DiNNO: Distributed Neural Network Optimization for Multi-Robot Collaborative Learning

Javier Yu, Joseph A. Vincent, and Mac Schwager, *Member, IEEE*

***Abstract*—We present DiNNO, a distributed algorithm that enables a group of robots to collaboratively optimize a deep neural network model while communicating over a mesh network. Each robot only has access to its own data and maintains its own version of the neural network, but eventually learns a model that is as good as if it had been trained on all the data centrally. No robot sends raw data over the wireless network, preserving data privacy and ensuring efficient use of wireless bandwidth. At each iteration, each robot approximately optimizes an augmented Lagrangian function, then communicates the resulting weights to its neighbors, updates dual variables, and repeats. Eventually, all robots' local model weights reach a consensus. For convex objective functions, this consensus is a global optimum. Unlike many existing methods we test our algorithm on robotics-related, deep learning tasks with nontrivial model architectures. We compare DiNNO to two benchmark distributed deep learning algorithms in (i) an MNIST image classification task, (ii) a multi-robot implicit mapping task, and (iii) a multi-robot reinforcement learning task. In these experiments we show that DiNNO performs well when faced with nonconvex deep learning objectives, time-varying communication graphs, and streaming data. In all experiments our method outperforms baselines, and was able to achieve validation loss equivalent to centrally trained models. See msl.stanford.edu/projects/dist_nn_train for videos and code.**

***Index Terms*—Deep learning methods, distributed robot systems, multi-robot systems.**

I. Introduction

A GROUP of collaborating robots has the ability to explore, interact with, and experience their environment as a collective much faster than a single robot acting alone. This ability to rapidly gather a large volume and variety of data makes multi-robot systems especially well suited for tasks that involve training deep neural networks using data gathered by the robots. In a cloud robotics scenario, one can imagine thousands of robots networked over a cloud server, able to collectively gather and process vast volumes of data for a common task (e.g. manipulation, autonomous driving, or human behavior prediction). In a mesh network scenario, one can similarly imagine a team of robots collaborating to map an environment, learn a control policy, or learn to visually recognize threats in the environment. A central unsolved problem in collaborative robotics, therefore, is how to train neural network models on the robots through local communication such that each robot benefits from the data collected by the entire multi-robot system.

To solve this problem, we propose Distributed Neural Network Optimization (DiNNO), an algorithm built on the alternating direction method of multipliers (ADMM) [1]. We demonstrate the effectiveness of DiNNO on experiments which require optimizing nonconvex deep learning loss functions which may be subject to time-varying communication graphs and streaming data. In addition, unlike similar approaches, DiNNO is shown to match centralized performance on difficult, multi-robot deep learning tasks while integrating easily with standard tools and optimizers such as PyTorch [2] and Adam [3]. Using DiNNO, robots alternate between local optimization of an objective function, and communication of intermediate model weights over the wireless network. The robots eventually reach a consensus on their model weights, with each robot learning a neural network that is as good as if it had been trained centrally with the data from all robots, as illustrated graphically in Fig. 1. DiNNO inherits the strong convergence properties of ADMM—for convex objective functions we prove that all robots obtain globally optimal parameters. However, neural network training is rarely convex. Using standard deep learning tools within DiNNO we retrieve state-of-the-art deep learning performance, but in a distributed, multi-robot implementation. Finally, DiNNO operates by sharing model weights over the communication network, not raw data. Therefore, robots using DiNNO preserve the privacy and integrity of their own local data set. This is crucial in scenarios where user data or observations of humans are involved, or when robot manufacturers must preserve the privacy of their own data sets.

A naive approach to solving the multi-robot deep learning problem is to use a mesh network routing protocol to aggregate the data gathered by all of the robots in the system to a single "leader" robot which then optimizes a deep neural network model, and sends a copy of that trained model back to all of the other robots in the system. We refer to this approach as a "centralized" solution, and it has a number of distinct drawbacks. First, depending on the size of the gathered data, algebraic connectivity of the communication graph, bandwidth of the communication links, and efficiency of the routing protocol, it can take a significant amount of time to aggregate the gathered

Manuscript received September 9, 2021; accepted January 3, 2022. Date of publication January 13, 2022; date of current version January 21, 2022. This letter was recommended for publication by Associate Editor G. A. Sartoretti and Editor M.-A. Hsieh upon evaluation of the reviewers' comments. The work of Javier Yu was supported by NSF Graduate Research Fellowship and the work of Joseph A. Vincent was supported by Dwight D. Eisenhower Transportation Fellowship. This work was supported in part by NASA ULI under Grant 80NSSC20M0163, and in part by NSF NRI under Grants 1925030 and 1830402. *(Corresponding author: Javier Yu.)*

The authors are with the Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305 USA (e-mail: javieryu@stanford.edu; josephav@stanford.edu; schwager@stanford.edu).

Digital Object Identifier 10.1109/LRA.2022.3142402

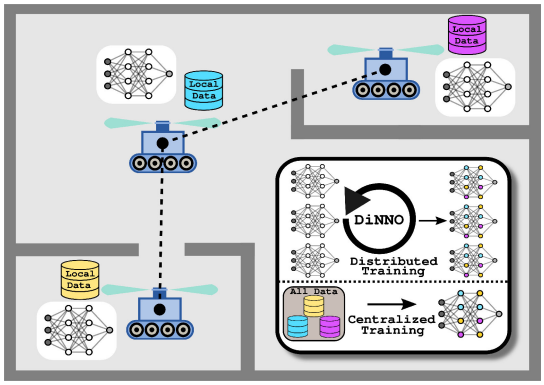


Fig. 1. DiNNO allows robots to cooperatively optimize local copies of a neural network model without explicitly sharing data. In this figure (representative of Section V-B), three robots use DiNNO to cooperatively optimize a building occupancy map represented as a neural network. Each robot only sees part of the building, collecting a local lidar data set (colored cylinders). The robots communicate over a wireless network (dashed lines) to cooperatively optimize their local neural network copies. The resulting model is as good as if it were trained centrally with all data at once.

data at the leader node. A centralized approach is not robust to failure of the leader node, and in some applications it may not be possible to transmit data due to privacy considerations, for instance, due to the European Union General Data Protection Regulation article 46 [4]. DiNNO overcomes all of these limitations by enabling leaderless distributed neural network training through local communication among the robots.

