Reputation-Aware Gossip Learning

Suhas Raja,[1] **Chi-Fan Lo,** [1] **Zejian Huang** [1]

[1] Carnegie Mellon University
suhasr@cs.cmu.edu, chifanl@cs.cmu.edu, zejianh@cs.cmu.edu

Abstract

In this work, we explore how dynamic policies for incorporating peer models can improve the resilience of gossip learning, a distributed learning framework, in unbalanced situations. We define settings where agents have limited amounts of potentially unbalanced local data, as well as potentially non-learning peers, and train agents using reinforcement learning and imitation learning methods to determine the combination factor used when communicating with different agents. Our imitation-based policies outperform both baseline and reinforcement learning-based policies in all settings, regardless of data homogeneity or the presence of non-learning agents. This work highlights the importance of dynamically selecting combination factors in gossip learning and may serve as a foundation for further research on the topic.

Introduction

In recent decades, we have witnessed an unprecedented amount of data being generated and an increasing need to leverage that data, leading to the rise of machine learning. Traditionally, machine learning models are trained on a central server across large data sets. The viability of these approaches at scale has become ambiguous due to a rapidly increasing amount of data and rising concerns of privacy and ethical issues: aggregating all data into a central server may become costly, inefficient, and even infeasible; sharing data among institutes would inevitably increase the security risks of information leakage. In such cases, distributed machine learning stands as an alternative to centralized learning. In distributed architectures, data can be kept locally - both alleviating the burden of transmitting large amounts data and providing strong privacy guarantees, while the learned models are communicated, synthesized, and redistributed across nodes.

A well-known approach to this challenge is a federated learning architecture [11, 9]. In a federated learning scheme, there is one central server with multiple client nodes. Each node accumulates a local dataset to train a local model. These models are transmitted to the central server, which then aggregates the parameters across models. The combined model is then redistributed to each client. This scheme is effective at improving privacy guarantees - since a user's data never leaves their device, the architecture is fundamentally privacy safe. Substantial work has been conducted to extend federated learning to be more efficient and robust. [19] The key constraint of federated learning is that all clients rely entirely on the central server. If the server goes down (due to technical issues, deprecation, etc), clients are left in the cold. In addition, federated learning requires clients to be capable of maintaining reliable and periodic communication to this central node, which may not be possible under some scenarios.

As a response, researchers developed Gossip Learning as a fully decentralized architecutre. [6]. In gossip-based architectures, clients communicate directly with each other. That is, no central resource is available for synchronization or model aggregation. By omission of a central server, gossip learning is capable of extending beyond the aforementioned limits of federated learning [17] and communicating in more flexible ways. Overall, gossip learning performs comparably to federated learning [7]. However, it requires more advanced communication protocols - no longer can a client node simply transmit their model to a server and trust that the model it receives is generalizable, but it must think critically about how to incorporate information received from any given peer.

Some work has been conducted on gossip learning architectures. Most research has focused on benchmarking gossip-learning against federated and purely centralized schemes, or exploring different mechanisms for model averaging [7, 17]. These works have failed to account for the potential diversity across agents and data that may exist in purely decentralized settings.

Most current gossip learning work implicitly assumes that peers have equally viable models, so an agent can simply incorporate peers' models into its own in a static fashion. Such an assumption may not be realistic. Certain agents may have worse or less generalizable models than others, due to worse choice of training hyper-parameters, late start of training, or even an adversarial mindset. In face of these uncertainties, collective success in communication-sparse environments crucially relies on considering received messages according to some estimate of peer viability.

For example, consider the case of optical number recognition in the MNIST dataset. If 4 peers each only have data for the labels 0-2, 3-5, 6-7, and 8-9, federated learning is somewhat capable of accommodating this heterogeneous data distribution. Each node trains a local model for their data, and the centralized model updates operate as a ”sanity check” to bring each model back to a global center. Gossip learning cannot afford such a luxury. Since pairwise interactions may be asynchronous with each other or biased towards certain clients more than others, there is no force to bring nodes back to a global aggregate of model parameters. In another case, if one peer is initialized with unreasonable hyper-parameters such as an unreasonably large learning rate, it's unlikely to produce good model. Any peer blindly incorporating this bad-performing model without considering it's viability would degrade the quality of their model.

In such settings, we want to explore how to optimize communication policies to improve the robustness of gossip learning to heterogeneity of agents, both in terms of data distributions and peer model qualities. We propose a variant of gossip learning that allows agents to approximate ”reputation” - that is, how well they tend to generalize to peer datasets, and use this to inform peers on how they should weigh model updates from that agent.

Related work

Related Approaches

In the face of higher cost and rising concerns of data privacy for centralized learning, federated learning is proposed to distribute the training process to client nodes [11, 9]. Multiple client nodes can train local models based on locally collected data and send to the central server, which synthesize all models and redistribute to client nodes. Besides traditional machine learning domain, approaches similar to federated learning are applied to reinforcement learning as well [14]. However, this method heavily relies on the central server. If it goes down, client nodes would not be able to receive updates from other clients. Moreover, each round of communication requires up-stream and down-stream traffic between the server and all client nodes, which may cause unsatisfying overheads [19]. Further, a central server may not be available in all contexts.

