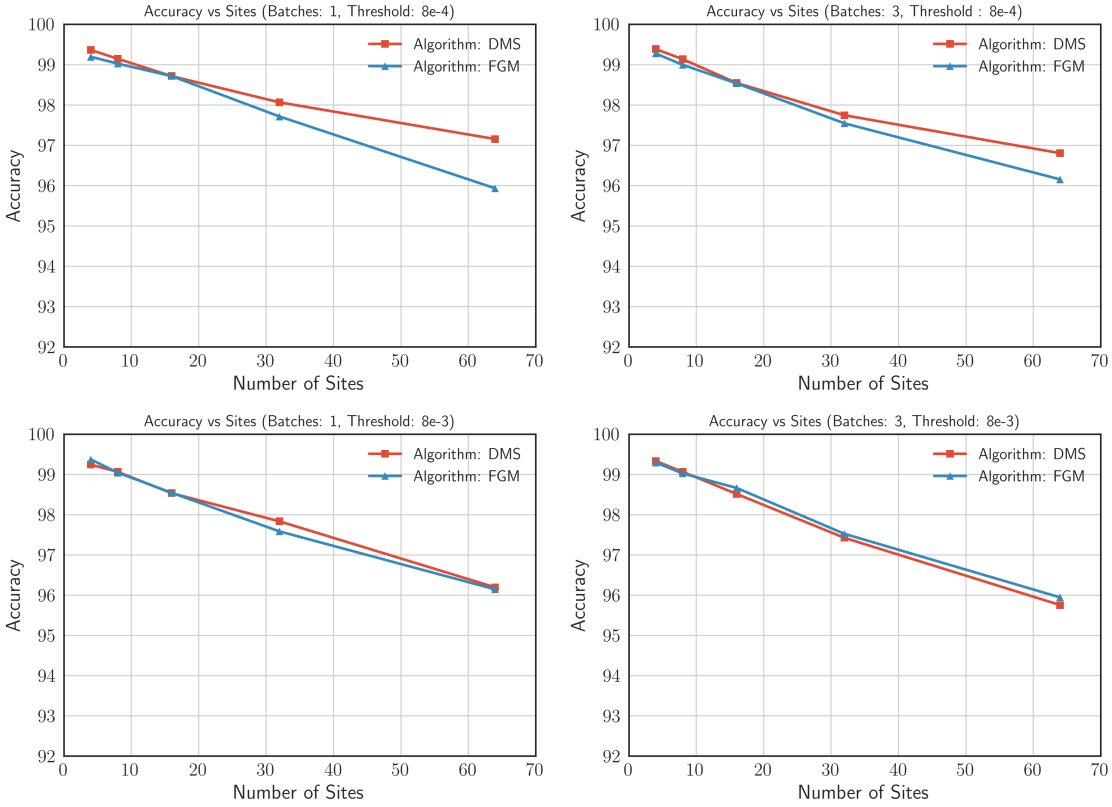

Figure 2: CNN accuracy and Sites for various DMS and ML-FGM setups.

We run experiments for thresholds (T input) $8e-4$, $8e-3$, $8e-2$, for 1, 2 and 3 observed batches per site (b input, which is the number of batches that must be observed by each node before it checks for a violation) and for 4, 8, 16, 32 and 64 remote sites (k input). Hence, we run a total of 45 experiments per dynamic averaging protocol. All mini-batch sizes were chosen to have a size of 64 data points. The learnable parameters were send within the network as there where, without any compression. Of course, compressing the parameters would reduce the total communication of the protocols, but we do not delve to compression or encryption methods in this study. We also assume that all site sampling rates are the same. We simulate this scenario by randomly assigning each mini-batch of the whole data set to a single site.

Fig 2. shows the predictive accuracy of the global CNN against the number of sites for various DMS and ML-FGM setups. Fig 3 presents the total communication of DMS and ML-FGM against the number of sites for the respective setups, and Fig.4 depicts the total communication of the distributed topologies for various variance thresholds of the same experiments. We observe that the predictive accuracy of the global CNN is approximately the same for the two distributed protocols. However, this accuracy is attained by the proposed communication protocol with substantially less communication. The communication gain of ML-FGM against DMS increases with the number of distributed sites (as shown in Fig.3), achieving a reduction of over 56% in communication for 4 distributed sites, and reaching up to 89% for 64 distributed sites. The mean communication reduction of all the experiments is close to 77%. It is evident from figures 3 and 4, that ML-FGM's communication scales much better than DMS's, as the number of remote nodes increases

and the threshold for the monitored variance of the distributed CNNs decreases. Moreover, these remarkable improvements in communication efficiency come at almost no cost. More specifically, the CNN's classification accuracy trained by ML-FGM is only worse by 0.106%. Lastly, we can see on Fig. 5 the cummulative communication incurred per fitted batch in the network for 32 remote sites, along with the total number of rounds per number of sites, by the two dynamic averaging protocols with $b = 1$ observed batches and threshold $T = 8e - 4$. It is evident, that the proposed method's communication scales much better than DMS's, as it performs fewer synchronization rounds.

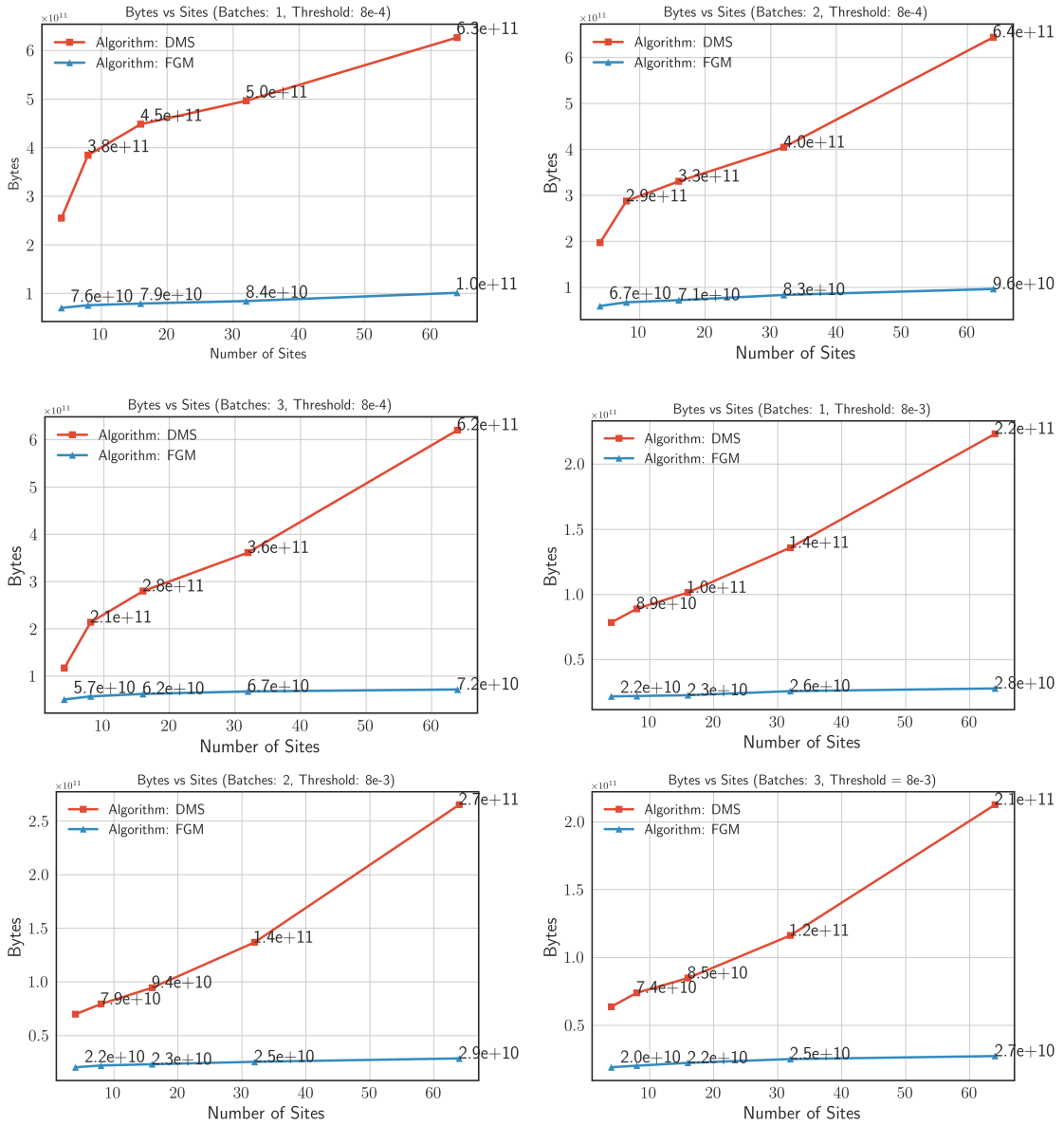

Figure 3: Total communication and Number of Sites for various DMS and ML-FGM setups with a CNN learner.