The paper is organized as follows. We give related work in Section II and introduce the distributed collaborative learning problem in Section III. In Section IV we derive DiNNO starting from a well-known variant of ADMM called Consensus ADMM. In Section V we present three example robotic deep learning tasks that showcase our method.

II. Related Work

Learning has been used to address a variety of problems for multi-robot systems and is only increasing in popularity. A deep learned controller is used to model multi-quadrotor interactions in [5], and is shown to considerably outperform traditional non-linear controllers. Reinforcement learning can also be a useful tool in multi-robot contexts for real-time and uncertainty aware collision avoidance [6] and communication resilient collaborative learning [7]. The collective training methods in [8] and [9] demonstrate how experience aggregation from multiple robots can speed up policy optimization. While not deep learning, Gaussian processes are another popular learning tool that has been used for various regression tasks with data collected online by multi-robot systems [10]–[12]. These works all showcase applicability of collaborative learning models for multi-robot systems, but in general, aside from [11], [12], do not provide a distributed framework from which to perform this learning.

Research on distributed deep learning for robotics tasks includes [13], [14]. The authors of [13] apply distributed gradient descent to multi-agent reinforcement learning (MARL) problems, and largely focus on the theoretical implications of these distributed training algorithms when the learned functions are linear. However, their experiments with nonlinear function approximation are limited to small neural networks. In [14], a novel federated learning framework is introduced which allows for distributed learning. In this framework, a global model is stored in shared-memory between all robots and updated based on averaging locally learned models after training epochs through a network flooding procedure.

The problem of training neural networks in a distributed way using data aggregated from individual robots can be viewed as a specific instantiation of a distributed optimization problem. Distributed optimization is the study of algorithms for solving optimization problems where a sum of individual objective functions, which correspond in this case to the individual robots, is optimized using local computation and message passing. This formulation was first proposed in [15], and has been of renewed interest since the seminal work [16] which presented distributed subgradient descent for convex distributed optimization problems. Subsequent research has focused on improving convergence rates [17] and extending the analysis to a broader range of problems including time-varying communication graphs [18] and streaming convex objectives [19]. An overview of the broader distributed optimization literature is given in the surveys of [20]–[22].

Some works in the distributed optimization literature address general nonconvex distributed optimization objectives and even consider simple distributed neural network training problems as examples. In [23], the distributed subgradient descent algorithm is extended to distributed stochastic gradient descent (DSGD), and uses training of a CIFAR-10 classification model as a benchmark problem. The Choco-SGD algorithm for distributed deep learning [24] is another algorithm similar to DSGD with the variations that it uses a gossip mechanism for consensus, and incorporates a quantization step for reducing communication bandwidth. One approach to improving convergence rates is to introduce an auxiliary variable that estimates the global gradient. Several works make use of this mechanism, and extend it to the domain of nonconvex optimization with stochastic gradients [25]–[27]. We refer to these methods as distributed stochastic gradient tracking (DSGT) methods.

Compared to DSGD and DiNNO, the gradient tracking methods of [25]–[27] communicate twice as many parameters at each round (primal variable *and* gradient estimate). We believe a distinct advantage of DiNNO over [14], [23]–[27] is that DiNNO uses a primal-dual method to achieve consensus. The other approaches utilize parameter averaging to ensure consensus which, as noted in [28], can lead slow convergence when robots have different data distributions. Our primal-dual consensus approach is more robust to differences in local data. In our experiments (V) we compare against DSGD and DSGT and show that DiNNO outperforms both methods.

The edge consensus learning algorithm proposed in [28] is similar to our approach in that it is derived from ADMM, but instead of addressing the nonconvex primal update directly it uses a linearization similar to that proposed in [29], which results in a gradient descent like update. While still technically a primal-dual method, the update equations do not include a local

optimization procedure, and are more similar to primal domain methods like DSGD and DSGT (see [29], Remark 1). A number of other nonconvex distributed optimization methods are discussed in the survey [30]. Compared to DiNNO, [28] allows for asynchronous updates, but does not consider multi-robot deep learning tasks and we believe DiNNO is simpler to implement. Our proposed algorithm for distributed deep learning, DiNNO, demonstrates superior performance compared to DSGD and DSGT benchmarks on multi-robot deep learning tasks such as neural implicit mapping and deep multi-agent reinforcement learning.

III. Problem Formulation

We consider deep learning problems where portions of a data set, $\mathcal{D}$, are collected by N robots that operate in a connected communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Let $\mathcal{D}_i$ be the portion of local data that belongs to robot $i \in \mathcal{V}$ where the union of all the local data sets, $\mathcal{D}_i$, is the joint data set, $\mathcal{D}$. In some cases $\mathcal{D}_i$ can represent access to a time-varying data set gathered from a private data-stream (as in Section V-B).

The model we would like to optimize has the form $y = f(x;\, \theta)$. Specifically we consider f to be a deep neural network that implements a continuous function $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ to give a map from inputs $x \in \mathbb{R}^n$ to outputs $y \in \mathbb{R}^m$. The neural network is parameterized by model weights $\theta \in \mathbb{R}^d$ where d is the number of parameters. We make no special assumptions about the architecture (e.g. feed-forward, convolutional, residual, etc.). We can then formalize our distributed learning optimization problem as

$$\underset{\theta \in \mathbf{R}^d}{\text{minimize}} \quad \sum_{i \in \mathcal{V}} \ell(\theta; \mathcal{D}_i) \tag{1}$$

where $\ell(\cdot)$ is the objective function (loss function) which is generally nonconvex and often nonsmooth (due to ReLU activation). Common deep learning tasks such as classification, regression, and unsupervised learning have different objective functions and a distributed deep learning optimizer should be general enough to achieve good performance across all of these problems.