Gossip learning is introduced as an alternative to address the situations when access to a central server is infeasible [17, 7]. Instead of communicating to the central server, peers communicate directly with each other.

Among current gossip learning approaches, gossip averaging [1, 8] is widely used. The work by Hegedús et al. [7] on gossip learning simply averages parameters between models, while work by Li et al [10] evaluates the local gradient of both the local and peer models for one iteration of training. Based on gossip averaging, some approaches try to solve the problem of who to communicate with and what to communicate [19, 12, 18].

Other related works require synchronous communication, which is infeasible in a large population. The work by Onoszko and Karlsson et al. [16] weights messages equally, but chooses a ”clique” of agents to communicate with. It assumes an agent has T rounds of communication to select m friends, then it only communicate with those friends afterwards. This is limiting since it requires $O(T)$ preliminary rounds of synchronous communication. Furthermore, it segregates sub-populations with diverse data, which makes learning a well-generalized model hard. Finally, instead of a group reputation score, the paper only considers a pairwise reputation score, which is limiting since each agents reputation information can not be passed to other agents. The work by Che et al. [3] uses a committee mechanism, but requires excessive synchronous communication where agents "activate" one another and have various designated roles. The heterogeneity inherent in this design, due to role assignment, is inaccurate to real-world conditions, e.g. if phones were to crowd-source keyboard recognition models, it is likely not viable to have leader and follower phones, with all initialized at once. We decide to use a much more practical way to simulate agent interaction, discussed in further detail in section 4.

Current asynchronous gossip learning approaches make an implicit assumption that evaluating a peer's model on an agent's local dataset is an accurate way of estimating the "trustworthiness" or the generalizability of a peer model. Such an assumption may not be realistic, and can lead agents to incorporate poor models - which is especially likely in adversarial contexts, or those where an agent may have poor local data. [5, 15, 2]. This reduces resiliency to heterogeneous data distributions, where an agents data/-model may be biased. Without accounting for the relative strength/weakness of individual peers, incorporating all peer models equally may not guarantee fast convergence to the optimal model.

Supporting Work

The paper ”Learning by Cheating” [4] presents an approach to training models, where some preliminary model is encouraged to cheat by looking at the ground-truth labels during training. This is different from the traditional approach, where the model is only given access some limited perspective on the input and must learn to make optimal predictions. This allows the model to learn from the ”cheating” behavior of the first stage, and achieve better performance on the task at hand. We apply a similar paradigm in our own work.

Problem Statement

Current settings in gossip learning framework assumes that all peers are equally trustworthy and that every agent updates their own model with a static and universal algorithm. However, such an assumption is not realistic, especially in face of heterogeneous distribution of data and variable quality of peer models.

We plan to design a new peer to peer communication protocol to resolve this vulnerability by allowing agents to maintain estimates of model generality, and incorporate those into model updates.

Our goal as researchers is to define a system that leads to an equilibria where, for any randomly selected agent from a population, their expected performance on the universal dataset is maximized. Note that this goal is not observable to individual agents and the agents do not have access to universal dataset. To work around this, each agent initially aims to maximize a proxy reward that is designed to estimate the generality of their own model. We measure generality by considering the class-wise f1 score (abbreviated as f1 or F1 in the following content) on test data, which is not seen by any agents during training time.

We cannot use an oracle to calculate real rewards at runtime, as it would violate the structure of gossip learning - that there cannot be any central authority containing the global dataset, in order to preserve privacy and the distributed nature of the protocol.

We proceed to define the observation, actions and interaction procedure below. Since our main method that succeed do not need reward, our reward section for Reinforcement Learning experiments is defined in Appendix. Initially, we assume that all agents are benevolent, i.e., they do not intentionally send bad models or feedback to peers, and we introduce agents with limited capacity - specifically, agents whose models cannot learn. This emulates the setting where agents might join the environment with poor models, and covers the case where an adversarial policy might transmit fully random models in hopes of disrupting a peer.

Environment

For simplicity, we assume all agents are located close enough to each other, so that they can communicate with any other agents.

Observation

The observation space is represented by a 30-tuple:

$$\{F1(M_a, D_a) - F1(M_b, D_a), F1(M_b, D_b) - F1(M_a, D_b), Z\},$$

where each variable represents a vector across 10 MNIST labels. $F1(M_a, D_b)$ denotes the accuracy of a's model on b's dataset, when a is communicating with b. The first variable represents the "advantage" of a's model against b's model on its own dataset, and the second variable represents the "disadvantage" on b's dataset. The final variable, Z, measures the L1 distance in the agent distributions across the 10 labels in the MNIST dataset.

At the beginning of a communication round, agents share their models, local accuracies, and distributions. Then, they share each other's accuracy on their own dataset, allowing the state to be calculated. In future work, we hope to incorporate an agent's "history" of performance on peer datasets - this would allow agents to approximate the strength of their models across various peer datasets.

