Random Walking Spiders for Decentralized Learning at Edge Networks

BY

FEDERICO GHIGLIONE
B.S, Politecnico di Torino, Turin, Italy, 2022

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2024

Chicago, Illinois

ACKNOWLEDGMENTS

I wish to thank all the committee members for their support, especially my two advisors, Professor Koyuncu and Professor Eros Gian Alessandro Pasero. Professor Koyuncu's help during my whole research was fundamental, answering all my questions; Professor Pasero's availability, despite the geographical distance and time difference, was outstanding. I want to show my appreciation to my family for their support, which has been essential for my college path because their guidance has been essential for my academic achievements. I also want to thank every member of my family, my mother, Claudia, and my father, Marco, for their dedication during both challenging and favorable times. Their support during tough times like the pandemic, when I had to study from home, was essential. I want to extend this appreciation to my brother, Edoardo, for his support and companionship during this challenging period. Spending time with him in a lot of activities like playing pool, watching sports games, and traveling around the world helped me stay motivated and focused on my studies. I want to thank also my friends, whose support has been invaluable throughout my college journey. I would particularly like to thank Emanuele G., known as "Lello," who was my roommate for four years during my college years in Italy. Like a brother, Lello became a crucial figure in my college journey, and I cannot thank him enough for his support. We share numerous memories that are too many to recount, but I must highlight the nights spent at "le Panche," "il sotto," and "il forno" with our friends, which were some of the best moments of my college life. Our "Thursday night before the Friday morning" tradition remains engraved in my memory. I want to thank the other two people who have always been with Lello and me during my last two years in Italy: Lorenzo G. N., also known as "Lonzo," and Massimiliano G., also known as "Max." We spent a lot of time together and even named our group "Mucchione." Some of our traditions included the "Fagiolata" and watching the Champions League with our friends "Chaquito" and "Tuono." I want to express my sincere appreciation to a dear friend of mine, Lucio F. During my final years at college in Italy, he was an unwavering source of support and guidance. Along with "Lello," he was akin to a brother to me, and I cherish the countless beautiful memories we shared. Our get-togethers were always characterized by hilarity; the wacky events that would occur during our time together never failed to elicit laughter for days on end. I am deeply grateful for the bond we share and its positive impact on my life. I also had the pleasure of meeting Michele G. during my college years. Although he bears several surnames, my favorite among them is "Tarallino". Our friendship was marked by our mutual love for "Orecchiette," socializing with our other companions, and a memorable week-long holiday in Croatia with some of our other friends. It was undoubtedly one of the most enjoyable vacations I have ever had. After them, I want to thank all my other friends that I met in Torino while I was studying there; the most important ones are Catia S. G., Angelo G., Alessandra F., Francesco P. C. known as "enf", Michele F., Marco R. G., Simone C., Antonio G., Giuseppe G. known as "Il sindaco", Luca G., Ludovica M., Mariateresa P., Allegra R., Samuele F. The following message expresses my appreciation to a group of friends from my hometown who have been fundamental in my personal development and growth. I extend thanks to Alessandro B., a dear friend since high school, Filippo C. (also known as "Coschiazza"), who was my personal driver when I was without a driver's license, Matteo B. (known as "Testone"), who shares my passion for football and the Argentina national team, and Lionel Messi, who has been a significant source of inspiration to me. I also express gratitude towards Edoardo C. G. (also known as "nino"), with whom I shared several vacations, Sofia Z., who has been a gracious host and invited us to celebrate her birthday and spend the day of "Ferragosto" at her home in the mountains, Alberto P. (also known as "Pu"), Simone C., and Elisa C., who traveled to visit me in Chicago from Italy while I was studying there. I also want to thank the wonderful people I met during my year-long stay in Chicago. I would like to begin by thanking my three roommates - Francesca F. (also known as Francina), Giuseppe G. (also known as Peppe), and Mattia C. (also known as Il Coscia) - for their unwavering support throughout this journey in the United States. We shared numerous unforgettable experiences together, such as eating the famous deep-dish pizza and watching movies at the cinema.

In addition, I would like to thank the other friends I made in Chicago, including Francesco V. (also known as Falco), Nicolò M., Lorenzo P., Martina C., Alice M., and Sara D. I cherish the countless fun-filled memories we created, such as movie nights, themed parties, poker nights, and much more. I want to extend my sincere thanks to all the other friends I've encountered here in Chicago., including Alessandro M., Riccardo S., Marco B., Simone M., Filippo D., Calliope, Davide M., Giuseppe L., Simone D., Giorgia, Matteo N., Andrea P., Carlo T., Gabriele B., Leonardo C., Luca R., Pietro M., and Ali. Once again, I want to thank everyone who made my stay in Chicago memorable.