Suppose that the decision variable, θ, is separated such that each robot maintains their own instance of it, $\theta_i \in \mathbb{R}^d$. This yields the equivalent optimization problem

$$\underset{\theta \in \mathbf{R}^d}{\text{minimize}} \quad \sum_{i \in \mathcal{V}} \ell(\theta_i; \mathcal{D}_i) \tag{2a}$$

$$\text{subject to} \quad \theta_i = \theta_j \quad \forall (i,j) \in \mathcal{E}. \tag{2b}$$

This optimization problem is amenable to a distributed solution in which robots minimize local objective functions, and take additional steps to come to agreement (consensus) on the value of the decision variable. Replacing the data defined loss functions in (2) with arbitrary objective functions yields the general formulation of a distributed optimization problem.

IV. Distributed Training

A standard method for solving convex distributed optimization problems is the consensus alternating direction method of multipliers (CADMM) [31]. CADMM is an ADMM-based optimization method where compute nodes (robots) alternate between updating their primal and dual variables and communicating with neighboring nodes. To achieve a distributed primal-dual update, CADMM introduces auxiliary primal variables (i.e. $\theta_i = z_{ij}$ and $\theta_j = z_{ij}$ instead of $\theta_i = \theta_j$). CADMM works by first optimizing the auxiliary primal variables, followed by the original primal variables, then the dual variables, as in the original formulation of ADMM [1]. Implementations of CADMM then perform minimization with respect to the primal variables and gradient ascent with respect to the dual on an augmented Lagrangian that is fully distributed among the robots:

$$\mathcal{L}_a = \sum_{i \in \mathcal{V}} \ell(\theta_i) + p_i^\top \theta_i + \frac{\rho}{2} \sum_{j \in \mathcal{N}_i} ||\theta_i - z_{ij}||_2^2 \tag{3}$$

where p_i represents the dual variable that enforces agreement between node i and its communication neighbors, and $\mathcal{N}_i$ is the set of indices for neighboring nodes of i. The parameter ρ that weights the quadratic terms in $\mathcal{L}_a$ is also the step size in the gradient ascent of the dual variable. Furthermore, the algorithm can be simplified by noting that the auxiliary primal variable update can be performed implicitly ($z_{ij} = \frac{1}{2}(\theta_i + \theta_j)$). Initializing the dual variables at zero then yields the following distributed update equations for CADMM:

$$p_i^{k+1} = p_i^k + \rho \sum_{j \in \mathcal{N}_i} (\theta_i^k - \theta_j^k) \tag{4a}$$

$$\theta_i^{k+1} = \underset{\theta}{\text{argmin}} \;\; \ell(\theta; \mathcal{D}_i) + \theta^\top p_i^{k+1} + \rho \sum_{j \in \mathcal{N}_i} \left\| \theta - \frac{\theta_i^k + \theta_j^k}{2} \right\|_2^2. \tag{4b}$$

Typically, the primal variables are initialized uniformly to an initial guess $\theta_i^0 = \theta_{\text{initial}}$. This derivation of CADMM is addressed in much finer detail in [32].

In CADMM, the objective function for the primal update (4b) is composed of three terms: a neural network loss on the robot's local data, a linear term from the dual variable, and a regularization term. It is obvious that applying CADMM directly to the neural network training problem results in intractable primal updates due to the neural network loss component. The key insight, which we use in our algorithm DiNNO, is that this primal optimization can be performed approximately, stopping well before convergence to a local optimum. Formally, we propose replacing the exact minimization of the primal update, (4b), with an approximate solution found by taking a small number of steps, B (typically between 2 and 10), of a stochastic first order method (SFO) on the entire primal objective function.

Our proposed algorithm is shown in Algorithm 1 with the approximate primal update performed in lines 12 - 16. Furthermore, in Algorithm 1 we let k represent the current communication round and τ represent the current step taken by a SFO in this communication round. We replace the current primal iterate with ψ in order to avoid including two iteration count super scripts, and let $G(\psi^\tau; \rho, p_i^{k+1}, \theta_i^k, \{\theta_j^k\}_{j \in \mathcal{N}_i}, \mathcal{D}_i)$ represent the step taken by a SFO on the objective in the primal update.

Algorithm 1 Distributed Neural Network Optimization (DiNNO)

1: **Require:** $\ell(\cdot)$, $\theta_{initial}$, $\mathcal{G}$, $\mathcal{D}$, ρ
2: **for** $i \in \mathcal{V}$ **do** ▷Initialize the iterates
3: $p_i^0 = 0$ ▷Dual variable
4: $\theta_i^0 = \theta_{initial}$ ▷Primal variable
5: **end for**
7: **for** $k \leftarrow 0$ to K **do** ▷Main optimization loop
8: **Communicate:** send θ_i^k to neighbors $\mathcal{G}$
9: **for** $i \in \mathcal{V}$ **do** ▷In parallel
10: $p_i^{k+1} = p_i^k + \rho \sum_{j \in \mathcal{N}_i} (\theta_i^k - \theta_j^k)$
11: $\psi^0 = \theta_i^k$
12: **for** $\tau \leftarrow 0$ to B **do** ▷Approximate primal
13: $\psi^{\tau+1} = \psi^\tau + G(\psi^\tau; \rho, p_i^{k+1}, \theta_i^k, \{\theta_j^k\}_{j \in \mathcal{N}_i}, \mathcal{D}_i)$
14: **end for**
15: $\theta_i^{k+1} = \psi^B$ ▷Update primal
16: **end for**
17: **end for**
19: **return** $\{\theta_i^K\}_{i \in \mathcal{V}}$

To be clear, G computes a stochastic gradient over $\mathcal{D}_i$ not a gradient on the full local data set.