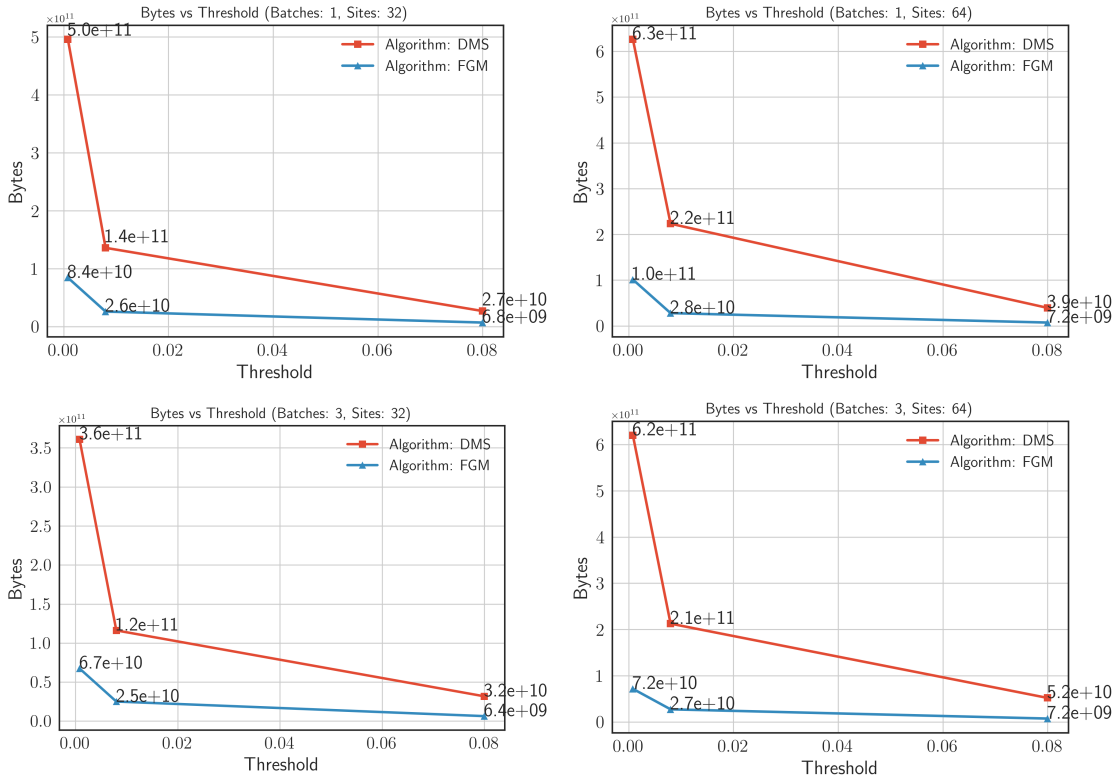

Figure 4: Total communication and Thresholds for various DMS and ML-FGM setups with a CNN learner.

For our second experiment, we tried to simulate an ill-posed distributed setting scenario by assuming different sampling rates for each remote site. In a realistic scenario, it is probable that the data rates of local streams are significantly different. For that reason, we conducted the same experiments as before, with the difference that 90% of the data are observed by just the 25% of the nodes. The results are depicted in figures 6 and 7. As it turns out, the communication gain of ML-FGM under these circumstances is even greater, achieving a reduction in communication ranging from 63% for 4 sites to 95% for 64 sites. The difference in terms of classification accuracy is a little bit bigger, with ML-FGM being worse by just 0.466%. Tables 1, 2, 3 and 4, at the Appendix section, provide more thorough and detailed information for all the aforementioned experiments.

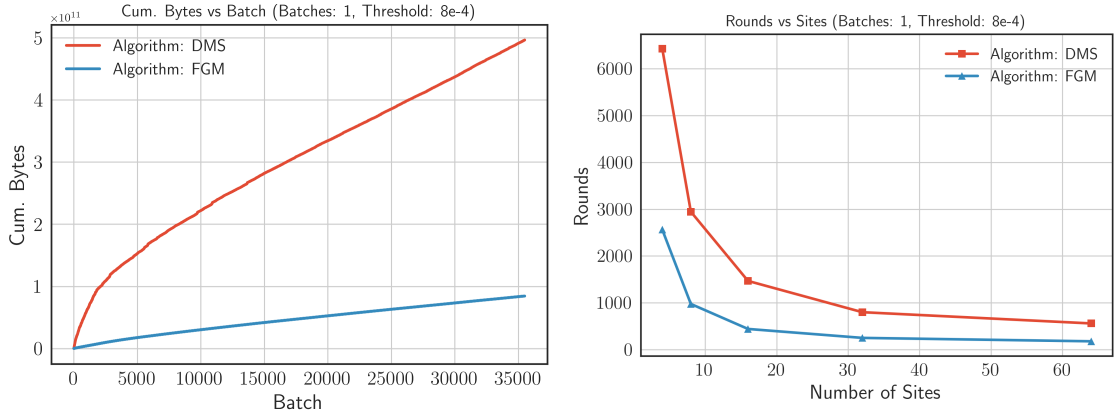

Figure 5: Cumm. communication/batch and number of rounds/sites with a CNN learner, with 32 remote sites for the left figure and $b = 1$ observed batches per site and threshold $8e - 4$ for both.

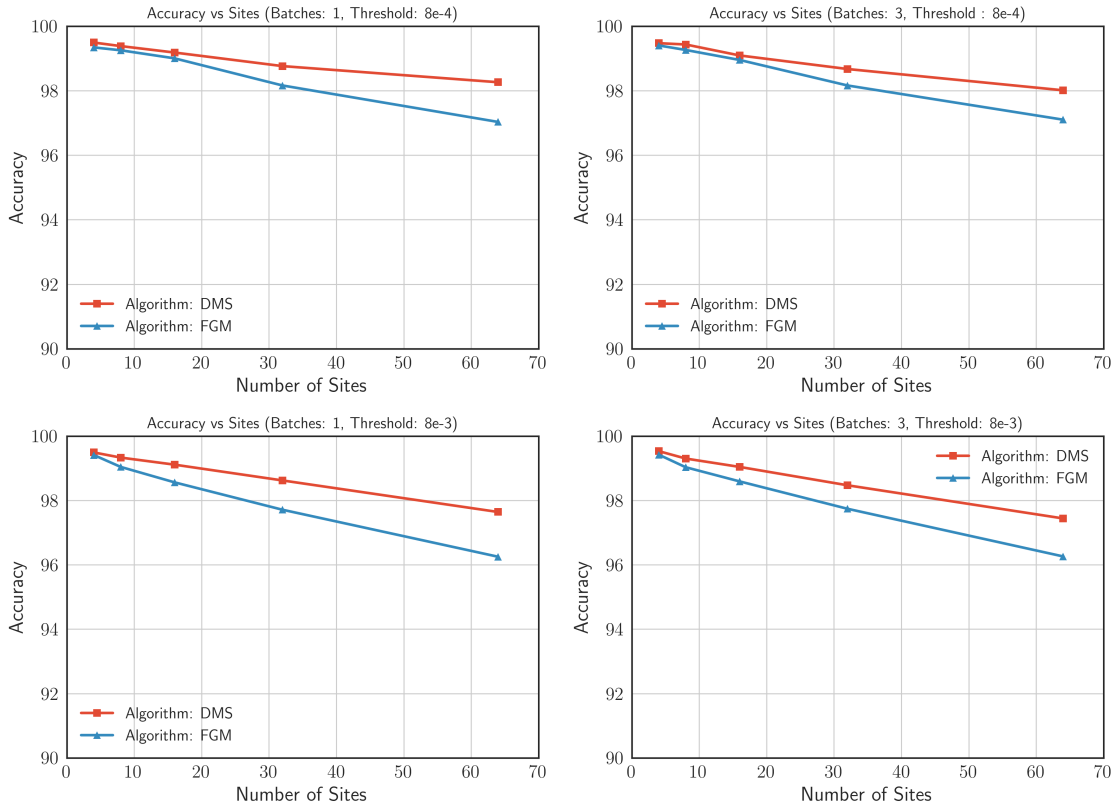

Figure 6: Total communication and Number of Sites for various DMS and ML-FGM setups with a CNN learner and different site sampling rates.