AB

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

FNN	Feedforward Neural Network
CNN	Convolutional Neural Network
RWL	Random Walk Learning
RWS	Random Walking Spiders

SUMMARY

In decentralized learning, a branch of machine learning, data is distributed across several nodes. With this approach, each node has its own dataset, usually a fraction of the whole dataset, and it can train a model with its own data without the need to communicate with the other nodes. One of the most popular algorithms used in this field is Random Walk Learning (RWL).In the Random Walk with Learning (RWL) approach, we dispatch a solitary model to a node chosen at random from the network. This node is responsible for training the model for a predetermined number of epochs, often just one, before forwarding it to a neighboring node, also selected at random. The receiving node then undertakes the same procedure. During the learning process, the model is capable of acquiring knowledge from diverse nodes, but it faces some challenges, such as scenarios where the data distribution among the nodes is heterogenous, leading to accuracy issues because the accuracy could depend on the path chosen from the first node to the last one.

A new algorithm called Random Walking Spiders(RWS) has been introduced to solve these issues. The main difference between RWL and RWS is that in RWS, we have more models trained simultaneously instead of just one, like in RWL. Each model is initially sent to a random node and is trained with the local dataset, like in RWL, for a certain number of epochs (usually one epoch). After the nodes have trained the model assigned to them, all the models are sent to the same node, and an average of the model parameters is performed there, so every model becomes equal to each other, and then each model is sent to a new node chosen randomly, and the process repeats itself. The idea behind this new algorithm is to learn from more nodes simultaneously instead of just one at a time, as in RWL.

The RWS algorithm can be implemented in two different variants: non-adaptive RWS and adaptive RWS. The non-adaptive RWS is the process described before, where the models are trained in different nodes, then averaged, and passed to other nodes. The adaptive-RWS is an update of the non-adaptive case where we have just one model at the beginning since it's enough in the first epochs, while after certain conditions are met, it is split into more models.

The results show the superiority of both the non-adaptive and adaptive RWS over the RWL algorithm in terms of accuracy. While the non-adaptive RWS is better in the long term, the adaptive RWS is better also in short-medium scenarios. This performance improvement makes this algorithm a significant advancement in the decentralized learning field.

CHAPTER 1

INTRODUCTION

The term machine learning is used to identify the process where a machine processes some input data(this process is called the training phase) to predict the outputs of data that were not part of the training dataset. There are different types of machine learning algorithms, and one of the most popular is the one called artificial neural networks. These networks are modeled on the structure of the human neuron and can be used for different purposes, such as classification, clustering, and predictions. The model's training can be accomplished using different methods; three of the most used are supervised learning, unsupervised learning, and reinforcement learning

1.1 Types of learning algorithm

In supervised learning, the network parameters (also called weights) are modified using the samples present in the dataset, called training samples. Every sample comprises the input pattern (it could be an image if we are doing an image classification task) and its corresponding output. This technique is one of the most used because the network can be trained to make good predictions on data that are not part of the training dataset by utilizing a set of training samples with known input-output. [1, 2]

In unsupervised learning, the aim is to discover patterns and structures in data without having corresponding outputs, in contrast to the approach taken with supervised learning.

Without the presence of explicit labels (supervision), this technique tries to reveal hidden insights or grouping within the data to learn from the provided data and make predictions about new, unseen samples. This approach is particularly useful when the outputs of the training data are unavailable or when the dataset is too large to be labeled. Unsupervised learning is mostly used in data science, where we want to discover insights and patterns in large and complex datasets. [3–5]

In Reinforcement Learning there is an agent that can make decisions by interacting with an environment to achieve specific goals. Unlike supervised learning, it doesn't learn from a set of data but from rewards or penalties based on the actions taken by the agent. Reinforcement learning algorithms learn a policy that maximizes cumulative rewards over time. [6, 7]

In the course of this study, we will have to test our new algorithm with a classification task where the dataset is composed of data in the input-output format. Therefore, the obvious choice is to use a supervised learning algorithm to train our models.

1.2 Network architechtures

There are multiple types of neural networks, each one specialized in accomplishing specific tasks and objectives and has its own architecture designed to solve those specific tasks. It is very important to understand the specific features of every type of neural network in order to choose the best one when facing a new problem. The types we will focus on are the Feedforward Neural Networks and the Convolutional Neural Networks. The Feedforward Neural Networks are made by layers of interconnected neurons where data moves in a unidirectional manner, starting from the input and proceeding to the output. They are mainly employed for classification and regression purposes. FNNs are characterized by the number of layers, usually divided into 3 types: the input layer, the hidden layers(can be more than one), and the output layer. [8, 9]

Convolutional Neural Networks are specifically engineered to process data structured in a grid pattern, like that found in images. These networks utilize convolutional layers to identify and extract spatial patterns within the input data. Common applications of CNNs include image recognition, object detection, and categorizing images. [10–12]

Since the dataset that we will use to test our algorithm is composed of images, we will use a CNN with an FNN at the end to classify the images. To better understand this process, we will now focus on how to train a CNN model.