Actions

Suppose that a is now paired with b and about to incorporate both models. We define the action as the choice of how much to incorporate b's model with its own to generate a better model. The action space is confined to be a real number $\beta_a \in [0, 1]$, such that the resulting model would be a convex combination of the two models. For each weight $w_{i,a}$ in agent a's model, we update it by

$$w_{i,a} \leftarrow \beta_a \cdot w_{i,a} + (1 - \beta_a) \cdot w_{i,b}$$

As an alternative to combining parameters, we also consider combining local gradient of two models directly. Our experiments showed this method to be ineffective in practice, thus we omit the discussion and result for this methods.

Communication Procedure

In each time step, agents would be randomly grouped into pairs of 2 based on geographical proximity, represented by being in the same coordinate C in the grid world. If an odd number of agents are located in the same grid, a random one would be ignored. The communication process between the 2-agent-pair contains 5 stages (described in the perspective of one agent a):

1. Evaluate local model on local data, obtain $F1(M_a, D_a)$
2. Send local model to peer, and receive peer model M_b
3. Evaluate peer model on local data to obtain accuracy $F1(M_b, D_a)$
4. Send $F1(M_a, D_a)$, $F1(M_b, D_a)$, and distribution of local data Z_a to peer agent, receive peer's evaluation both models on peer data $F1(M_b, D_b)$, $F1(M_a, D_b)$ and peer's distribution Z_b
5. Update local model

Note that only steps $2, 4$ require communication, which means the communication between a pair has 2 rounds. A thorough algorithm including local learning and interaction is provided below (algorithm 1) as pseudo code.

Evaluation

We plan to test our method on the MNIST hand-written digit classification task. Each agent will begin with models of identical architectures, but randomly initialized parameters. At the end of each iteration, we evaluate the average performance of each agent's model on the full dataset. For each experiment, we adjust the following hyperparameters and consider how they affect final results.

1. **The data distributions across agents.** We are considering how we want to distribute data of each label, and may explore schemes where distributions are a function of location - this may help mimic the real-world conditions of data locality and biased collection.
2. **RL algorithms for the β-policy.** Some may work better than others to develop an incorporation policy.

Algorithm 1: Overall procedure

```
Input: $N$ agents with same underlying model structure with random initialized parameter $M_1, \dots, M_N$ and distribution across 10 MNIST labels $Z_1, \dots, Z_N$, local learning frequency $F$, total communication rounds $T$
for $t = 1, \dots, T$ do
    for $i = 1, \dots, F$ do
        Train local model $M_a$ on local data $D_a$
    end for
    Randomly divides $N$ agents into pairs of two. Exclude one agent if $N$ is odd.
    for Agent pair $(a, b)$ do
        Evaluate local model on local data, obtain $F1(M_a, D_a)$
        Send local model $M_a$ to peer, obtain peer model $M_b$
        Evaluate peer model on local data, obtain accuracy $F1(M_b, D_a)$
        Send $F1(M_a, D_a)$, $F1(M_b, D_a)$, $Z_a$, obtain $F1(M_b, D_b)$, $F1(M_a, D_b)$, $Z_b$
        Calculate $\beta_a$ and update model $M_a$
    end for
end for
```

3. **Percentage of Nonlearning Peers.** To simulate difficult environments, where certain agents may enter the system with random models, or fail to learn viable local models, we consider the case where there are peers which cannot train their models. By demonstrating our approaches resilience to this situation, we show that our approach can withstand difficult environments much better than fixed averaging schemes.

Methods

We developed 6 different modes of beta selection for comparison: 1 fixed mode as a baseline, 2 RL algorithms, and 3 related to our imitation-based methods.

Fixed β

In the first case, agents always incorporate all peer models equally ($\beta = .5$), as in the original Gossip Learning formulation [6]. This is used as a baseline for our methods.

Alternatively, agents may ignore all other models ($\beta = 1$). This simulates the simplest case, where agents simply train on their own data. If any method does not beat this, that indicates we have given each agent too much data - they may as well just train locally. We intend to test the case where local data training is suboptimal compared to distributed training, so this baseline serves to indicate if we have violated this constraint.

REINFORCE

We discretized the β value into 11 values (0, 0.1, 0.2, ..., 1.0) and treat this as a discrete action space for an reinforcement learning agent. We used the classic REINFORCE algorithm [20] to train it, which utilized policy gradient to perform back-propagation. The β-model π is a simple two-layer MLP.

DDPG

We treat the β learning problem as an actor-critic problem where we learn a continuous-action actor and a critic (Q-function) that is differentiate. Then, we can update the actor by back-propagating through the critic as mentioned in the DDPG paper [13].

Cheater

The "cheater" policy is a special algorithm that serves as the "expert" for imitation algorithms and a useful reference. It is not viable as a deployable policy. In cheater mode, each agent has access to the global dataset. At each communication round, it searches across multiple β choices $(0, 0.1, 0.2, \dots, 1)$, tests the resulting composite model on the global dataset, and picks the best choice. It can be viewed as doing a grid search over different β values in a greedy approach. We also log the 30-tuple observation with the best β value as the expert data.