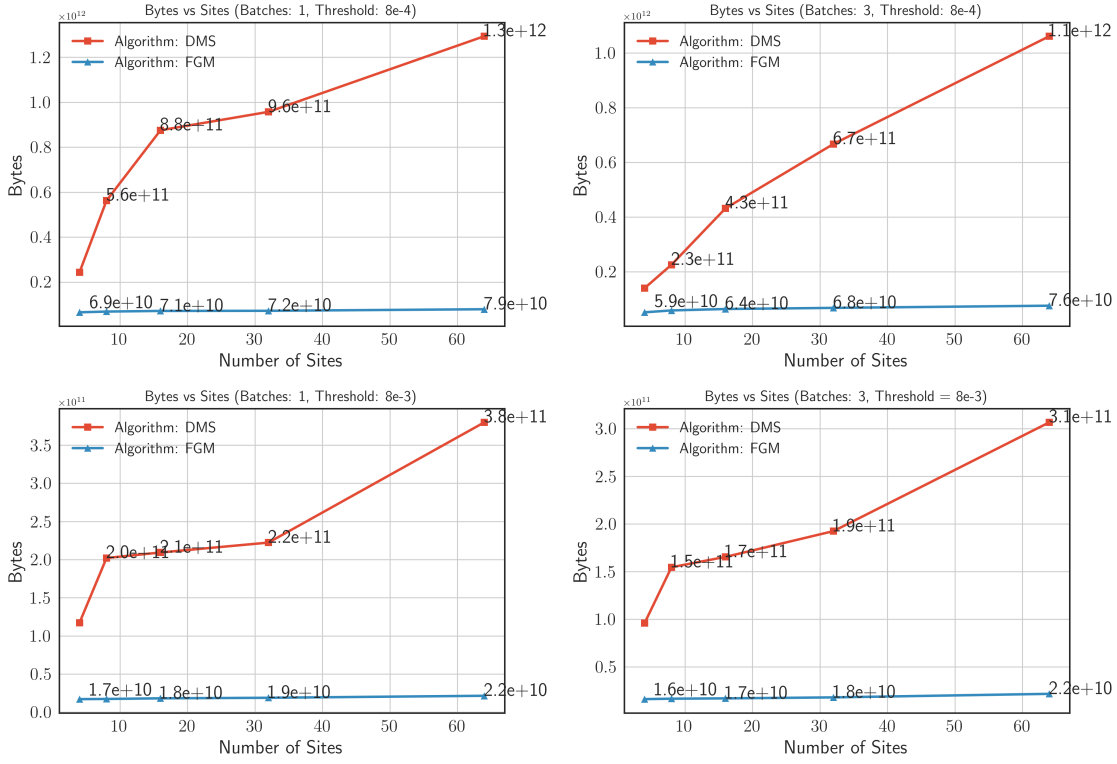

Figure 7: Total communication and Thresholds for various DMS and ML-FGM setups for a CNN learner and different site sampling rates.

4.2 AOS-ELM training on distributed streams with concept drift

We follow up with the problem of tracking a rapidly drifting distributed stream, for the purpose of classifying the data using an AOS-ELM classifier. The data set that we used for this experiment is once again the MNIST data set. In order to simulate CD, we build six separate data sets C1, C2, C1C2, C3, C4 and C5. All six data sets are augmented MNIST data sets with 1000000 training examples and 20000 testing examples each. The C1 data set consist the digits from 0 through 5, C2 consists the rest (6 through 9) and C1C2 consists all of them. The remaining three data sets have the analogous digit distribution, but the features are no longer 784, but rather 928. Each example has its original 784 pixel/features and also consists additional attributes from the 9×9 bins histogram of oriented gradients (HOG) of grey-level image features [33]. Thus we define the following six different stream concepts:

- $\boldsymbol{C_1}$ is MNIST[X_{grey}] class (0-5)
- $\boldsymbol{C_2}$ is MNIST[X_{grey}] class (6-9)
- $\boldsymbol{C_1 C_2}$ is MNIST[X_{grey}] class (0-9)
- $\boldsymbol{C_3}$ is MNIST[$X_{grey} X_{HOG}$] class (0-5)
- $\boldsymbol{C_4}$ is MNIST[$X_{grey} X_{HOG}$] class (6-9)
- $\boldsymbol{C_5}$ is MNIST[$X_{grey} X_{HOG}$] class (0-9)

With this concepts in our possession, we now define the following four CD experiments:

- Experiment 1 - VD
  $\boldsymbol{C_1 C_2} \underset{VD}{\Longrightarrow} \boldsymbol{C_5}$
- Experiment 2 - RD (recurring context)
  $\boldsymbol{C_1} \underset{RD}{\Longrightarrow} \boldsymbol{C_1 C_2}$
- Experiment 3 - RD (sudden drift)
  $\boldsymbol{C_1} \underset{RD}{\Longrightarrow} \boldsymbol{C_2}$
- Experiment 4 - HD
  $\boldsymbol{C_1} \underset{HD}{\Longrightarrow} \boldsymbol{C_5}$

The hidden layer of AOS-ELM has 256 neurons and uses a hyperbolic tangent activation function. We run the four aforementioned CD scenarios for threshold $T = 9e + 7$, for 1, 2 and 3 observed batches per site (b input) and for 4, 8, 16, 32 and 64 remote sites (k input). Hence, we run a total of 60 experiments per dynamic averaging protocol. All mini-batch sizes were chosen to have a size of 64 data points. The results are depicted on figures 9 through 10.

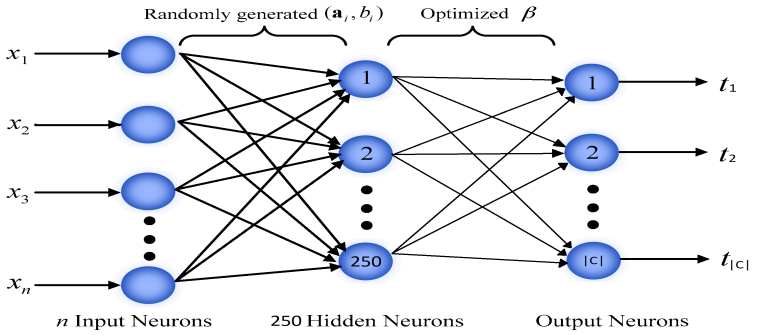

Figure 8: CNN architecture.

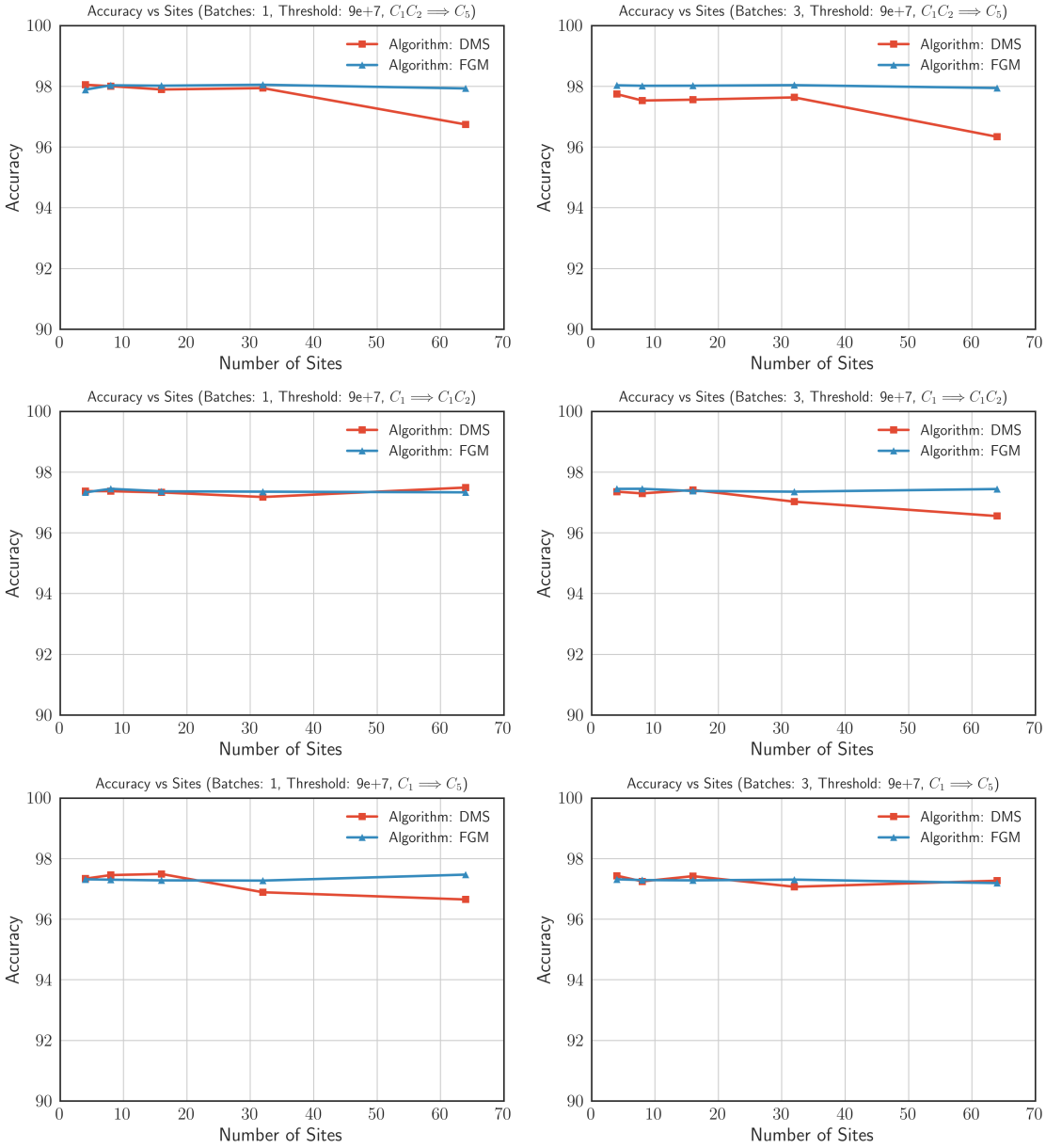

Figure 9: AOS-ELM accuracy and sites for various DMS and ML-FGM setups and CD scenarios.