1.3 Training a CNN

CNNs are a popular tool for image classification tasks. A dataset of images of size nxn pixels is typically utilized to train a CNN for such a task. Each image in the dataset is assigned to a specific class, corresponding to a particular type of image (e.g., dog, cat). The ultimate objective is to develop a CNN that can accurately classify new and unseen images to one of the pre-defined classes with the highest possible degree of accuracy.

Therefore, selecting an appropriate dataset with appropriate class labels is essential to ensure that the CNN is trained effectively. The CNN architecture must also be carefully designed and trained using an appropriate algorithm to maximize its performance. Once trained, the CNN can be used to classify new and unseen images.

The learning process is usually designed in this way:

1. Data preprocessing: Before training a CNN, it is necessary to preprocess the data. First of all, we want all the images to have the same size, so we have to resize every image to a fixed size; after that process, the pixel values are usually normalized to a predetermined range like [0, 1] or [-1, 1]. Following the processing of all images, we categorize the dataset into three distinct sections: training, validation, and test set.

2. Initialization: The CNN parameters, including both weights and biases of the convolutional layers, are initialized with techniques such as random initialization, Xavier initialization, or He initialization. [13, 14]

3. Forward pass: In the forward pass, the input data is transformed into feature representations by applying convolutions, activations, and pooling sequentially. The final layer produces the predicted class probabilities or values. This is a crucial component of the training process.

4. Loss computation: After the forward pass, the CNN proceeds to compare its output with the true labels given together with the input, utilizing a loss function to quantify how the predicted output is close to the correct one. Some examples of loss functions employed in classification tasks are cross-entropy loss[15, 16], softmax loss[17], and hinge loss[18], while the mean square error (MSE) [19, 20] is used in regression tasks. Since our goal is to solve a classification task, we decided to use the cross-entropy loss.

5. Backpropagation: Backpropagation is a crucial process in training a CNN, where, with respect to the parameters of the CNN, the gradients of the loss function are computed. In the backpropagation process, the error is spread regressively through the network, layer by

layer, using the chain rule of calculus. For each parameter, the gradients are computed, to show how a small change in that parameter would change the loss. This process helps the neural network to iteratively optimize the parameters in order to improve its performance. [21, 22]

6. Parameter Update (Optimization): Upon completion of gradient computation, the CNN parameters are updated using an optimization algorithm. Some examples of optimization algorithms are Stochastic Gradient Descent (SGD)[23], Adam [24], RMSprop [25], and AdaGrad [26]. These algorithms modify parameters to reduce the loss function, aiming to find the best set of parameters that decrease prediction error. In this work, we decided to use the Adam optimizer.
7. Iteration: The process from steps 3 to 6 is repeated iteratively for a specific number of times, called epochs, or until the model reaches a convergence criterion. In each iteration, the CNN learns to approach the true correlation that connects the input data with the output labels, gradually improving its performance on the training dataset.
8. Validation and Testing: During the training process of a CNN, it is important to monitor the generalization performance and avoid overfitting [27]. To achieve this, the performance of the CNN is periodically evaluated on a separate validation dataset. Once the training is completed, the final performance of the CNN is assessed on an independent test dataset to determine its performance on new and unseen data.

At the conclusion of the training phase, the main goal is to have a model that can accurately classify new samples into one of the classes featured in the dataset if it is a member of any.

However, we made the assumption that we would have access to the whole dataset, but what if such a dataset is distributed across multiple clients? In this case, we are in a decentralized learning scenario.