It's noteworthy that this cheater policy is not an optimal policy. It sometimes perform suboptimally. We discuss the possible reasons in the appendix.

Linear Imitation & Derived Heuristic

We trained both logistic and clipped linear regressions on data collected from the cheater. Specifically, we used the dataset of (30-tuple observation, β value) pairs from the cheater on various environments, and trained the regressors to predict the optimal beta. 1 shows results of the two regressions. At test time, clipped linear regression performs better than the logistic regression, so we omit the discussion of logistic regression later on.

	Linear Regression	Logistic regression
Train MSE	0.0195	0.0640
Train MAE	0.0921	0.2018
Test MSE	0.0134	0.0632
Test MAE	0.0812	0.2019

Table 1: Regression result

We then deployed these trained models into the environments where they could select β values according to the current state despite not having access to the global dataset. The clipped linear regression will clip the output to range $[0, 1]$ to align with the action space.

The clipped linear regression is essentially a mapping from 30 inputs to one β value. One noteworthy observation is that, the ideal weights would be the same across 10 labels. Under this hypothesis, we create a flatten version of the clipped linear regression, averaging the first 10, second 10, and last 10 weights. The resulting model has weights 0.1005 on advantage term on local data (per label), weights -0.1005 on disadvantage term on peer data (per label), weights close to 0 on distributional difference term, and a bias of 0.5 (per label).

From this flatten linear model, we derived a heuristic to choose β from the observation: $\beta = 0.5 + 0.1 \cdot \sum[F1(M_X, D_X) - F1(M_Y, D_X)] - 0.1 \cdot \sum[F1(M_Y, D_Y) - F1(M_X, D_Y)]$

The fact that imitation learners recovered this heuristic from the cheater when looking at a relatively large state space confirms this is a near optimal heuristic, and it does perform well as we'll show in the next section.

Nonlinear Imitation

We also prototyped neural models for nonlinear imitation of the cheater, similar to the aforementioned linear models for imitation. To preserve label invariance, we applied a parameter-sharing approach described as follows. Label invariance is critical - suppose we have two different experiments, where the only difference is the label of the data. The optimal policy should behave the same in both situations, so our policy should similarly remain the same.

Our parameter-sharing approach is like a 1-D convolution over class-wise F1, as illustrated in the figure 1. For simplicity, we only applied 1 filter in each layer, and 1 layer in total (no stacking). This simple approach does not work as well as heuristic, however there is a large space for improvement, including scaling up the number of filters and layers.

Experiments and Results

We tested the methods discussed in section 4 in 4 experiment settings to consider their ability to select superior β values for combing peer models. Our two imitation-based policies, heuristics and neural network, constantly matches or outperforms the cheater and other algorithms in all settings.

In all 4 settings, there are 20 agents, each with 30 images as local dataset. The environment contains 100 communication rounds, each preceded by 5 steps of local learning. We mainly consider 2 factors that influence the gossip learning difficulty: distribution of local data, and percentage of non-learning (NL) agents. We use 2 types of data distribution to allocate local dataset for each agent: a balanced, uniform distribution across 10 labels, and an unbalanced distribution with 4 random labels over-represented by 15 times. We refer to the former as ”balanced” and the latter as ”unbalanced”. NL agents skip their local learning steps and cannot combine with peer models, meaning their model parameters never change from their random initializations. We use them as a simple representation of bad models to avoid combining with. The 4 settings are set as follows:

1. Balanced local datasets, 0% NL agents
2. Unbalanced local datasets, 0% NL agents
3. Balanced local datasets, 40% NL agents
4. Unbalanced local datasets, 40% NL agents

Balanced datasets, no NL agent

This environment is an ideal setting for gossip learning. Each agent has balanced local dataset, so local learning could generate models that perform decently on the global dataset. However, since the size of local data is limited, it might not suffice to merely learn from local data. Combining models with peers then becomes beneficial. The absence of NL agents makes the averaging strategy ($\beta = 0.5$) a well performing model in this case. As figure 2 shows, fix $\beta = 0.5$ performs comparably well to the cheater, which serves as a reference for a close-to-optimal policy, while fix $\beta = 1$ suffers from insufficient local data and is suboptimal. Among other policies, the imitation-derived heuristic and neural network outperforms the cheater by a tight margin, reinforce gets similar result, and ddpg is also suboptimal. This shows our approach can at least perform comparably well to the previous gossip learning policies with fixed $\beta = 0.5$ combining rate in simple settings.