ML-FGM achieves a huge reduction in network communication against DMS. Our proposed protocol attains a 41% reduction in network communication for 4 sites, and up to 97% for 64 sites. In our surprise, this reduction in communication comes without any cost in classification accuracy. On the contrary, the classification accuracy of the AOS-ELM, trained by the ML-FGM averaging protocol, has been increased by 0.235%. Lastly, we plot for a single experiment, with CD $\boldsymbol{C_1} \underset{RD}{\Longrightarrow} \boldsymbol{C_2}$, 32 remote sites, threshold $T = 9e + 7$ and $b = 1$, the cumulative communication incurred to the network by each fitted batch on Fig.11, along with the total number of rounds per number of remote sites for the same settings, and the evolution of accuracy per fitted batch on Fig.12 for each of the two protocols. We observe, yet again, that the communication of the ML-FGM protocol scales much better than DMS's. On Fig.12 we can perceive that the accuracy of the AOS-ELM classifier trained by ML-FGM rises faster than the one trained by DMS, even after the concept drift that is happening on batch 16000. We believe that this is due to the nature of the AOS-ELM training. The parameters are trained by online least squares fit, and the sparser averaging of AOS-ELM yields better accuracy, as the distributed classifiers are trained on more data before each synchronization, yielding distributed parameters that are more similar to each other, thus computing better averages during the end of each ML-FGM round.

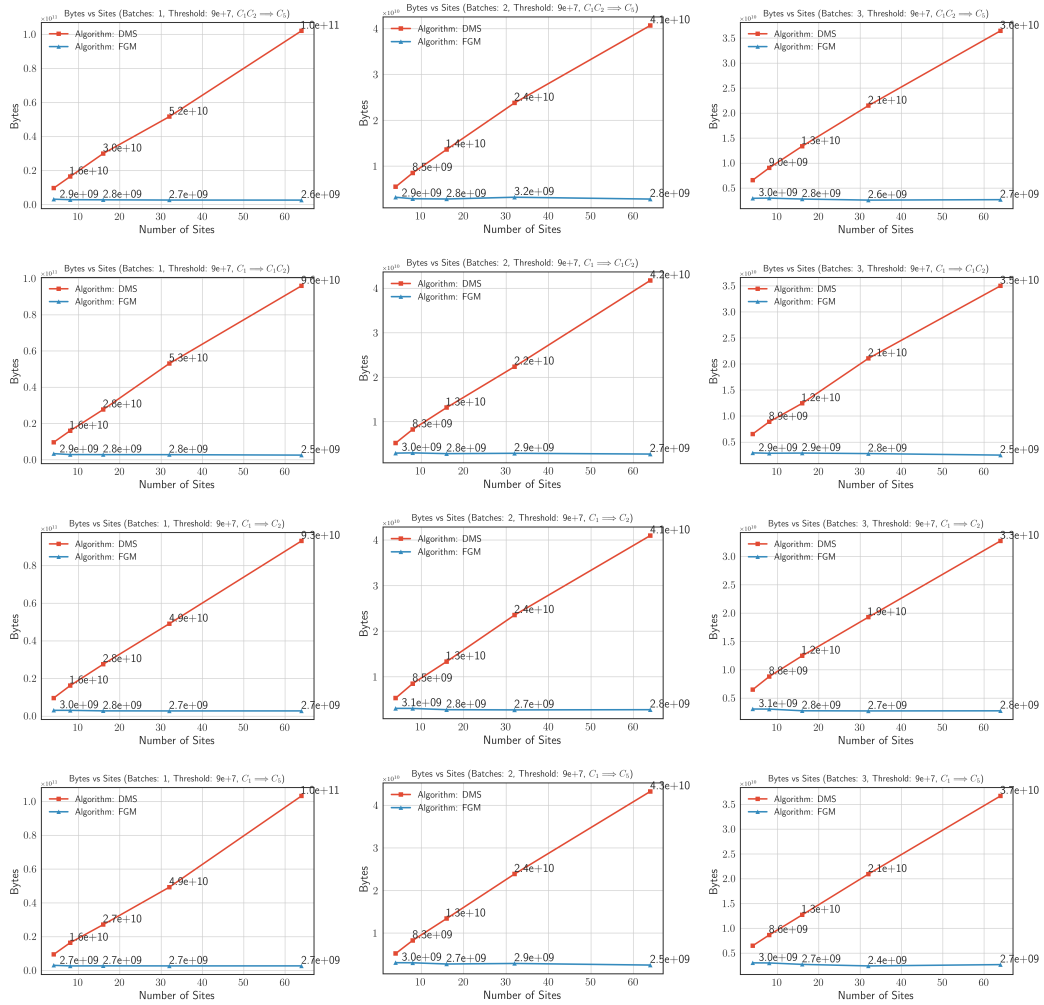

Figure 10: Total communication and number of sites for various DMS and ML-FGM setups and CD scenarios, with an AOS-ELM classifier.

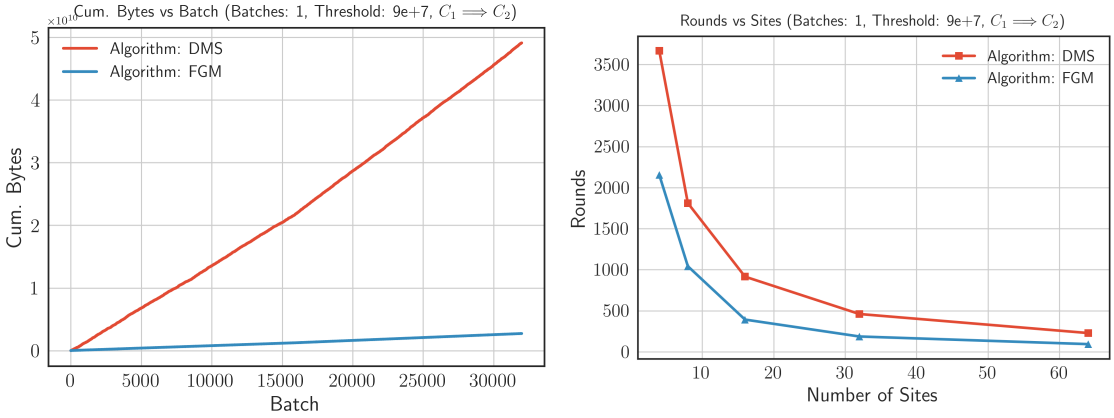

Figure 11: Cumulative communication per batch and rounds/batch with an AOS-ELM classifier, 32 remote sites for the left figure, and b=1 observed batches per site and threshold $9e+7$. The concept drift happens in batch 16000.

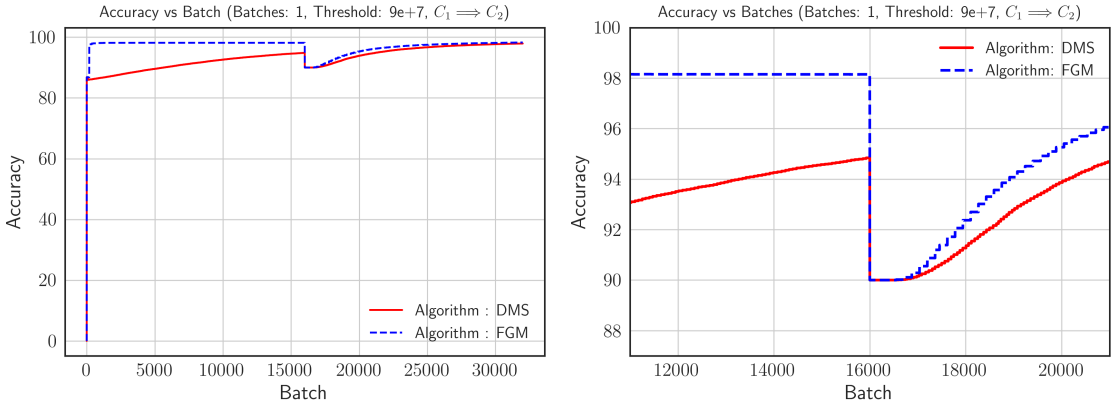

Figure 12: Accuracy per batch for DMS and ML-FGM with an AOS-ELM classifier, 32 remote sites, b=1 observed batches per site and threshold $9e+7$. The concept drift happens in batch 16000.

Once again, for our second experiment, we tried to simulate an ill-posed distributed setting scenario by assuming different sampling rates for each remote site. We conducted the experiment 4 (with CD $\boldsymbol{C_1} \underset{HD}{\Longrightarrow} \boldsymbol{C_5}$), again with the difference that 90% of the data are observed by just the 25% of the nodes. The results are depicted in figures 13 through 14. It turns out that the communication gain of ML-FGM under these circumstances is once again even greater, achieving a reduction in communication ranging from 68% for 4 sites to 98% for 64 sites, as opposed to 41% to 97% communication reduction for the uniform case. In terms of classification accuracy the AOS-ELM classifier trained by ML-FGM performs worse by just 0.259%.