In some of our experiments we found it beneficial to also add a "scheduled" increase (Algorithm 1, line 9) for the penalty parameter ρ in similar fashion to the learning rate schedules used in deep learning. For notational simplicity, we overload the variable ρ to also mean this schedule of parameter values, and make explicit note of all cases where one is used. Although generally we leave this term constant, it can be useful to gradually increase it when faster consensus is desired. This schedule can be provided to robots prior to optimization, and does not compromise the distributed nature of DiNNO.

An added benefit of DiNNO is that it pairs well with existing deep learning libraries because the approximate primal minimization can be performed with minimal changes to the typical training loops used to optimize individual neural networks. We find that this is beneficial because automatic differentiation and state-of-the-art neural network optimizers, like Adam, can be used to perform the approximate primal update, and practitioner knowledge from experience training individual neural networks is transferable.

A. Convergence Properties

Corollary 1 (Convex Optimality of Algorithm 1): Let each local objective function $l(\theta, \mathcal{D}_i)$ be strongly convex and L-smooth. Furthermore, let

$$G = -\frac{1}{L}\nabla_\theta \left(l(\theta, \mathcal{D}_i) + \theta^\top p_i^{k+1} + \rho \sum_{j \in \mathcal{N}_i} \left\| \theta - \frac{\theta_i^k + \theta_j^k}{2} \right\|_2^2 \right)$$

and suppose the number of gradient steps $B \to \infty$. Then Algorithm 1 converges to the unique global solution with linear convergence rate.

Proof: Given strongly convex and L-smooth local objectives, the primal update (Algorithm 1 line 13) converges to the global solution with a linear convergence rate as shown in [33]. Given globally optimal primal updates and the stated assumptions, Algorithm 1 is a special case of the decentralized ADMM algorithm studied in [34] where it was shown to have linear convergence to the global solution. ■

Clearly, in deep learning problems global solutions and linear convergence are not ensured due to neural networks creating nonconvex, and often nonsmooth, objective functions, making general convergence results extremely challenging to prove. Moreover, in distributed deep learning problems it is impractical to take exact gradients and perform many stochastic gradient steps (B) to solve each subproblem to a high degree of accuracy. However, it is known that inexact primal updates converge to the global solution under convexity assumptions [32]. Though the form of our inexact primal update is slightly different, we also observe, even when using stochastic gradients and few descent steps for each subproblem, Algorithm 1 converges to solutions similar in quality to those from centralized optimization.

B. Baseline Algorithms

In Section V we show that DiNNO is an extremely effective method for distributed training of neural network models. We compare DiNNO against two other commonly referenced stochastic first order distributed optimization methods: DSGD [23] and DSGT [26]. Like DiNNO, both methods have each robot maintain a local copy of the optimization variable (neural network weights), and use message passing and locally computed stochastic gradients to collaboratively optimize the neural network. DSGD uses the update

$$\theta_i^{k+1} = \sum_{j \in \mathcal{V}} w_{ij} \theta_j^k - \alpha^k g(\theta_i^k) \tag{5}$$

where w_{ij} is an element of a doubly stochastic matrix W that has a sparsity pattern matching that of the graph Laplacian of $\mathcal{G}$, α^k is a decaying step size, and $g(\theta_i^k)$ is a stochastic (or mini-batch) gradient of $\ell(\theta_i^k; \mathcal{D}_i)$. While (5) may not at first appear to be a distributed algorithm, the sparsity pattern of W means that each node only needs θ_j^k from its immediate neighbors to compute its update step.

The updates for DSGT are similar to those of DSGD, but an additional auxiliary variable is added to estimate the gradient of the joint loss,

$$\theta_i^{k+1} = \sum_{j \in \mathcal{V}} w_{ij} (\theta_j^k - \alpha y_j^k) \tag{6}$$

$$y_i^{k+1} = \sum_{j \in \mathcal{V}} w_{ij} y_j^k + g(\theta_i^{k+1}) - g(\theta_i^k). \tag{7}$$

It is important to note that for DSGT the message size sent at each communication round is double that of both DSGD and DiNNO which only send θ_i^k. For DSGT and DSGD we use the Metropolis-Hastings weights as W. Alternative benchmark algorithms include [24], [25], [27], [28], [30] but, in general, they share many core characteristics with the proposed baselines DSGT and DSGD.

C. Data Distributions

The way in which local data is partitioned between the robots strongly influences the convergence rate of distributed optimization. In classification tasks, for example, problems where each robot has access to a subset of examples from all classes are easier to solve with distributed optimization than problems in which each robot only has access to labelled data for a single class. We refer to these two data distributions as *homogenous* and *heterogenous* respectively. In homogeneous classification a robot which optimizes directly on its local data set without communication may be able to achieve a relatively high classification accuracy. A robot in the heterogeneous case is unlikely to achieve a high accuracy on any class other than what it observed.

D. Limitations and Future Work

One limitation of DiNNO is that the current formulation does not allow for asynchronous updates. However, introducing additional dual variables to the DiNNO formulation, as in [28], would enable asynchronous updates at the cost of added communication complexity. Though DiNNO is robust to most robot failures, like deletion, it can be vulnerable to repeated package drops and adversarial attacks because asymmetric communication between robots can result in steady state error in the solution. There is a substantial literature on resilient consensus algorithms [35], and extending these strategies to DiNNO is an interesting direction for further research. In applications, like computer vision, where data collected by robots has a large memory footprint DiNNO is more communication efficient than centralized methods. When the local data is small and the communication graph is highly connected flooding schemes that transmit all data to a leader node can sometimes require less communication than DiNNO. However, flooding schemes have numerous points of failure, and lack the privacy benefits of DiNNO. One method for reducing the communication overhead of DiNNO is message quantization, as in [24], which is another avenue for future research.