Unbalanced datasets, no NL agents

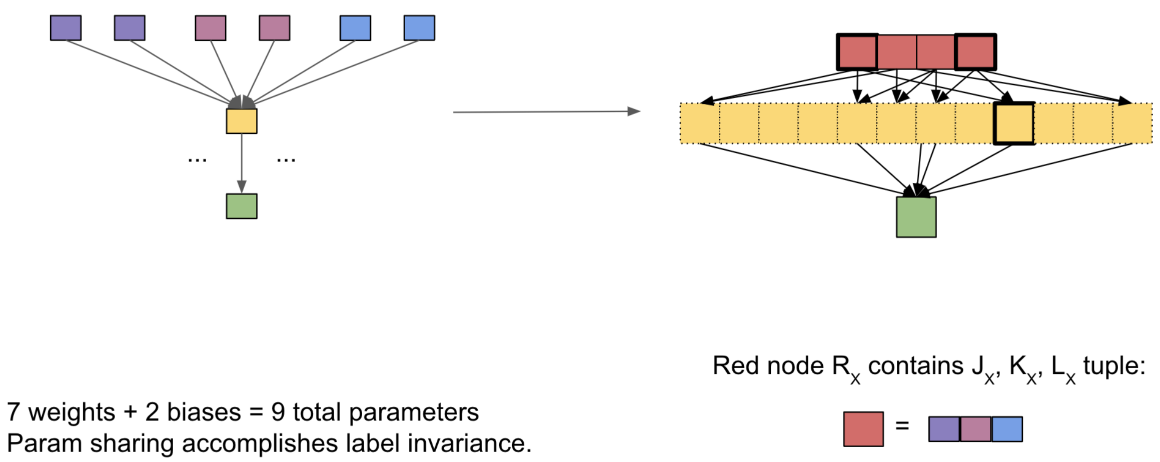
This second environment is slightly harder comparing to the first. Unbalanced dataset increases the need for communication and learning from peer models. Figure 3 shows the result. $\beta = 1$ performs badly, since local data is heavily skewed to 4 labels. A fixed combination factor of $\beta = 0.5$ is no longer close to optimal, possibly due to different relative data distribution and performance. The imitation-based heuristic and neural network reaches accuracy close to the cheater, while rl-based reinforce and ddpg perform worse.

Balanced datasets, 40% NL agent

With the introduction of NL agents, it is no longer beneficial to blindly combine with peer models, because of the risk of being hurt by bad model parameters. A dynamic policy to choose β when communicating with different peers may become beneficial. Figure 4 shows that fixed $\beta = 0.5$ can only reach suboptimal result. Reinforce and ddpg both performs poorly. The two imitation-based policies instead perform slightly better than the cheater. This proves that considering the relative performance and data distribution can help with choosing better β, and dynamic β selection is useful to combat combining with bad models.

Unbalanced datasets, 40% NL agent

This environment setting is the hardest among the 4. Agents are forced to communicate with peers because of the unbalanced and limited local dataset, while the presence of NL agents make the communication risk high. Again, fixed $\beta = 1$, $\beta = 0.5$, reinforce and ddpg all perform poorly. It might be suprising to see that the cheater also doesn't perform well. We explain some insights in the appendix. Our two imitation-based policies still work relatively well in this hard setting, beating the cheater by a significant margin. This shows our approach can perform well even in significantly challenging environments.



...
...
Red node R_x contains J_x, K_x, L_x tuple:
=
7 weights + 2 biases = 9 total parameters
Param sharing accomplishes label invariance.

Figure 1: Nonlinear Imitation

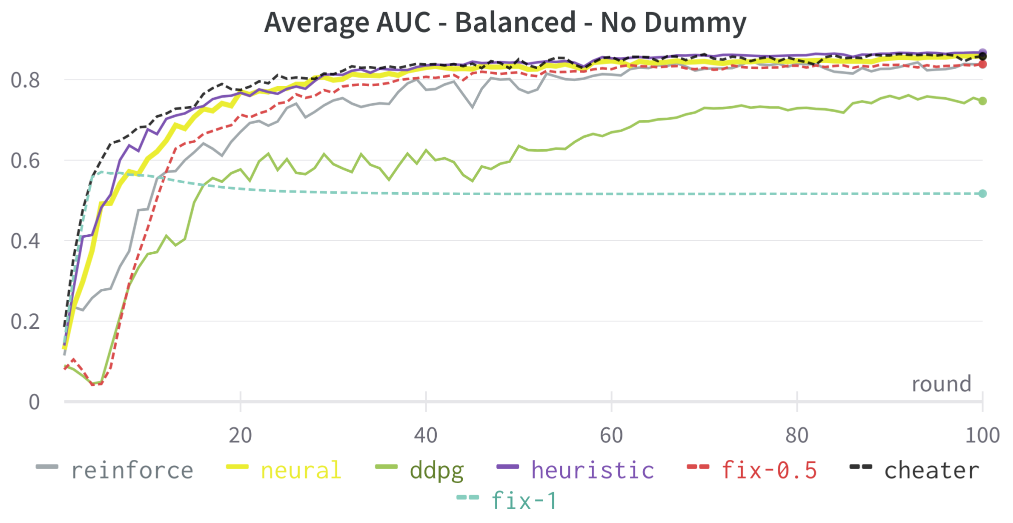


Figure 2: Balanced dataset, no NL agent

Future Work

We have the following ideas to extend this work, to either improve performance or better emulate real-world conditions.