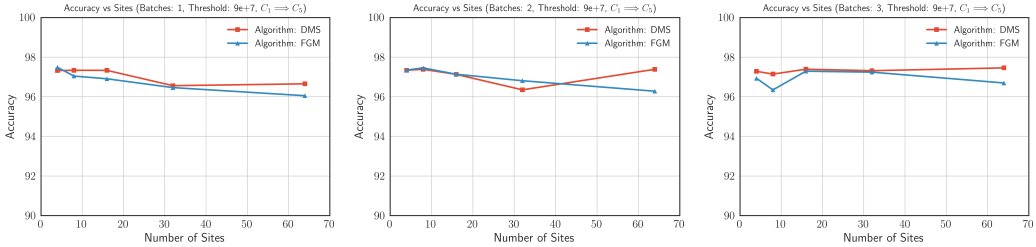

Figure 13: AOS-ELM accuracy and sites for various DMS and ML-FGM setups scenarios and different site sampling rates and hybrid drift.

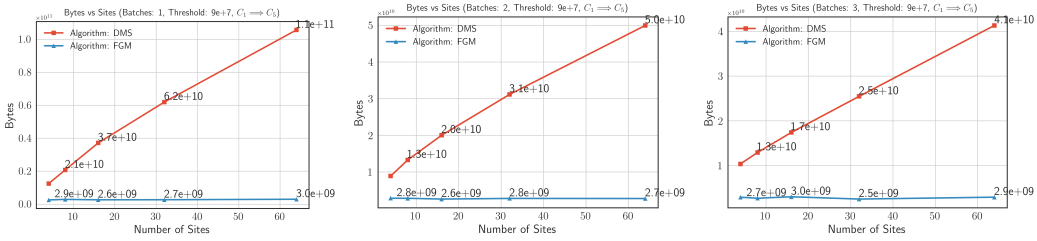

Figure 14: Total communication and number of sites for various DMS and ML-FGM setups, with an AOS-ELM classifier and different site sampling rates and hybrid drift.

5 Conclusion and Future Work

Static averaging protocols are the de facto methods in decentralized deep learning. Although these algorithms tend to work quite well, none of them take into account the communication they incur to the distributed topologies. Our proposed dynamic averaging protocol, ML-FGM, achieves high predictive performance, yet requires substantially less communication than any other contemporary static or dynamic averaging protocol. At the same time, it is applicable to traditional (static) learning scenarios, but also to non-static ones with concept drift. Moreover, the method treats the underlying learning algorithm and the optimizer as black-boxes.

Although ML-FGM was experimentally proven to be successful, in terms of predictive accuracy and network communication, it still has plenty of room for improvement. The core idea of all the dynamic averaging protocols, like DMS and ML-FGM, is to monitor a non-linear query/function on a distributed input. For example, in the machine learning problem we monitored the variance of the distributed learnable parameters in order to measure the divergence of the distributed learners. In reality, FGM provides us with the possibility of monitoring any non-linear function on the distributed parameters, and the performance of ML-FGM depends vastly on the quality of the safe zone function. Thus, a very interesting study could be to come up with a "good" safe zone, by combining information and learning theory with methods for composing safe zones in [15].

Another study that could potentially improve the efficiency of ML-FGM, is the conception of a good rebalancing strategy. Rebalancing could greatly improve the performance of the ML-FGM, especially under cases of high stream variability where the value of the monitored query changes rapidly, a case not so rare in real-world applications.

Finally, message compression would obviously reduce the number of bytes that are transferred through the network, but it needs to be handled with care as the compression of gradient updates could potentially degrade the learning procedure. Ergo, the integration of a smart compression of the gradients would be an interesting and much needed addition to the ML-FGM protocol.

References

[1] 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 Advances in Neural Information Processing Systems, pages 1223–1231, 2012.

[2] Suyog Gupta, Wei Zhang, and Josh Milthrope. Model accuracy and runtime tradeoff in distributed deep learning. arXiv preprint arXiv:1509.04210, 2015.

[3] Kai Chen and Qiang Huo. Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering. In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 5880–5884. IEEE, 2016.

[4] Hang Su and Haoyu Chen. Experiments on parallel training of deep neural network using model averaging. arXiv preprint arXiv:1507.01239, 2015.

[5] Augustus Odena. Faster asynchronous sgd. arXiv preprint arXiv:1601.04033, 2016.

[6] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. Staleness-aware async-sgd for distributed deep learning. IJCAI, 2016.

[7] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 26, pages 1223–1231. Curran Associates, Inc., 2013.

[8] Tal Ben-Nun, Torsten Hoefler, Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CoRR, abs/1802.09941, 2018, http://arxiv.org/abs/1802.09941, August 2018

[9] I. Sharfman, A. Schuster, and D. Keren. "A geometric approach to monitoring threshold functions over distributed data streams". In SIGMOD, 2006.

[10] I. Sharfman, A. Schuster, and D. Keren. Aggregate threshold queries in sensor networks. In 21th International Parallel and Distributed Processing Symposium(IPDPS 2007), Proceedings, 26-30 March 2007, Long Beach, California, USA, pages 1-10, 2007.

[11] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. In SODA, 2008.

[12] Minos N. Garofalakis, Daniel Keren, and Vasilis Samoladas, 2013. Sketch-based Geometric Monitoring on Distributed Stream Queries. PVLDB (2013).

[13] Keren, D., Sharfman, I., Schuster, A., Livne, A.: Shape sensitive geometric monitoring. IEEE Transactions on Knowledge and Data Engineering 24(8), 1520–1535 (2012)

[14] Lazerson, A., Sharfman, I., Keren, D., Schuster, A., Garofalakis, M., Samoladas, V.: Monitoring distributed streams using convex decompositions. Proceedings of the VLDB Endowment 8(5), 545–556 (2015)

[15] Minos Garofalakis and Vasilis Samoladas. 2017. Distributed Query Monitoring through Convex Analysis: Towards Composable Safe Zones. In ICDT '17.

[16] Vasilis Samoladas, Minos Garofalakis: Functional Geometric Monitoring for Distributed Streams. 22nd International Conference on Extending Database Technology (EDBT), EDBT2019.

[17] Kamp, M., Boley, M., Keren, D., Schuster, A., Sharfman, I.: Communication- efficient distributed online prediction by dynamic model synchronization. In: Machine Learning and Knowledge Discovery in Databases, pp. 623–639.Springer (2014)

[18] Gabel, M., Keren, D., Schuster, A.: Communication-efficient distributed variance monitoring and outlier detection for multivariate time series. In: Proceedings of the 28th International Parallel and Distributed Processing Symposium. pp. 37–47. IEEE (2014)

[19] Boley, M., Kamp, M., Keren, D., Schuster, A., Sharfman, I.: Communication-efficient distributed online prediction using dynamic model synchronizations. In: BD3@ VLDB. pp. 13–18 (2013)

[20] Michael Kamp, Linara Adilova, Joachim Sicking, Fabian Hüger, Peter Schlicht, Tim Wirtz, Stefan Wrobel. Efficient Decentralized Deep Learning by Dynamic Model Averaging. http://arxiv.org/abs/1807.03210, August 2018

[21] J. A. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Computing Surveys, vol. 46, no. 4, article 44, pp. 1–37, 2014.

[22] Crammer, Koby and Dekel, Ofer and Keshet, Joseph and Shalev-Shwartz, Shai and Singer, Yoram. Online Passive-Aggressive Algorithms. J. Mach. Learn. Res Vol. 9. http://dl.acm.org/citation.cfm?id=1248547.1248566. December 2006

[23] J. a. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. ACM Comput. Surv., 46(4):44:1–44:37, Mar. 2014.

[24] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," Neurocomputing, vol. 70, no. 1–3, pp. 489–501, 2006.

[25] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 42, no. 2, pp. 513–529, 2012.

[26] G.-B. Huang, "An insight into extreme learning machines: random neurons, random features and kernels," Cognitive Computation, vol. 6, no. 3, pp. 376–390, 2014.

[27] G. Huang, G.-B. Huang, S. Song, and K. You, "Trends in extreme learning machines: a review," Neural Networks, vol. 61, pp. 32–48, 2015.

[28] Y. Jun and M.-J. Er. An enhanced online sequential extreme learning machine algorithm. In Control and Decision Conference, 2008. CCDC 2008. Chinese, pages 2902–2907, July 2008.

[29] G.-B. Huang, "What are extreme learning machines? filling the gap between Frank Rosenblatt's dream and John von Neumann's puzzle," Cognitive Computation, vol. 7, no. 3, pp. 263–278, 2015.

[30] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," IEEE Transactions on Neural Networks, vol. 17, no. 6, pp. 1411–1423, 2006.