V. Experiments

In the following three examples we demonstrate that DiNNO can be applied to a wide range of multi-robot learning applications, and demonstrates a substantial improvement against baseline distributed optimization algorithms. In each example we compare with two other common distributed optimization methods: DSGD and DSGT. We implement DiNNO, DSGD, and DSGT in a general framework such that for each experiment the optimization algorithm is unchanged, but a different objective function and data set are provided. Hyperparameter values are reported in Section VI.

A. MNIST Classification

To clearly illustrate the potential for Algorithm 1 to train a shared neural network from disparate data observers, we first consider the well known MNIST classification problem [36]. Here a neural network learns to classify images of handwritten digits. We train a model composed of a convolutional layer with three 5x5 filters followed by 2 linear layers of width $576, 64$ with ReLU activation and a log-softmax output layer. We use the negative log-likelihood loss function. Each robot only has access to labelled digits from a single class.

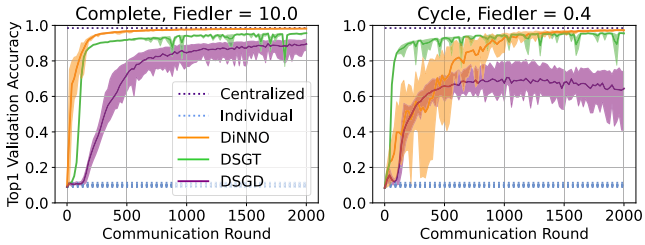


Fig. 2. Each plot shows the Top-1 accuracy of the neural network models on the validation set of the MNIST problem. For each algorithm we evaluate the local neural networks stored by each robot on the validation set. Solid lines show the average validation accuracy across all robots at the current communication round, and filled areas are upper and lower bounded by the best and worst performing robots for each particular algorithm. The Fiedler value indicates the connectivity of the graph.

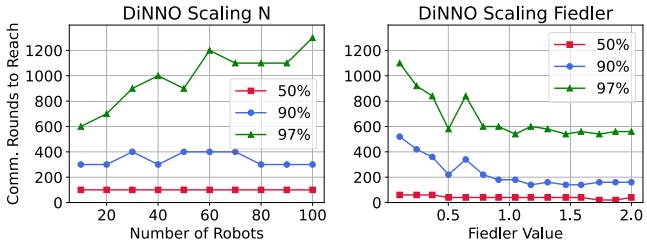


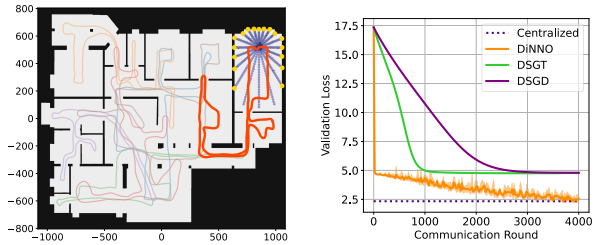
Fig. 3. These plots show the number of communication rounds required by DiNNO for the average validation accuracy to reach a specific threshold on the distributed MNIST problem as both number of robots (left) and the Fiedler value of the communication graph (right) are increased. When the number of robots is increased (Fiedler value is 1 ± 0.01) DiNNO consistently reaches 90% accuracy in less than 400 communication rounds, but requires more time to converge to high accuracy (97%) as the network grows. Increasing the Fiedler value, but holding the number of robots constant at 20, we see that DiNNO is slower to converge when the Fiedler value is low (low connectivity), but speeds up when the Fiedler value increases.

In Fig. 2 we show the average, worst, and best Top-1 accuracy for each method on the distributed MNIST classification problem with 10 robots on two different communication graphs (complete and cycle). Also included is the centralized result (98.5% validation accuracy), and the individual results, which as expected have $\sim$ 10% validation accuracy. While DSGT quickly trains to good accuracy in these problems, DiNNO achieves much better final accuracy as training progresses with lower variance in later iterations. DSGD has relatively poor performance with high variance.

To understand how DiNNO scales to larger networks of robots and its performance under different network connectivity we repeated the distributed MNIST training under a range of conditions, and show the results in Fig. 3. To vary network size we generate geometric graphs with a target number of robots and a communication radius that yields a graph with a Fiedler value (algebraic connectivity) of 1 ± 0.01. The data is sorted by label and divided evenly amongst robots so each robot has examples from at most two classes (heterogeneous data). We then train using DiNNO, and record the communication rounds required to reach certain accuracy thresholds. The rounds required to reach

50% and 90% accuracy remain roughly constant across network sizes, but more rounds are required to reach 97% accuracy when networks are large. We believe this is because as each robot has progressively less local data, fine tuning of weights becomes more difficult. Notably, even with 100 robots and a local data set of only 600 images DiNNO converges to an accuracy matching that of centralized training.

We use a similar approach to test how DiNNO performs with a range of different Fiedler values. For this test we fix the network size at 20 robots, and again generate geometric graphs with communication radius chosen to obtain desired Fiedler values. The data is divided as before, and we train using DiNNO. Predictably, DiNNO is slower to converge when the connectivity is low, but speeds up as the connectivity increases. Fiedler values larger than 1 have little increase in performance suggesting that consensus no longer is a limiting factor in the convergence rate.



(a) (b)

Fig. 4. (a) Ground truth map. Highlighted is a single robot's trajectory and a single lidar scan is shown with high (gold) and low (blue) density points. (b) Average validation loss versus communication iteration for the neural implicit mapping experiment with maximum and minimum values plotted in a lighter shade (DSGD and DSGT have high agreement throughout so these bounds are not visible). Both baseline algorithms DSGT and DSGD appear to consistently converge to a poor quality minima while DiNNO (ours) converges to a model with validation loss matching that of the centralized solution. Though DSGT and DSGD appear to converge to a similar local minimum, Fig. 5 shows that the reconstructions are different.