1.4 Decentralized Learning principles

Decentralized learning [28] is a machine learning field in which the training process is spread out over various devices or nodes., often without sharing the underlying data. Decentralized learning is an innovative approach that stands on several key principles. First and foremost, it involves the distribution of data across various nodes, which can range from personal devices such as smartphones and IoT devices to servers. The core idea behind this distribution is not only to utilize the computational power available at the edge but also to significantly enhance privacy. By ensuring that data remains on its original node and isn't shared with others or a central repository, decentralized learning places a high value on preserving the privacy of individual data. [29] Another cornerstone of decentralized learning is its support for collaborative efforts. It achieves this through a unique model where updates or gradients calculated locally are then aggregated to improve a global model. This process happens without the need to share sensitive or private data directly, leveraging either a central server that synthesizes these updates or a consensus mechanism among the nodes. Moreover, decentralized learning heavily relies on edge computing [30]. This means computations are performed as close to the data source as possible, which not only enhances processing speeds but also significantly reduces latency and the demand on bandwidth since there's less need to send data back and forth to a central server. In terms of safety and security, this learning model offers robust protection. Avoiding a centralized point where data is stored or processed makes it much tougher for any single point of failure or attack to compromise the system. This distributed nature makes the entire framework more resilient to adversarial attacks. Decentralized learning is also designed to scale efficiently. Because the computational efforts are spread out across numerous devices or nodes, it allows for processing large datasets without the need for extensive computing power centrally. This scalable nature is vital for handling the increasing volumes of data generated and used today. Finally, an important benefit of decentralized learning is its ability to aid organizations in meeting regulatory compliance, particularly with regard to data protection regulations like GDPR [31]. By minimizing the central collection and storage of data, organizations can better align with regulatory demands, ensuring that sensitive information is managed appropriately and reducing the risk of non-compliance.

Decentralized learning presents itself as a privacy-preserving and collaborative approach that facilitates the development of robust and scalable models while upholding individual privacy and data sovereignty. However, it is necessary to acknowledge that this approach has limitations. Despite the advantages of decentralized learning, it presents certain challenges that must be addressed. It is essential to understand these limitations to ensure that this approach is implemented effectively and efficiently.

1.5 Decentralized Learning limitations

In our previous discussion, we delved into the core principles of decentralized learning and its potential advantages for machine learning. However, it's essential to acknowledge that this methodology comes with its own set of challenges and limitations. One significant challenge involves the communication overhead required for model aggregation or synchronization between the central server and participating nodes. This can lead to increased latency and reduced efficiency, particularly in scenarios involving numerous nodes or under poor network conditions. [32]

Another hurdle is the heterogeneity of the nodes due to the diverse computing capabilities, network connectivity, and data distributions of the nodes. Ensuring compatibility and fairness across such a wide range of devices and data sources poses a considerable challenge. Linked to this is the problem of data imbalance, where an uneven distribution of data among nodes can lead to nodes with scant data contributing less effectively to the model training, potentially resulting in biased or less accurate global models.

Security concerns also emerge as a prominent issue, especially the risk of model poisoning attacks by malicious nodes intending to undermine the overall model's integrity. Thus, guaranteeing the security and trustworthiness of participating nodes is crucial. In addition, as theoretically scalable as decentralized learning might be, the reality of managing numerous nodes and coordinating model updates at scale introduces complex scalability challenges. This complexity makes ensuring efficient aggregation and synchronization mechanisms increasingly difficult. [33]

Developing efficient algorithms for decentralized learning also presents a notable challenge. These algorithms need to strike a balance between privacy, communication overhead, convergence speed, and model quality, which is far from trivial. Such algorithms must be carefully designed to perform well across diverse datasets and computational environments.

Lastly, decentralized learning can face regulatory compliance challenges, particularly in highly regulated industries like healthcare or finance, which are governed by strict data protection laws and regulations. Navigating these regulations while leveraging the benefits of decentralized approaches can be complex. These challenges underscore the need for continuous exploration and improvement in the field of decentralized learning to fully harness its potential. [34]

Ongoing research and development initiatives serve a crucial role in addressing the limitations associated with decentralized learning. Therefore, it is crucial to focus on key areas, including communication protocols, privacy-preserving techniques, security mechanisms, and algorithmic innovations. These efforts aim to improve the robustness, efficiency, and applicability of decentralized learning across various domains. As a result, the development of innovative and effective strategies in these areas can significantly contribute to advancing the field of decentralized learning.

Our main goal is to create a new algorithm that can obtain a higher accuracy with respect to other decentralized learning algorithms.

CHAPTER 2

PREVIOUS WORK

2.0.1 Federated Learning

Federated Learning (FL) [35, 36] is a method used in the decentralized learning field. The centralized FL is the most common type among all the FL approaches. Despite the name, also the Centralized FL refers to a context where data are distributed across multiple nodes; the name is due to the fact that we have a main node that has the task of managing the other nodes, controlling the steps of the learning algorithm such as moving a model from one node to another. This approach has some advantages and disadvantages, so we have to explore all of its features before choosing to use it.

Centralized Federated Learning offers a series of benefits that address core concerns around data privacy, efficiency, and utilization. One of its key advantages is the promise of enhanced privacy and security. By allowing data to remain on local devices and only sharing model updates, Federated Learning greatly diminishes the risk of sensitive data exposure. This feature is crucial for industries that handle confidential information, ensuring they can train models without compromising data security. [37]

Another significant benefit is the improved utilization of data. Federated Learning enables the leverage of decentralized data sources, which may be siloed or unreachable due to privacy laws or regulatory boundaries. This approach widens the scope and diversity of data accessible for model training, offering a more robust dataset than traditional methods.