Emulating Real Conditions

Currently, our system randomly pairs peers together for comunication at each round. In a real environment, this would translate to all the peers being in constant proximity. Similarly, real-world conditions may also imply that an agent's distribution of data is sensitive to that agent's location of origin - for example, self driving cars from colder or warmer climates may have different datasets. To accommodate these situations, we intend to initialize agents in different locations in a grid world, and condition their local data distributions based on these locations (i.e. agents in a particular corner may overrepresent some labels in their local set).

Improving Nonlinear Imitation Methods

We have only prototyped one architecture for the neural regressor, which provided lackluster results. While the tested architecture was not effective, we suspect that it is possible for other network architectures to have superior performance and hope to explore it in the future. One possible future step to try is to scale up our 1D-CNN structure described in section 4.6.

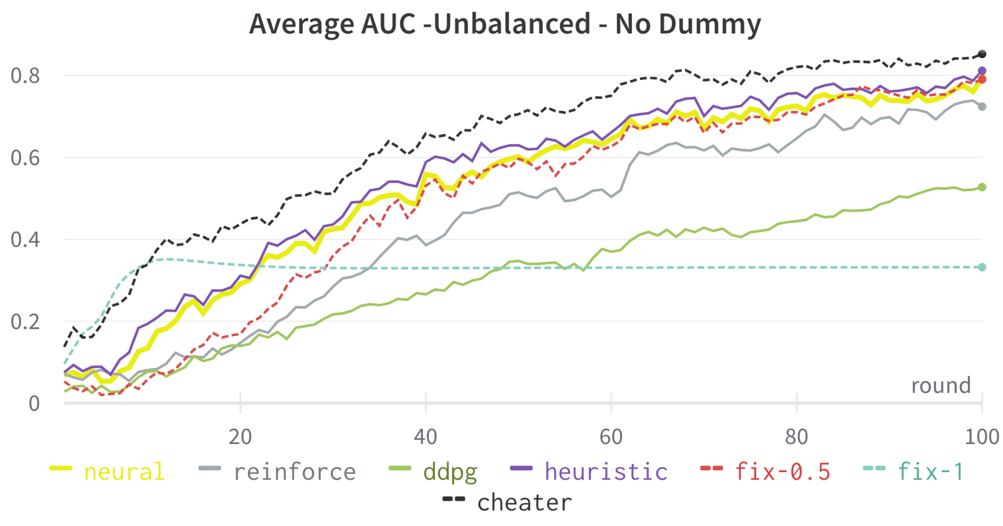


Figure 3: Unbalanced dataset, no NL agent

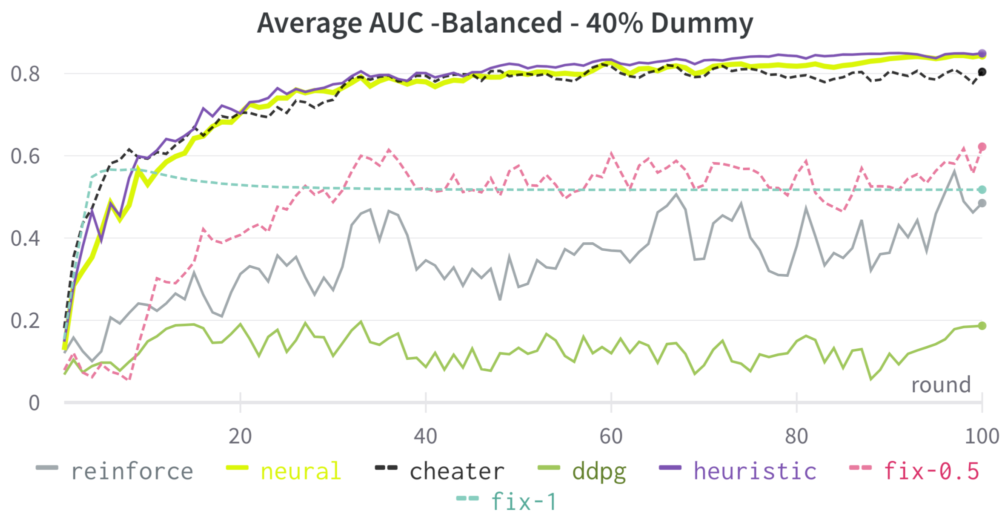


Figure 4: Balanced dataset, 40% NL agent

Augmenting Local Datasets with Encoded Sharing

It is often possible to train networks to learn based on encoded representations of input samples, as is often done in the case of encoder-decoder architectures. We would like to further explore whether it is possible to incorporate encoded data sharing, where agents can share the most ”distinctive” or ”useful” samples with peers. This would allow agents to gradually balance out their dataset as they interact with their peers, while preserving agent privacy. If this showed promise, we would explore development of a higher-level policy to decide whether to share model parameters or encoded data within a fixed communication budget of bytes.

Improving Performance through Multi-Step Reputation

Our implementation currently uses only ”1-step” reputation, where agents consider how each of their models worked on each of their datasets. It might, however, be useful to leverage the results of past interactions - such as an agents average accuracy on previous peers. In this case, if an agent tends to do better in past correspondence than their current peer, they may prefer higher β values.