B. Neural Implicit Mapping

In robotics there is growing interest in using neural networks to represent functions which implicitly define the geometry of an environment [37], [38]. In their basic form, implicit density field networks take as input an (x, y, z) spatial coordinate and output a single density value between 0 and 1. Such networks are able to represent complicated 3D scenes in a single memory-efficient function. In this example we use DiNNO to learn the density field of a two dimensional environment where data collection and computation is distributed across multiple robots. The robots also have access to a global coordinate frame which enables cooperative mapping, but a future line of research would be to implement this same pipeline in conjunction with a distributed pose optimization algorithm.

The environment we seek to map is a 2D building floorplan environment from the CubiCasa5K data set [39]. This data set does not include the scale of the floorplans, thus we treat each pixel as one unit. Seven robots are deployed, and each robot gathers data from the environment by collecting lidar scans as it traverses a closed loop, precomputed trajectory. To simulate data streaming the robots update their local networks at regular intervals from data sets of their last 400 collected lidar scans (one trajectory has 3000-4000 scans). Fig. 4(a) shows the ground truth environment with seven robot paths and one lidar scan. There is some overlap in the locations traversed by each robot, but many locations, especially at the borders, are only viewed by one robot.

We train a feedforward network with four hidden layers of size $256, 64, 64, 64$ where the first hidden layer has sinusoidal activation, the remaining hidden layers have ReLU activation, and the output layer has sigmoid activation to restrict our density estimates to (0,1). The sinusoidal activations are common in implicit mapping [40]. We use binary cross entropy loss between the sampled and predicted density.

The validation set is composed of novel lidar scans from uniformly sampled locations across the entire map, and this ensures that the validation data reflects loss only on areas where the robots have can gather data (not inside walls). For the communication graph, we use a geometric graph based on the positions of the robots, where the radius is set to 1500 units. The motion of the robots results in a time-varying graph which we observe is always connected.

Fig. 4(b) shows the validation loss for our method as well as DSGD and DSGT. DiNNO best minimizes the validation loss, once again approaching the performance of centralized training whereas DSGD and DSGT train less effectively, converging to poor quality solutions. Fig. 5 shows the map learned by each method, and maps from individual robots training on only their local data. As suggested by Fig. 4(b), when using DiNNO robots are able to provide a faithful reconstruction of the ground truth environment whereas with DSGD and DSGT robots converge to incoherent maps.

To verify the performances of DSGT and DSGD we reran this experiment several times, and both methods always converged to poor performing local minima. Additionally, we emphasize that the implementation for these two methods is unchanged between this experiment and Section V-A where both methods learn acceptable classifiers. We speculate that this is a challenging problem where only a small amount of suboptimality is allowable to achieve a useful representation. DSGT and DSGD may be unable to either fine tune their weights, escape poor local minima, or handle streaming data.

C. Multi-Agent Reinforcement Learning

For the final example we use DiNNO for distributed learning of a decentralized policy applied to a standard continuous state and action, multi-robot, predator-prey problem that was first introduced in [41]. MARL is known to be an especially hard learning task due to the inherent *nonstationarity* of the environment. That is, the environment changes during learning because other agents also have evolving policies. For more background on deep MARL see [42] and [43].

In our learning environment three robots must work together to pursue a faster evader robot in the presence of fixed obstacles, as shown in Fig. 6(a). Implemented in PettingZoo [44], the environment operates according to the

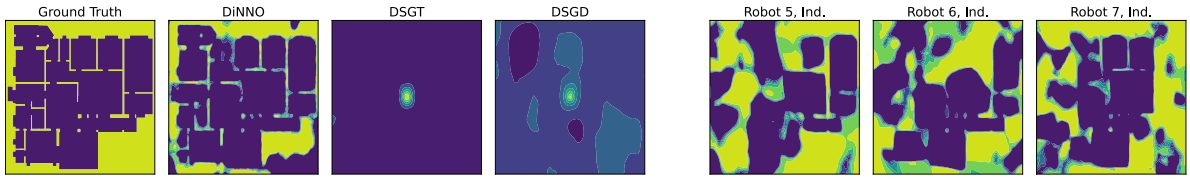
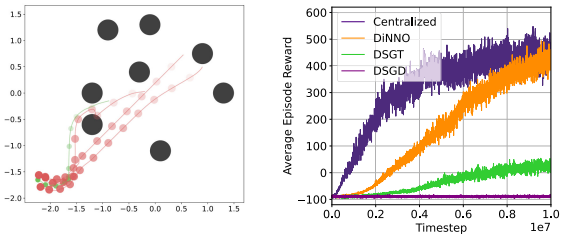


Fig. 5. The left most plot shows the ground truth density map, and moving right the next three plots are the reconstructions from the neural implicit maps found by the three tested distributed algorithms. Here DiNNO is the only method that is able to learn a coherent map. The reconstructions were produced by querying the optimized (and agreed upon) networks on a grid mesh of points on the map. The last three plots show reconstructions produced from three of the seven robots when communication is not used (training exclusively on local data with Adam for 10 epochs). Since these robots do not have information from other areas on the map, they are only able to reconstruct regions which those robots have traversed. Though visually the individually learned maps appear betarned by DSGT and DSGD, they actually have a higher validation loss compared to DSGT and DSGD.



(a) (b)

Fig. 6. (a) A decentralized policy rollout in the predator-prey environment. Pursuers (red), using policies learned with DiNNO applied to PPO, attempt to capture a faster evader (green) in the presence of obstacles (black). (b) Episodic reward (averaged across 10 episodes per network update) vs environment time steps (summed across all episodes) for DiNNO, DSGD, DSGT, and centralized. Shown results are averaged across 5 training runs to show training robustness. DiNNO is the only algorithm to achieve good performance, matching centralized performance after 10 million time steps.