[31] A. Budiman, M. I. Fanany, and C. Basaruddin. Adaptive online sequential elm for concept drift tackling. Hindawi, 2016.

[32] Y. LeCun and Y. Bengio: Convolutional Networks for Images, Speech, and Time-Series, in Arbib, M. A. (Eds), The Handbook of Brain Theory and Neural Networks, MIT Press, 1995

[33] O. Ludwig Junior, D. Delgado, V. Gonçalves, and U. Nunes, "Trainable classifier-fusion schemes: an application to pedestrian detection," in Proceedings of the 12th International IEEE Conference on Intelligent Transportation Systems (ITSC '09), pp. 1–6, St. Louis, Mo, USA, October 2009.

Appendices

Appendix A Detailed experiment results

All the star network topologies are interconnected through TCP channels. In our experiments, the CNN is consisted of 857738 learnable parameters, each one being a float number, giving a total of 3.44MBytes. The AOS-ELM classifier has 192 learnable parameters, but the algorithm monitors the vector $\hat{W} \in \mathbb{R}^{L^2+L|C|}$ as discussed in §3.5. Hence, for $|C| = 10$ we have approximately a total of 0.156 MBytes.To commute the communication of a singe TCP message we use the formula

$$TCP\ Size = Message\ Size + 40\ \frac{Message\ Size + 1023}{1024},$$

where 40 is the TCP header in bytes. For example, the communication cost in terms of bytes, that is posed to the network topology by a remote site for sending the weights of the aforementioned CNN to the hub is, according to the above formula, 3.566 MBytes.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	8e-4	1	6430	10688	-	25720	255235689824	99.36
2	4	8e-4	2	5294	7848	-	21176	196724179888	99.38
3	4	8e-4	3	3899	3464	-	15596	116927031312	99.38
4	4	8e-3	1	1421	4212	-	5684	78479664256	99.24
5	4	8e-3	2	1250	3677	-	5000	69703280832	99.3
6	4	8e-3	3	1140	3328	-	4560	63582457184	99.33
7	4	8e-2	1	286	855	-	1144	15652016752	99.34
8	4	8e-2	2	279	833	-	1116	15254025984	99.37
9	4	8e-2	3	275	818	-	1100	15048168672	99.35
10	8	8e-4	1	2945	16129	-	23560	384095359120	99.14
11	8	8e-4	2	2116	11204	-	16928	287115990304	99.15
12	8	8e-4	3	1449	6402	-	11592	213570039200	99.13
13	8	8e-3	1	700	4885	-	5600	88992100176	99.06
14	8	8e-3	2	623	4288	-	4984	79282498624	98.92
15	8	8e-3	3	583	3986	-	4664	73889038240	99.06
16	8	8e-2	1	149	1036	-	1192	18410502736	99.1
17	8	8e-2	2	142	987	-	1136	17641968816	99.05
18	8	8e-2	3	137	948	-	1096	17038120784	98.95
19	16	8e-4	1	1466	18464	-	23456	447938512688	98.71
20	16	8e-4	2	990	12264	-	15840	330078358192	98.68
21	16	8e-4	3	660	7312	-	10560	279506062352	98.54
22	16	8e-3	1	371	5506	-	5936	101528802816	98.53
23	16	8e-3	2	330	4826	-	5280	94461033824	98.49
24	16	8e-3	3	302	4374	-	4832	84723986672	98.51
25	16	8e-2	1	79	1170	-	1264	20572003360	98.44
26	16	8e-2	2	79	1171	-	1264	21354260416	98.51
27	16	8e-2	3	75	1111	-	1200	21134678720	98.43
28	32	8e-4	1	798	18962	-	25536	496122827536	98.06
29	32	8e-4	2	553	12442	-	17696	403974224640	97.86
30	32	8e-4	3	381	7259	-	12192	360977468128	97.74
31	32	8e-3	1	210	6507	-	6720	135865779840	97.83
32	32	8e-3	2	184	5470	-	5888	136730369760	97.78
33	32	8e-3	3	164	4791	-	5248	116151511984	97.42
34	32	8e-2	1	49	1488	-	1568	26720272992	97.18
35	32	8e-2	2	47	1443	-	1504	28435748224	97.51
36	32	8e-2	3	44	1361	-	1408	31736322896	97.27
37	64	8e-4	1	558	19547	-	35712	626327578048	97.15
38	64	8e-4	2	315	12526	-	20160	643564494272	96.71
39	64	8e-4	3	253	6627	-	16192	620165354016	96.8
40	64	8e-3	1	123	7711	-	7872	223238417296	96.19
41	64	8e-3	2	107	6242	-	6848	265144052736	95.75
42	64	8e-3	3	94	5334	-	6016	212561272400	95.75
43	64	8e-2	1	34	2079	-	2176	39339321744	95.95
44	64	8e-2	2	29	1764	-	1856	42976127024	95.54
45	64	8e-2	3	27	1630	-	1728	51917186400	96.21

Table 1: CNN training via DMS with uniform site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	8e-4	1	2563	14037	2560	10252	70294045696	99.19
2	4	8e-4	2	2225	7918	2057	8900	58752040668	99.18
3	4	8e-4	3	2132	5470	1545	8528	50449029368	99.27
4	4	8e-3	1	618	5781	971	2472	21793686228	99.37
5	4	8e-3	2	662	3835	817	2648	20283987920	99.26
6	4	8e-3	3	684	2884	701	2736	18993910864	99.29
7	4	8e-2	1	126	1763	264	504	5338646644	99.37
8	4	8e-2	2	127	1497	258	508	5270015620	99.21
9	4	8e-2	3	129	1305	247	516	5146492264	99.29
10	8	8e-4	1	974	9577	1784	7792	75673914912	99.02
11	8	8e-4	2	938	5063	1497	7504	66807970644	99.04
12	8	8e-4	3	921	3399	1155	7368	56954143984	98.99
13	8	8e-3	1	241	3428	564	1928	22068189784	99.04
14	8	8e-3	2	253	2254	547	2024	21930857892	99.06
15	8	8e-3	3	250	1551	485	2000	20146707092	99.02
16	8	8e-2	1	52	973	141	416	5270031252	99.1
17	8	8e-2	2	51	778	141	408	5242566896	99.01
18	8	8e-2	3	51	671	136	408	5105319732	99.1
19	16	8e-4	1	440	6112	1004	7040	79214818788	98.71
20	16	8e-4	2	421	2791	880	6736	71364299568	98.56
21	16	8e-4	3	434	1904	699	6944	62141765020	98.53
22	16	8e-3	1	118	2009	296	1888	22672074540	98.54
23	16	8e-3	2	124	1217	298	1984	23111116512	98.45
24	16	8e-3	3	122	869	284	1952	22232736936	98.66
25	16	8e-2	1	32	560	71	512	5599414100	98.43
26	16	8e-2	2	33	436	71	528	5654288924	98.4
27	16	8e-2	3	34	363	72	544	5764068136	98.44
28	32	8e-4	1	249	3764	520	7968	84320268248	97.71
29	32	8e-4	2	263	1614	492	8416	82782582276	97.63
30	32	8e-4	3	245	1019	370	7840	67411734144	97.54
31	32	8e-3	1	79	1210	157	2528	25801167388	97.58
32	32	8e-3	2	77	663	155	2464	25361841116	97.54
33	32	8e-3	3	78	473	151	2496	25032410744	97.52
34	32	8e-2	1	24	347	39	768	6807132808	97.34
35	32	8e-2	2	23	242	35	736	6258145916	97.44
36	32	8e-2	3	23	193	36	736	6367921428	97.4
37	64	8e-4	1	176	2432	286	11264	101228321220	95.93
38	64	8e-4	2	162	921	275	10368	95737937008	96.37
39	64	8e-4	3	159	501	168	10176	71583792488	96.15
40	64	8e-3	1	51	675	77	3264	27887232464	96.14
41	64	8e-3	2	53	347	78	3392	28545784604	96.09
42	64	8e-3	3	49	253	76	3136	27228236476	95.94
43	64	8e-2	1	16	186	18	1024	7246305560	95.36
44	64	8e-2	2	16	131	18	1024	7246270316	95.33
45	64	8e-2	3	16	108	18	1024	7246255268	95.86