Additionally, the model boasts efficiency in data transmission. By transmitting model updates rather than the raw data itself, there's a substantial reduction in bandwidth requirements. This is particularly advantageous in settings with limited network resources, ensuring smooth and cost-effective operations.

Lastly, localized training underpins the customization and relevance of models. Since the training occurs on the devices generating the data, the models inherently reflect local preferences and behaviors, making them highly tailored to the specific needs and nuances of each data environment.

The disadvantages of Federated Learning [38] include various factors crucial to the operation and efficiency of federated networks. One significant challenge is communication costs, primarily due to the imperative to keep data local on devices to address privacy concerns, making wide-scale data consolidation impractical. Federated networks must manage a myriad of devices, each differing in storage, computational power, and connectivity. This diversity creates an environment where only a limited subset of devices might actively participate at any given time due to constraints like battery power or network access, further compounded by the possibility of devices becoming unreliable due to connectivity issues or power failures.

Another aspect to consider is the variability in data among the network's devices, which can be quite uneven, influencing both the amount and type of data collected. This effect is evident in scenarios requiring predictions based on data inputs that vary significantly, such as in language processing tasks on mobile phones. Such diversity poses challenges in achieving uniform model accuracy across the network.

Privacy preservation remains a towering concern within federated learning applications, primarily because the methodology involves transmitting model updates rather than raw data. Despite this approach minimizing direct exposure, the indirect sharing of information through model updates can still potentially reveal sensitive data either to a middleman or the central server controlling the operation. Although advancements like secure multiparty computation and differential privacy are in development to fortify privacy measures, they often lead to compromises in model effectiveness or operational efficiency.

Lastly, the risk of central failure is an inherent disadvantage. Despite the decentralized data approach of federated learning, the presence of a central node critical to managing the network's operations introduces a single point of failure. Any malfunction at this node could disrupt the entire learning algorithm, leading to significant operational setbacks for the federated network.

To conclude, Federated Learning (FL) offers enhanced privacy and efficiency by training models on decentralized data sources while keeping data local. However, challenges such as communication costs, heterogeneous device capabilities, and privacy concerns remain significant. The reliance on a central coordinating node introduces potential single points of failure. Addressing these complexities is essential for maximizing FL's benefits while mitigating operational risks and ensuring data security across diverse applications.

2.0.2 Gossip Algorithms

Gossip algorithms [39–49], also known as epidemic algorithms in certain contexts, are decentralized methods that are employed for information dissemination in distributed systems and networks. These algorithms are inspired by the way rumors or information proliferate in social networks, and they allow nodes within a network to communicate with each other randomly. This random communication pattern facilitates the propagation of information throughout the network in an organic and efficient manner.

Within the framework of Gossip Algorithms, the process begins with each participant, known as a node or agent, being equipped with particular information or a value. As the algorithm unfolds, these nodes engage in a randomized exchange of information during each cycle. This is achieved by having one node select another at random to share the data it holds. This strategic sharing ensures that with each round, every participant is increasingly informed about the broader picture within the network. Following the exchange, nodes assimilate the newly acquired data into their existing information, gradually leading to a unified understanding across the network. This methodology highlights the power of distributed communication and the iterative refinement of information towards consensus.

Gossip algorithms stand out for their exceptional scalability, achieved through a decentralized approach that eliminates the need for a central controller. This allows them to seamlessly adapt and grow within networks of varying sizes, from small clusters to large-scale distributed systems and peer-to-peer networks. Their flexibility is unparalleled, making them versatile across different network types. Furthermore, these algorithms are bolstered by robust fault tolerance, courtesy of their decentralized nature. This ensures that even if some nodes fail or disconnect, the dissemination of information can continue through alternative pathways, thus maintaining the integrity and flow of data across the network.

Notably, gossip protocols are highly adaptable and flexible, perfectly suited for dynamic environments where nodes frequently change, join, or leave. This ability to adjust to ongoing changes in network topology underscores their effectiveness in such settings.

However, these algorithms are not without their drawbacks. Their convergence speed can be relatively slow in certain network conditions due to their reliance on random communication, which may necessitate multiple iterations for complete information dissemination across all nodes. This aspect underscores the need for careful parameter adjustments to achieve efficient network communication. Additionally, the very nature of their communication pattern can lead to redundant data transmission. This redundancy might overwhelm the network by consuming excessive bandwidth and computational resources. Lastly, despite their decentralized design, gossip algorithms cannot always guarantee successful information spread to every node, especially in large or rapidly changing networks. This could result in delayed information propagation or some nodes being completely bypassed.