Actor Environment Cycle Game model in which pursuers make observations, act, and receive rewards sequentially before the environment as a whole is updated. The pursuers have actions $a =$ `[none, right, left, up, down]` $\in \mathcal{A} \subset \mathbb{R}^5$ and observations $o =$`[self_vel, self_pos, other_pursuers_rel_pos, evader_rel_pos, evader_rel_vel]` $\in \mathcal{O} \subset \mathbb{R}^{12}$. Actions are clipped to be on the interval [0,1]. The evader obeys a heuristic policy, moving opposite the nearest pursuer. To prevent unfair evasion, the evader cannot propel itself outside a square of radius 1.2. The reward function penalizes pursuing robots based on their distance from the evader and pursuers receive a positive reward for tagging the evader.

To solve this problem we extend the PPO actor-critic algorithm [45] with DiNNO to train a shared, decentralized policy. At consensus the robots all converge to the same policy in accordance with a parameter sharing approach which has been shown to be effective for MARL problems [42], [46]. Typically the policies for parameter sharing are trained by some centralized compute node that aggregates the experiences of each of the robots. Applying DiNNO to PPO results in a relatively unexplored paradigm for MARL where both training and execution are distributed.

In this example the actor and critic networks are feedforward ReLU networks with 3 hidden layers of 64 neurons each. The robots communicate through a fully connected graph and update their policies using individually collected data every 10 episodes. Results from this experiment are shown in Fig. 6(b). For each algorithm we show the mean (over 5 runs) of the average episodic reward achieved by the multi-robot predator team as training progresses.

DiNNO achieves the same average episodic reward as a policy trained using PPO with aggregated data from all three robots. DSGD seems unable to learn a policy that results in positive episodic reward and though DSGT learns a policy with positive episodic reward, it is far inferior to DiNNO's.

VI. Conclusion

We present the DiNNO algorithm that enables high performance distributed training of deep neural networks for multi-robot teams with streaming data and time-varying communication graphs. We showcase DiNNO's versatility on three diverse multi-robot learning tasks. Compared to existing distributed learning methods our algorithm consistently achieves better validation performance and converges to performance of centrally trained models. Directions for future work include learning neural implicit density functions from real 3D depth data and exploring the capabilities of DiNNO for more complex distributed reinforcement learning tasks. Links to our code and a video visualizing the experiments can be found here: msl.stanford.edu/projects/dist_nn_train.

A. Hyperparameters

MNIST: Hyperparameters used across all four graphs were the same. DiNNO uses $B = 2$, $\rho_0 = 0.5$ increasing 0.3% per communication round, and Adam as its primal optimizer with a log learning rate schedule (5e-3 - 5e-4) for the primal update. DSGT uses $\alpha =$5e-3. DSGD uses a decaying stepsize following $\alpha^{k+1} = \alpha^k(1 - \mu\alpha^k)$ where $\alpha^0 =$5e-3 and $\mu =$1e-3. All methods use batch size 64.

Implicit Mapping: DiNNO uses $B = 5$, $\rho^0 = 0.1$ increasing 0.3% per communication round, batch size of 1e5, and Adam with a log learning rate schedule of (1e-3 - 1e-4). DSGT uses $\alpha =$ 1e-3, and a batch size of 2e5. DSGD uses $\alpha^0 =$ 1e-3, $\mu =$ 1e-3, and a batch size of 2e5.

Multi-agent Reinforcement Learning: For each algorithm we use the following hyperparameters: 200 steps per episode, 2e3 steps between actor/critic network updates, reward discount factor $\gamma = 0.99$, and PPO clipping parameter 0.2. We allow each algorithm 5 gradient steps ($B = 5$) per batch of data to update actor and critic networks. The actor learning rates for DiNNO, DSGD, and DSGT are 3e-4, 1e-3, and 1e-2, respectively. Only

DSGT has a separate critic learning rate, 1e-5, due to exploding gradients otherwise. For DiNNO we set a constant $\rho = 1$.

REFERENCES

[1] S. Boyd, N. Parikh, and E. Chu, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Boston, MA, USA: Now Publishers Inc, 2011.

[2] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, arXiv:1412.6980.

[4] B. Custers, A. M. Sears, F. Dechesne, I. Georgieva, T. Tani, and S. Van der Hof, *EU Personal Data Protection in Policy and Practice*. Berlin, Germany: Springer, Mar. 2019.

[5] G. Shi, W. Hönig, X. Shi, Y. Yue, and S.-J. Chung, "Neural-swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions," *IEEE Trans. Robot.*, pp. 1–17, 2021, doi: 10.1109/TRO.2021.3098436.

[6] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 285–292.

[7] K. Corder, M. M. Vindiola, and K. Decker, "Decentralized multi-agent actor-critic with generative inference," 2019, *arXiv:1910.03058*.

[8] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine, "Collective robot reinforcement learning with distributed asynchronous guided policy search," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 79–86.

[9] G. Sartoretti, Y. Wu, W. Paivine, T. S. Kumar, S. Koenig, and H. Choset, "Distributed reinforcement learning for multi-robot decentralized collective construction," in *Proc. Distrib. Auton. Robot. Syst.*, 2019, pp. 35–49.

[10] W. Luo and K. Sycara, "Adaptive sampling and online learning in multi-robot sensor coverage with mixture of gaussian processes," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 6359–6364.

[11] G. Habibi and J. P. How, "Human trajectory prediction using similarity-based multi-model fusion," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 715–722, Apr. 2021.

[12] W. Luo, C. Nam, G. Kantor, and K. Sycara, "Distributed environmental modeling and adaptive sampling for multi-robot sensor coverage," in *Proc. 18th Int. Conf. Auton. Agents MultiAgent Syst.*, 2019, pp. 1488–1496.