Table 2: CNN training via ML-FGM with uniform site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	8e-4	1	13690	3307	-	54760	243296325344	99.49
2	4	8e-4	2	12723	2014	-	50892	209563158864	99.53
3	4	8e-4	3	9105	893	-	36420	140340097648	99.47
4	4	8e-3	1	4146	4065	-	16584	117270138368	99.49
5	4	8e-3	2	3597	3469	-	14388	104438351536	99.45
6	4	8e-3	3	3288	3080	-	13152	95703132912	99.53
7	4	8e-2	1	644	1545	-	2576	30912908432	99.44
8	4	8e-2	2	633	1519	-	2532	30480607952	99.42
9	4	8e-2	3	609	1519	-	2436	30103202080	99.38
10	8	8e-4	1	6160	15371	-	49280	562251281312	99.38
11	8	8e-4	2	5441	8545	-	43528	387272660288	99.3
12	8	8e-4	3	5037	3120	-	40296	226127667968	99.43
13	8	8e-3	1	1457	9342	-	11656	202007731728	99.33
14	8	8e-3	2	1204	7915	-	9632	195468312928	99.2
15	8	8e-3	3	1016	5763	-	8128	154495866096	99.3
16	8	8e-2	1	275	1917	-	2200	35647617680	99.21
17	8	8e-2	2	270	1881	-	2160	35105526784	99.26
18	8	8e-2	3	260	1805	-	2080	34117411648	99.25
19	16	8e-4	1	2342	22466	-	37472	875044171552	99.18
20	16	8e-4	2	2003	11935	-	32048	586679452768	99.18
21	16	8e-4	3	1722	6586	-	27552	432101287408	99.09
22	16	8e-3	1	702	10389	-	11232	209281344880	99.11
23	16	8e-3	2	593	8517	-	9488	183803074400	99.11
24	16	8e-3	3	494	6956	-	7904	165344533632	99.04
25	16	8e-2	1	147	2190	-	2352	40025514256	98.77
26	16	8e-2	2	141	2115	-	2256	40649946368	98.79
27	16	8e-2	3	136	2023	-	2176	39030536064	98.74
28	32	8e-4	1	1075	24466	-	34400	956872339888	98.76
29	32	8e-4	2	799	13381	-	25568	879888532928	98.73
30	32	8e-4	3	657	7794	-	21024	667176316304	98.67
31	32	8e-3	1	329	9919	-	10528	222236611184	98.62
32	32	8e-3	2	301	8716	-	9632	211086006432	98.55
33	32	8e-3	3	257	7110	-	8224	192401024944	98.47
34	32	8e-2	1	85	2604	-	2720	48705827232	97.97
35	32	8e-2	2	85	2604	-	2720	53207236256	98.0
36	32	8e-2	3	80	2449	-	2560	54058110432	97.88
37	64	8e-4	1	630	24838	-	40320	1293489973552	98.26
38	64	8e-4	2	434	13467	-	27776	1266488230784	97.78
39	64	8e-4	3	375	7950	-	24000	1061035917808	98.01
40	64	8e-3	1	197	12224	-	12608	379868294832	97.64
41	64	8e-3	2	162	8887	-	10368	333866063536	97.52
42	64	8e-3	3	140	7217	-	8960	306706630608	97.44
43	64	8e-2	1	59	3656	-	3776	78555122304	96.73
44	64	8e-2	2	54	3339	-	3456	88038269792	96.72
45	64	8e-2	3	51	3150	-	3264	106613441616	96.74

Table 3: CNN training via DMS with different site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	8e-4	1	2122	15177	2644	8488	65394650552	99.34
2	4	8e-4	2	2161	9623	2188	8644	59671586248	99.47
3	4	8e-4	3	2137	7689	1674	8548	52288096840	99.4
4	4	8e-3	1	625	7235	624	2500	17127645128	99.41
5	4	8e-3	2	613	5433	577	2452	16317859632	99.35
6	4	8e-3	3	621	4100	543	2484	15960984256	99.42
7	4	8e-2	1	103	1191	102	412	2799714240	99.26
8	4	8e-2	2	101	1127	100	404	2744815092	99.28
9	4	8e-2	3	102	1066	101	408	2772259680	99.29
10	8	8e-4	1	1164	10181	1337	9312	68619928792	99.25
11	8	8e-4	2	1135	6547	1197	9080	63980980028	99.29
12	8	8e-4	3	1071	5115	1083	8568	59095185640	99.26
13	8	8e-3	1	232	4044	401	1856	17347243484	99.04
14	8	8e-3	2	230	3050	385	1840	16853101004	99.07
15	8	8e-3	3	231	2356	368	1848	16413882316	99.03
16	8	8e-2	1	43	727	62	344	2854617460	98.99
17	8	8e-2	2	42	596	62	336	2827158320	99.0
18	8	8e-2	3	42	560	62	336	2827154624	99.0
19	16	8e-4	1	459	6277	844	7344	71474613132	99.0
20	16	8e-4	2	431	3881	787	6896	66808137484	98.89
21	16	8e-4	3	435	2899	738	6960	64337704860	98.95
22	16	8e-3	1	113	2223	219	1808	18170697176	98.56
23	16	8e-3	2	107	1520	210	1712	17347155244	98.51
24	16	8e-3	3	104	1182	202	1664	16743251788	98.59
25	16	8e-2	1	27	427	34	432	3293789680	98.36
26	16	8e-2	2	28	371	35	448	3403570408	98.49
27	16	8e-2	3	28	339	35	448	3403564868	98.49
28	32	8e-4	1	205	3600	452	6560	72023676036	98.16
29	32	8e-4	2	209	2141	455	6688	72791767196	98.26
30	32	8e-4	3	198	1493	426	6336	68399956012	98.16
31	32	8e-3	1	66	1165	107	2112	18884352808	97.71
32	32	8e-3	2	63	815	106	2016	18445077556	97.81
33	32	8e-3	3	61	640	102	1952	17786277408	97.74
34	32	8e-2	1	19	258	19	608	4062339012	97.38
35	32	8e-2	2	19	214	19	608	4062324360	97.46
36	32	8e-2	3	19	205	20	608	4172110824	97.41
37	64	8e-4	1	121	2137	240	7744	79050453524	97.03
38	64	8e-4	2	117	1166	233	7488	76634491524	96.98
39	64	8e-4	3	122	836	227	7808	76414722628	97.1
40	64	8e-3	1	43	630	56	2752	21519360644	96.25
41	64	8e-3	2	42	442	55	2688	21080076180	96.28
42	64	8e-3	3	43	357	56	2752	21519182524	96.26
43	64	8e-2	1	13	188	13	832	5489654432	95.48
44	64	8e-2	2	13	141	12	832	5270043208	95.85
45	64	8e-2	3	13	123	13	832	5489612748	96.08

Table 4: CNN training via ML-FGM with different site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	9e+7	1	3524	7220	14096	-	9660859592	98.05
2	4	9e+7	2	3161	2003	12644	-	5466585272	97.97
3	4	9e+7	3	2935	4123	11740	-	6582933080	97.74
4	8	9e+7	1	1834	9958	14672	-	16462400456	98.0
5	8	9e+7	2	1592	3483	12736	-	8482491656	97.45
6	8	9e+7	3	1547	4784	12376	-	9030737960	97.53
7	16	9e+7	1	1047	10616	16752	-	30002818376	97.89
8	16	9e+7	2	912	3895	14592	-	13600883720	97.19
9	16	9e+7	3	860	4847	13760	-	13374375176	97.56
10	32	9e+7	1	441	12106	14112	-	51662189960	97.94
11	32	9e+7	2	456	4454	14592	-	23768122376	97.98
12	32	9e+7	3	460	4884	14720	-	21476765192	97.63
13	64	9e+7	1	235	12220	15040	-	102117573896	96.74
14	64	9e+7	2	230	4380	14720	-	40686816776	97.11
15	64	9e+7	3	223	5160	14272	-	36452839688	96.33

Table 5: AOS-ELM training via DMS with CD 1 and uniform site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	9e+7	1	2301	8918	492	9204	3163770656	97.89
2	4	9e+7	2	2260	9237	608	9040	3184879200	97.97
3	4	9e+7	3	1958	9139	838	7832	2952776504	98.03
4	8	9e+7	1	877	6040	545	7016	2858284736	98.03
5	8	9e+7	2	877	6040	545	7016	2858284736	98.03
6	8	9e+7	3	933	6009	537	7464	2987362788	98.01
7	16	9e+7	1	381	3706	363	6096	2802013896	98.02
8	16	9e+7	2	381	3706	363	6096	2802013896	98.02
9	16	9e+7	3	381	3706	363	6096	2802013896	98.02
10	32	9e+7	1	178	1969	176	5696	2659555376	98.05
11	32	9e+7	2	214	2403	210	6848	3185911000	97.95
12	32	9e+7	3	174	1881	170	5568	2590028964	98.03
13	64	9e+7	1	89	986	83	5696	2627775212	97.92
14	64	9e+7	2	94	1035	89	6016	2786667120	98.05
15	64	9e+7	3	90	1014	86	5760	2677435480	97.94

Table 6: AOS-ELM training via ML-FGM with CD 1 and uniform site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	9e+7	1	3539	7292	14156	-	9624813488	97.37
2	4	9e+7	2	2715	2475	10860	-	5195198768	97.01
3	4	9e+7	3	2861	4228	11444	-	6552028304	97.35
4	8	9e+7	1	1760	10148	14080	-	16010579008	97.37
5	8	9e+7	2	1630	3359	13040	-	8252399488	97.41
6	8	9e+7	3	1537	4782	12296	-	8882691232	97.3
7	16	9e+7	1	878	11543	14048	-	27685668608	97.33
8	16	9e+7	2	777	4086	12432	-	13162228544	97.41
9	16	9e+7	3	770	5269	12320	-	12440451584	97.41
10	32	9e+7	1	461	12125	14752	-	53083424896	97.18
11	32	9e+7	2	429	4307	13728	-	22357800064	97.03
12	32	9e+7	3	447	4998	14304	-	21054563200	97.02
13	64	9e+7	1	226	12328	14464	-	95960987648	97.49
14	64	9e+7	2	205	4576	13120	-	41747525888	97.25
15	64	9e+7	3	219	5175	14016	-	34999175936	96.55