In summary, while gossip algorithms offer significant advantages in scalability, fault tolerance, and adaptability, they also come with challenges such as slower convergence speeds, potential resource intensiveness, and no absolute guarantees on information dissemination.

Gossip algorithms provide a decentralized, scalable, and fault-tolerant way to distribute information in distributed systems and networks. They offer many benefits, such as adaptability and resilience, but can also pose challenges, such as possible delays in convergence and resource-intensive operations. The effectiveness and suitability of gossip algorithms depend largely on the specific requirements, constraints, and characteristics of the network environment in which they are deployed.

2.0.3 Random Walk Learning

Random Walk Learning (RWL) [50–53] is a novel approach in the machine learning landscape, where the learning process is analogous to a random walk. This methodology is especially prominent in decentralized learning environments, where it leverages the distributed nature of data across various nodes or agents. RWL embodies the principle of incremental learning from multiple data sources in a network without a centralized database.

Random Walk Learning (RWL) brings a novel edge to the field with its decentralized nature, effortlessly fitting into scenarios where data centralization is hindered by privacy, legalities, or sheer volume. Its seamless scalability makes it a powerhouse for extensive applications, where it can distribute its learning process across countless nodes without a hitch. Privacy doesn't take a backseat either; RWL maintains an ironclad grip on data security by learning from data without the need to relocate it—a nod to the principles of federated learning. Additionally, RWL stands out for its adaptability, gracefully accommodating data shifts or the introduction of fresh data, a vital trait for thriving in dynamic data landscapes.

However, RWL's journey is not devoid of challenges. Its hallmark, the random walk algorithm, while unique, can decelerate the convergence process, as the lack of an optimized learning path leads to variability. The architecture's dependency on network topology further complicates matters; a weakly connected network can severely dampen the learning efficacy. There's also the issue of data skewness and bias—frequent visits to certain nodes or dominantly influential data could distort the model, presenting hurdles in managing non-Uniformly distributed data. Alongside, RWL's implementation complexity and the overhead of managing model updates across a broad network can't be overlooked. And akin to other decentralized learning frameworks, RWL navigates through murky waters of security, where the threat of model tampering by malevolent nodes looms large, underscoring the complexity of safeguarding the learning process in an open ecosystem.

In summary, Random Walk Learning presents an innovative approach to learning in decentralized environments, offering scalability, privacy preservation, and adaptability benefits. However, the challenges associated with convergence efficiency, data skewness, and security need careful consideration and mitigation in practical applications. Balancing the pros and cons is essential when deploying RWL in real-world scenarios.

CHAPTER 3

RANDOM WALKING SPIDERS

This section presents a new algorithm called Random Walking Spiders, which has been developed to overcome existing approaches' limitations and improve the final model's accuracy. The introduction of this approach is intended to provide a more effective solution to the problems that have been encountered in the decentralized learning field. The Random Walking Spiders algorithm has been designed to address the challenges associated with traditional methodologies, incorporating unique features that make it a more robust and reliable alternative. Our discussion will delve into the technical aspects of this approach, highlighting its unique features and the potential it holds for advancing the field.

3.1 Network Model and Random Walk Learning

In network analysis, the structure is often illustrated as an undirected graph, denoted as $\mathcal{G}(V, E)$. In this representation, $V = [N] = \{1, \ldots, N\}$ denotes the nodes or vertices, while $E \subseteq V \times V$ symbolizes the links or communication pathways connecting these nodes [54].Imagine we have a dataset $X = \{x_1, \ldots, x_K\} \subset \mathbb{R}^d$ accompanied by a set of corresponding labels $D = \{d_1, \ldots, d_K\}$. This dataset is not centralized but rather distributed across the graph's nodes. Specifically, every node n can see only a subset of the data, denoted by $X_n \subset \{x_1, \ldots, x_K\}$, and the associated labels D_n. The goal for each node is to utilize its local data, X_n, D_n(where X_n is the input and D_n is the output), in the training process. The main goal is to adjust the neural network weights w to minimize a global loss function, which is computed as an average over all the data points in the dataset X.

To better understand, we recall the Random Walk Learning (RWL) concept, explaining how its training process works. RWL starts with the initialization of the model at a node, typically selected randomly. This node begins the learning process by training the model on its local dataset using a method known as stochastic gradient descent, carried out over several epochs. After this local training phase is completed, the model is transferred to a neighboring node. The selection of this next node is also random, though various strategies can be devised to determine which neighbor to choose. This training procedure is performed at a node, and then the model is passed to a randomly selected neighbor, which is repeated iteratively. The process continues this cycle of local training and model passing until a point is reached where the model weights no longer experience significant changes, indicating convergence.

This method facilitates the development of a distinct model by eliminating the necessity to gather all data in a single location, thus safeguarding the privacy and independence of every node in the network.