[13] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 5872–5881.

[14] N. Majcherczyk, N. Srishankar, and C. Pinciroli, "Flow-FL: Data-driven federated learning for spatio-temporal predictions in multi-robot systems," *IEEE Int. Conf. Robot. Automat.*, pp. 8836–8842, 2021.

[15] J. N. Tsitsiklis, "Problems in decentralized decision making and computation," *Massachusetts Inst. Tech Cambridge Lab for Inf. and Decis. Syst.*, Tech. Rep., 1984.

[16] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, Jan. 2009.

[17] W. Shi, Q. Ling, G. Wu, and W. Yin, "Extra: An exact first-order algorithm for decentralized consensus optimization," *SIAM J. Optim.*, vol. 25, no. 2, pp. 944–966, 2015.

[18] A. Nedić and A. Olshevsky, "Distributed optimization over time-varying directed graphs," *IEEE Trans. Autom. Control*, vol. 60, no. 3, pp. 601–615, Mar. 2015.

[19] S. Hosseini, A. Chapman, and M. Mesbahi, "Online distributed convex optimization on dynamic networks," *IEEE Trans. Autom. Control*, vol. 61, no. 11, pp. 3545–3550, Nov. 2016.

[20] T. Halsted, O. Shorinwa, J. Yu, and M. Schwager, "A survey of distributed optimization methods for multi-robot systems," 2021, *arXiv:2103.12840*.

[21] A. Nedić, A. Olshevsky, and M. G. Rabbat, "Network topology and communication-computation tradeoffs in decentralized optimization," *Proc. IEEE*, vol. 106, no. 5, pp. 953–976, May 2018.

[22] T. Yang *et al.*, "A survey of distributed optimization," *Annu. Rev. Control*, vol. 47, pp. 278–305, 2019.

[23] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5336–5346.

[24] A. Koloskova, T. Lin, S. U. Stich, and M. Jaggi, "Decentralized deep learning with arbitrary communication compression," in *Proc. Int. Conf. Learn. Representations*, 2020.

[25] P. D. Lorenzo and S. Scardapane, "Parallel and distributed training of neural networks via successive convex approximation," in *Proc. IEEE 26th Int. Workshop Mach. Learn. Signal Process.*, 2016, pp. 1–6.

[26] S. Pu and A. Nedić, "Distributed stochastic gradient tracking methods," *Math. Program.*, vol. 187, no. 1, pp. 409–457, 2021.

[27] S. Lu, X. Zhang, H. Sun, and M. Hong, "GNSD: A gradient-tracking based nonconvex stochastic algorithm for decentralized optimization," in *Proc. IEEE Data Sci. Workshop*, 2019, pp. 315–321.

[28] K. Niwa, N. Harada, G. Zhang, and W. B. Kleijn, "Edge-consensus learning: Deep learning on P2P networks with nonhomogeneous data," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 668–678.

[29] Q. Ling, W. Shi, G. Wu, and A. Ribeiro, "DLM: Decentralized linearized alternating direction method of multipliers," *IEEE Trans. Signal Process.*, vol. 63, no. 15, pp. 4051–4064, Aug. 2015.

[30] T.-H. Chang, M. Hong, H.-T. Wai, X. Zhang, and S. Lu, "Distributed learning in the nonconvex world: From batch data to streaming and beyond," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 26–38, May 2020.

[31] G. Mateos, J. A. Bazerque, and G. B. Giannakis, "Distributed sparse linear regression," *IEEE Trans. Signal Process.*, vol. 58, no. 10, pp. 5262–5276, Oct. 2010.

[32] T.-H. Chang, M. Hong, and X. Wang, "Multi-agent distributed optimization via inexact consensus ADMM," *IEEE Trans. Signal Process.*, vol. 63, no. 2, pp. 482–497, Jan. 2015.

[33] H. Karimi, J. Nutini, and M. Schmidt, "Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2016, pp. 795–811.

[34] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the linear convergence of the ADMM in decentralized consensus optimization," *IEEE Trans. Signal Process.*, vol. 62, no. 7, pp. 1750–1761, Apr. 2014.

[35] D. Saldana, A. Prorok, S. Sundaram, M. F. Campos, and V. Kumar, "Resilient consensus for time-varying networks of dynamic agents," in *Proc. Amer. Control Conf.*, 2017, pp. 252–258.

[36] Y. LeCun, "The mnist database of handwritten digits," 1998. [Online]. Available: http://yann. lecun. com/exdb/mnist/

[37] B. Mildenhall, P.P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NERF: Representing scenes as neural radiance fields for view synthesis," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 405–421.

[38] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "iMAP: Implicit mapping and positioning in real-time," in *Proc. IEEE/CVF Int. Conf. Comput. Vision*, Oct. 2021, pp. 6229–6238.

[39] A. Kalervo, J. Ylioinas, M. Häikiö, A. Karhu, and J. Kannala, "CubiCasa5K: A dataset and an improved multi-task model for floorplan image analysis," in *Proc. Scand. Conf. Image Anal.*, 2019, pp. 28–40.

[40] M. Tancik *et al.*, "Fourier features let networks learn high frequency functions in low dimensional domains," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020.

[41] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6382–6393.

[42] J. K. Terry, N. Grammel, A. Hari, L. Santos, and B. Black, "Revisiting parameter sharing in multi-agent deep reinforcement learning," 2020, *arXiv:2005.13625*.

[43] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, "Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks," in *Proc. 35th Conf. Neural Inf. Process. Syst. Datasets Benchmarks Track*, 2021, https://github.com/uoe-agents/epymarl.

[44] J. K. Terry *et al.*, "PettingZoo: Gym for multi-agent reinforcement learning," 2020, *arXiv:2009.14471*.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[46] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2017, pp. 66–83.