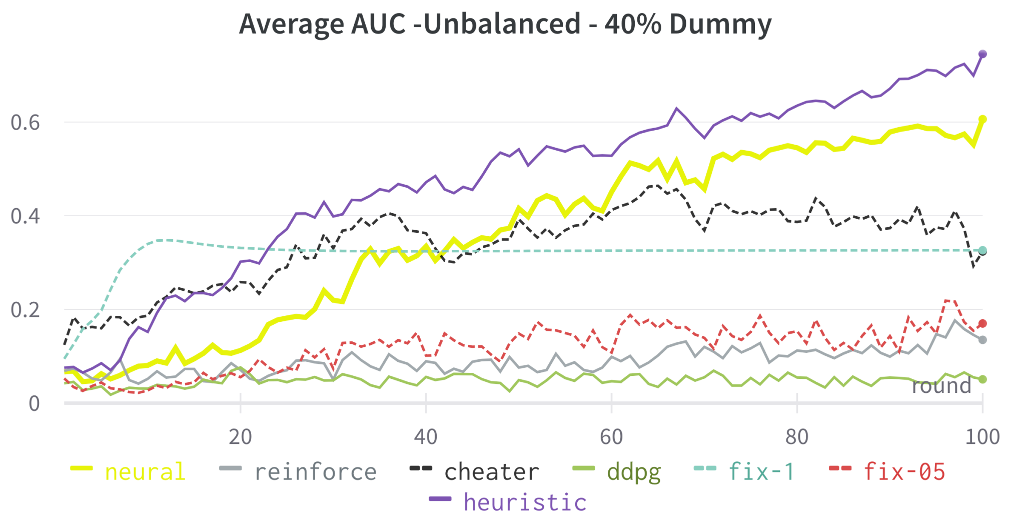


Figure 5: Unbalanced dataset, 40% NL agent

Feedback Responses

Why not grid search or evolutionary approach?

Grid search is only possible for the cheating policy - otherwise, it is not possible to identify the optimal beta on the global dataset. An evolutionary approach would require far more resources, as the system would need to maintain and train a large population of agents for a substantial period of time. Running a few cheating policies and imitating their behavior is much more practical.

What exactly are the constraints in transmitting information? The number of bits or how many other agents you can communicate with?

Agents first communicate their model and their local accuracy. Then, they communicate how well their peer's model performs on their local data. Thus, the communication budget is simply a model and 2 real numbers. Each agent only speaks to one other peer in each communication round.

What is the motivation/rationale behind the ”some NL” setting?

We want to examine a setting where naively combining with any other peers would be suboptimal. By examining this case, we demonstrate a resilience to less extreme cases where this principle applies - such as a new agent entering the system with a randomized model, or some agent has a naive adversarial policy. In the first case, the new agent would likely learn rapidly and mitigate their damage, but examining our fixed case demonstrates resilience to these sorts dynamics even in the worst case.

Conclusion

Gossip learning is a promising framework for distributed learning - it is both fully decentralized and globally asynchronous, unlike existing approaches. In addition, it preserves at least the same guarantees as centralized or federated learning regarding bandwidth and privacy concerns. In this work, we explore whether dynamic policies for incorporating peer models can improve resilience to unbalanced situations - such as those where agent's local data distributions are unbalanced or some agents have low-quality models.

We define and implement settings where agents have limited amount of available potentially unbalanced local data, in addition to potentially non-learning peers. In these settings, agents have to intelligently determine the combination factor when they communicate with different agents, possibly based on their relative performance and data distribution.

We frame the selection of a combination factor as a game, and train agents using both reinforcement learning and imitation learning methods. We derive 4 policies: REINFORCE, DDPG, imitation-based heuristic, and imitation-based neural network. We test them in comparison fixed combination factors of $\beta = .5$ and $\beta = 1$ as baseline policies. Note that the former is equivalent to existing gossip learning protocols. In all settings, regardless of data homogeneity or the degree of non-learning agents, our imitation-based policies match or exceed the performance of both baseline and RL-based policies. Our approaches demonstrate the importance of dynamically selecting combination factors for pairwise interactions in the gossip learning setting. We hope this work can serve as a step stone for further investigation into the problem.

References

[1] Boyd, S.; Ghosh, A.; Prabhakar, B.; and Shah, D. 2006. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6): 2508–2530.

[2] Carlini, N.; and Wagner, D. 2017. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, 39–57.

[3] Che, C.; Li, X.; Chen, C.; He, X.; and Zheng, Z. 2022. A Decentralized Federated Learning Framework via Committee Mechanism with Convergence Guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 1–18.

[4] Chen, D.; Zhou, B.; Koltun, V.; and Krähenbühl, P. 2019. Learning by Cheating.

[5] Goodfellow, I.; Shlens, J.; and Szegedy, C. 2014. Explaining and Harnessing Adversarial Examples. *arXiv 1412.6572*.