3.2 Random Walking Spiders (RWS): The Non-Adaptive Case

We now present a novel method, RWS, that builds upon the RWL concept to provide a more flexible and dynamic approach to distributed learning across a network graph. In its simplest form, RWS initializes M instances of a model on M distinct nodes $n_1, ..., n_M$, chosen randomly from a larger set of N nodes within a graph. This ensures an unbiased distribution of model instances across the network, and enables parallel processing across multiple nodes.

Like RWL, each node is responsible for training its assigned model using exclusively the data available to it. This localized training methodology ensures that each model instance is uniquely adapted to the data characteristics of its node, thereby enhancing model diversity. In our experimental setup, we have standardized the training duration to one epoch per node to maintain consistency and simplicity in the process evaluation.

An innovative aggregation strategy is employed following the local training phase. This strategy involves averaging the parameters of the locally trained models at a designated node among $n_1, ..., n_M$. To facilitate this, we presume that each node has access to a common random seed, enabling it to generate and share a sequence of random numbers predictably. This shared sequence ensures that all nodes are synchronized in based on their knowledge of which nodes are participating in the model initialization and aggregation phases. Thus, each node knows that they have to send their model (after the local training) to the node designated as the aggregator, determined to be Node $\min_{j \in \{1,...,M\}} n_j$, for the averaging process.

Upon completion of the averaging process at the aggregator node, the newly averaged model is distributed to the next set of nodes, $n'_1, ..., n'_M$, which were previously selected randomly from each node's neighbors. This sequential progression of training and aggregation phases enables a continuous flow of learning and model refinement across the network.

It is noteworthy that when the model initialization parameter M is set to 1, RWS simplifies to the RWL process. This versatility of RWS makes it a promising framework for distributed learning across various scales of network graphs.

To describe the operational dynamics of RWS, we use the analogy of a spider, with the nodes where training occurs representing the legs of a spider, and the aggregation node serving as the head. Once training on the 'legs' is complete and the averaging is done at the 'head', the spider 'moves' to a new set of positions. This randomness in node selection for training and aggregation gives the appearance of a spider moving unpredictively across the network graph, hence the naming of the method as Random Walking Spiders (RWS). This visual metaphor not only aids in understanding the process but also highlights the inherently dynamic and adaptive nature of RWS as it navigates the network graph, optimizing distributed learning through a combination of localized training and strategic aggregation. The algorithm is illustrated in Fig. Figure 1.

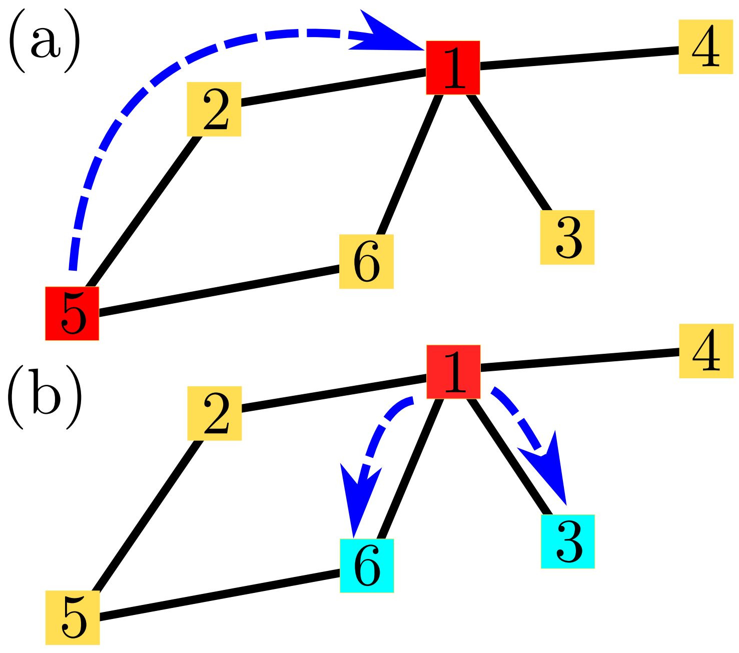

Figure 1: An illustration of the RWS algorithm for $M = 2$. In (a), the initialization for $n_1 = 5$ and $n_2 = 1$ is shown. Next neighbors of the nodes are assumed to be chosen as $n'_1 = 6$ and $n'_2 = 3$. After local training, Node 5 sends its model to Node 1 for averaging. In (b), the averaged model is forwarded to the Nodes 3 and 6 for further training.