Table 7: AOS-ELM training via DMS with CD 2 and uniform site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	9e+7	1	2499	9607	452	9996	3351149352	97.33
2	4	9e+7	2	2027	8849	707	8108	2926442332	97.44
3	4	9e+7	3	2027	8849	707	8108	2926442332	97.44
4	8	9e+7	1	914	5772	487	7312	2850184596	97.45
5	8	9e+7	2	947	6055	531	7576	2985425848	97.36
6	8	9e+7	3	914	5772	487	7312	2850184596	97.45
7	16	9e+7	1	394	3810	351	6304	2808124532	97.36
8	16	9e+7	2	394	3810	351	6304	2808124532	97.36
9	16	9e+7	3	398	3821	376	6368	2889008292	97.37
10	32	9e+7	1	189	2085	186	6048	2788074704	97.35
11	32	9e+7	2	196	2179	192	6272	2886506144	97.33
12	32	9e+7	3	189	2085	186	6048	2788074704	97.35
13	64	9e+7	1	87	972	82	5568	2546470488	97.33
14	64	9e+7	2	93	1012	87	5952	2713908272	97.38
15	64	9e+7	3	85	965	81	5440	2497213568	97.44

Table 8: AOS-ELM training via ML-FGM with CD 2 and uniform site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	9e+7	1	3668	7042	14672	-	9604839392	98.28
2	4	9e+7	2	2998	2120	11992	-	5319311936	98.27
3	4	9e+7	3	3014	3895	12056	-	6491031872	98.3
4	8	9e+7	1	1811	10185	14488	-	16196762464	98.29
5	8	9e+7	2	1569	3589	12552	-	8492753056	98.27
6	8	9e+7	3	1539	4727	12312	-	8801059168	98.3
7	16	9e+7	1	915	11403	14640	-	27559717568	98.3
8	16	9e+7	2	814	4123	13024	-	13319850752	98.3
9	16	9e+7	3	811	5019	12976	-	12466015424	98.31
10	32	9e+7	1	460	11839	14720	-	49095974656	98.29
11	32	9e+7	2	422	4525	13504	-	23523231232	98.27
12	32	9e+7	3	416	5202	13312	-	19279079680	98.29
13	64	9e+7	1	228	12093	14592	-	93078777344	94.54
14	64	9e+7	2	212	4562	13568	-	40967320064	98.23
15	64	9e+7	3	214	5343	13696	-	32743771136	98.1

Table 9: AOS-ELM training via DMS with CD 3 and uniform site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	9e+7	1	2156	9089	659	8624	3059376948	98.27
2	4	9e+7	2	2166	9051	647	8664	3064279144	98.24
3	4	9e+7	3	2176	9066	640	8704	3072187588	98.27
4	8	9e+7	1	1042	6097	390	8336	3048265296	98.29
5	8	9e+7	2	1045	6049	388	8360	3053226504	98.29
6	8	9e+7	3	1051	6103	381	8408	3059510040	98.3
7	16	9e+7	1	392	3697	337	6272	2765248180	98.27
8	16	9e+7	2	396	3660	332	6336	2772543760	98.27
9	16	9e+7	3	391	3684	338	6256	2762764116	97.8
10	32	9e+7	1	186	2019	181	5952	2735218472	98.25
11	32	9e+7	2	185	2033	182	5920	2730355372	98.25
12	32	9e+7	3	187	1967	179	5984	2735206032	98.25
13	64	9e+7	1	93	1059	90	5952	2744309688	98.23
14	64	9e+7	2	96	1018	87	6144	2774273068	98.24
15	64	9e+7	3	94	1039	89	6016	2754426564	98.23

Table 10: AOS-ELM training via ML-FGM with CD 3 and uniform site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	9e+7	1	3411	7487	13644	-	9485744056	97.34
2	4	9e+7	2	2768	2381	11072	-	5183530792	97.34
3	4	9e+7	3	3524	2883	14096	-	6484024552	97.43
4	8	9e+7	1	1847	10011	14776	-	16425906408	97.45
5	8	9e+7	2	1478	3576	11824	-	8253410952	97.32
6	8	9e+7	3	1426	5137	11408	-	8646919944	97.24
7	16	9e+7	1	856	11656	13696	-	27124846216	97.49
8	16	9e+7	2	751	4239	12016	-	13393406920	97.26
9	16	9e+7	3	801	5118	12816	-	12708529480	97.42
10	32	9e+7	1	426	12178	13632	-	49271793672	96.89
11	32	9e+7	2	377	4662	12064	-	23852971656	96.88
12	32	9e+7	3	469	4730	15008	-	20931611784	97.07
13	64	9e+7	1	261	11653	16704	-	103291024648	96.65
14	64	9e+7	2	181	5061	11584	-	43249193224	96.42
15	64	9e+7	3	249	4676	15936	-	36696334600	97.27

Table 11: AOS-ELM training via DMS with CD 4 and uniform site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	9e+7	1	2164	8627	606	8656	3034176792	97.31
2	4	9e+7	2	2164	8627	606	8656	3034176792	97.31
3	4	9e+7	3	2164	8627	606	8656	3034176792	97.31
4	8	9e+7	1	809	5947	555	6472	2675229032	97.3
5	8	9e+7	2	1029	5931	388	8232	3013797088	97.29
6	8	9e+7	3	1029	5931	388	8232	3013797088	97.29
7	16	9e+7	1	385	3601	334	6160	2724167592	97.28
8	16	9e+7	2	385	3601	334	6160	2724167592	97.28
9	16	9e+7	3	385	3601	334	6160	2724167592	97.28
10	32	9e+7	1	182	1992	179	5824	2688515500	97.27
11	32	9e+7	2	193	2126	190	6176	2850806712	97.23
12	32	9e+7	3	164	1795	159	5248	2412710360	97.3
13	64	9e+7	1	92	1037	87	5888	2701142264	97.47
14	64	9e+7	2	83	983	82	5312	2475299000	97.24
15	64	9e+7	3	91	1018	87	5824	2681665536	97.19

Table 12: AOS-ELM training via ML-FGM with CD 4 and uniform site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	9e+7	1	8047	3525	32188	-	12473277688	97.32
2	4	9e+7	2	6331	1403	25324	-	8868771640	97.33
3	4	9e+7	3	7440	1572	29760	-	10351725352	97.28
4	8	9e+7	1	3632	10425	29056	-	20741746248	97.33
5	8	9e+7	2	3487	3374	27896	-	13234759656	97.38
6	8	9e+7	3	3718	4047	29744	-	12896195208	97.14
7	16	9e+7	1	1712	14293	27392	-	37131549832	97.33
8	16	9e+7	2	2093	4190	33488	-	19988428360	97.13
9	16	9e+7	3	1839	5157	29424	-	17387986888	97.39
10	32	9e+7	1	1203	13351	38496	-	61898783112	96.55
11	32	9e+7	2	1132	4863	36224	-	31111973640	96.34
12	32	9e+7	3	1064	5333	34048	-	25452742920	97.31
13	64	9e+7	1	524	14424	33536	-	105659001352	96.65
14	64	9e+7	2	620	4990	39680	-	49945980424	97.38
15	64	9e+7	3	615	5265	39360	-	41333160712	97.45

Table 13: AOS-ELM training via DMS with CD 4 and different site sampling rates.

Exp	Sites	T	b	Rounds	Subrounds	Rebalances	Safezones	Bytes	Accuracy
1	4	9e+7	1	1579	10556	988	6316	2550881072	97.48
2	4	9e+7	2	1735	11725	1093	6940	2807193780	97.34
3	4	9e+7	3	1765	12092	1169	7060	2890289704	96.93
4	8	9e+7	1	912	6915	508	7296	2871468268	97.04
5	8	9e+7	2	876	6789	519	7008	2796559840	97.45
6	8	9e+7	3	813	6617	534	6504	2659763492	96.34
7	16	9e+7	1	386	3814	294	6176	2630325144	96.9
8	16	9e+7	2	388	3826	271	6208	2584196728	97.13
9	16	9e+7	3	457	4224	298	7312	2990189548	97.29
10	32	9e+7	1	183	2247	174	5856	2673995392	96.46
11	32	9e+7	2	193	2315	171	6176	2757723356	96.8
12	32	9e+7	3	172	2092	156	5504	2477044080	97.24
13	64	9e+7	1	100	1342	98	6400	2967236676	96.05
14	64	9e+7	2	92	1225	90	5888	2731433448	96.28
15	64	9e+7	3	97	1279	97	6208	2897896628	96.69

Table 14: AOS-ELM training via ML-FGM with CD 4 and different site sampling rates.