[6] Hegedundefineds, I.; Berta, A.; Kocsis, L.; Benczúr, A. A.; and Jelasity, M. 2016. Robust Decentralized Low-Rank Matrix Decomposition. 7(4).

[7] Hegedűs, I.; Danner, G.; and Jelasity, M. 2019. Gossip Learning as a Decentralized Alternative to Federated Learning. In Pereira, J.; and Ricci, L., eds., *Distributed Applications and Interoperable Systems*, 74–90. Cham: Springer International Publishing. ISBN 978-3-030-22496-7.

[8] Khosravi, A.; and Kavian, Y. S. 2017. Broadcast Gossip Ratio Consensus: Asynchronous Distributed Averaging in Strongly Connected Networks. *IEEE Transactions on Signal Processing*, 65(1): 119–129.

[9] Konečnỳ, J.; McMahan, H. B.; Yu, F. X.; Richtárik, P.; Suresh, A. T.; and Bacon, D. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

[10] Li, C.; Li, G.; and Varshney, P. K. 2022. Decentralized Federated Learning via Mutual Knowledge Transfer. *IEEE Internet of Things Journal*, 9(2): 1136–1147.

[11] Li, Q.; Wen, Z.; Wu, Z.; Hu, S.; Wang, N.; Li, Y.; Liu, X.; and He, B. 2021. A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection. *IEEE Transactions on Knowledge and Data Engineering*, 1–1.

[12] Lian, X.; Zhang, C.; Zhang, H.; Hsieh, C.-J.; Zhang, W.; and Liu, J. 2017. Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781510860964.

[13] Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning.

[14] Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Harley, T.; Lillicrap, T. P.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous Methods for Deep Reinforcement Learning. JMLR.org.

[15] Nguyen, A.; Yosinski, J.; and Clune, J. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 427–436.

[16] Onoszko, N.; Karlsson, G.; Mogren, O.; and Zec, E. L. 2021. Decentralized federated learning of deep neural networks on non-iid data.

[17] Ormándi, R.; Hegedűs, I.; and Jelasity, M. 2013. Gossip learning with linear models on fully distributed data. *Concurrency and Computation: Practice and Experience*, 25(4): 556–571.

[18] Tang, H.; Gan, S.; Zhang, C.; Zhang, T.; and Liu, J. 2018. Communication Compression for Decentralized Training. Red Hook, NY, USA: Curran Associates Inc.

[19] Tang, Z.; Shi, S.; and Chu, X. 2020. Communication-Efficient Decentralized Learning with Sparsification and Adaptive Peer Selection. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 1207–1208.

[20] Zhang, J.; Kim, J.; O'Donoghue, B.; and Boyd, S. 2020. Sample Efficient Reinforcement Learning with REINFORCE.

Appendix

Rewards for RL Methods (REINFORCE/DDPG)

The target reward for each agent $a \in A$ is a mixture of local accuracy and accuracy on their peers. Local reward is simply their own accuracy on their local dataset, which is in $[0, 1]$. Let $Acc_x(y)$ evaluate the accuracy of the y's model on x's dataset.

$$r_L(a) = Acc_a(a)$$

where r_L stands for local reward. Peer/Feedback reward is a function of how well an agent's model historically performed on each peer dataset. Suppose that the peers of agent a is $p_1, p_2, \ldots, p_{|P_a|} \in P_a$. To incorporate the fact that feedback reward (accuracy of model on peer dataset) will be gradually outdated as local model is updated, we maintain a "age" of the score of each peer reward, which is relative to the age of local model. The age of each peer is incremented each time the local model is updated. For the peer p_k that has just interacted with a, its age $Age_a(p_k)$ is reset to 1 since its reward corresponds to the current model. The numerical reward is a value between 0 and 1, reflecting an *weighted average* of these performances weighted *exponentially by the age of that metric*.

$$r_F(a) = \frac{w_{P_a}(a)}{q_{P_a}(a)}$$

$$w_{P_a}(a) = \Sigma_{p \in P_a} Acc_p(a) \times \sigma^{Age_a(p)}$$

$$q_{P_a}(a) = \Sigma_{p \in P_a} \sigma^{Age_a(p)}$$

where r_F stands for feedback reward and σ is the age discount factor. The total reward, as used during online training, is given below.

$$r(a) = r_L(a) + r_F(a)$$

Suboptimal Cheater

In figure 5, we observe the suboptimal performance of the cheater policy. We note that this is possible, mainly due to the fact that the cheater only considers the immediate accuracy after it picks a β value and combine with peer model. In this case, it may be beneficial to combine with a non-learning model, whose parameters are randomly initialized ans fixed. However, this may move the parameters of the resulting model to a very different location in the parameter space and result in large gradients when performing local learning on local dataset. Unusual change and instability can both occur.

Since the cheater can be viewed as a grid search based on immediate accuracy, doing naive grid search in our problem setting would not increase the performance. We hypothesize that doing multiple look-ahead and taking local learning steps into account may mitigate this problem, but it may require lifting some constrains to allow keeping multiple peer models or fixing communication peers for more rounds.