Convergence time stands as a pivotal metric for assessing the efficacy of learning algorithms. In the context of graph-based learning algorithms, achieving convergence necessitates that each node within the graph is visited at least once. The absence of such a visitation precludes the model from assimilating the data associated with any unvisited nodes, thereby impeding the learning process. Consequently, the expected time, denoted by T, required for all nodes to be visited at least once, emerges as a key indicator of the learning algorithm's convergence speed. The relationship between the expected time T and the structural characteristics of the graph, "Specifically, the total count of nodes, represented as N, and the quantity of spider legs, represented as M.", is elucidated through the subsequent theorem:

Theorem 1 *Let V be a complete graph. The expected time to visit all nodes at least once satisfies $T \in \Theta(\frac{1}{M} N \log N)$.*

This theorem elucidates that employing M spider legs facilitates a training process that is M times faster than the traditional approach, which utilizes a singular leg as seen in ordinary random walk learning algorithms. This enhancement in processing speed, however, is not the sole advantage. The use of Random Walk Spiders (RWS) over Random Walk Learning (RWL) significantly bolsters model diversity, an attribute of paramount importance, especially when dealing with heterogeneous data sets.

In this way, it would appear that the RWS algorithm is M times faster than the RWL algorithm. However, we have to consider that when we train a single model in RWL for n epochs, we do nC number of computations where C denotes the number of computations needed to train a model for 1 epoch. When we train M models for n epochs, we do MnC number of computations, so it seems like there's no real improvement. Nevertheless, The extra advantage of RWS over RWL is the model diversity it provides, especially in the case of heterogeneous data. Note that RWL updates only a single version of a model. If a node alters this single version to a largely suboptimal direction (for example, due to a local data distribution that is significantly biased towards one class), then recovering from such an unfavorable local update may not be straightforward. On the other hand, RWS updates multiple copies of the model and then averages them out. Bad updates, if any, are smoothed over to an overall decent model improvement.

3.3 Random Walking Spiders (RWS): The Adaptive Case

In this section, we elucidate an advanced variation of the RWS algorithm, termed Adaptive RWS. This innovation was conceptualized to circumvent a specific limitation identified in the non-adaptive iteration of the algorithm.

A pivotal consideration in evaluating the efficacy of the RWL and RWS algorithms involves a nuanced comparison of their accuracies after a designated number of epochs. It is imperative to recognize that a direct comparison is not feasible due to the augmented computational effort in the RWS method, where computations are magnified by a factor of M. Consequently, the initial i epochs in the RWS algorithm are juxtaposed against the first iM epochs in the RWL algorithm for a balanced assessment.

A notable challenge emerges in the initial stages, where the accuracy disparity between RWL and RWS, despite equivalent computational exertion, might be pronounced. However, it is anticipated that RWS will demonstrate superior performance over an extended duration. To address this, the adaptive mechanism initiates with the RWL algorithm until a predefined accuracy criterion is satisfied, subsequently transitioning to the RWS algorithm. This strategy aims to match the early-epoch accuracy of RWL while achieving enhanced accuracy in the later epochs.

Delving into specifics, the adaptive version commences with a singular instantiation of the Model on a randomly selected node, n_i, from among the N nodes constituting the graph. This approach mirrors the non-adaptive variant, wherein the node subjects the model to training on its localized data for a predetermined number of epochs. In our experimental setup, this duration was standardized to one epoch per node. This phase parallels the RWL methodology, whereupon completion of training, the incumbent node selects the subsequent node for training. The selection process involves choosing one of the neighbors of the actual node, referred to as Node n_i', in a uniformly random manner. For every node n_i, one of its neighbors is selected in this way, for all i in the range $1, ..., N$.

The defining feature of adaptivity intervenes after each training epoch, i. The algorithm evaluates the discrepancy between the accuracy on the training set after i epochs of training (Tr_i)

and that after $j = i - K$ epochs of training (Tr_j), where K is a predetermined constant. Upon the difference, $\Delta Tr = Tr_i - Tr_j$, falling below a pre-specified threshold, τ, a bifurcation of the model is triggered.

This bifurcation entails the initialization of $M - 1$ additional instances of the model across $M - 1$ randomly chosen nodes, $n_1, ..., n_{M-1}$, from the N nodes of the graph. These models are endowed with identical parameters as the initial model, culminating in M congruent models. Subsequently, the algorithm reverts to the methodology of the non-adaptive variant, with each node independently training its model on localized data for a fixed epoch count. In our experimental framework, this was set to one epoch per node. Post-training, the models undergo an averaging process at one of the nodes, $n_1, ..., n_M$. The node responsible for averaging then disseminates the aggregated model to Nodes $n'_1, ..., n'_M$, thereby perpetuating the learning cycle. The following theorem elucidates the operational principle of this algorithm:

Theorem 2 *Let K be the depth of epochs to be checked, τ a threshold value and Tr_i the training accuracy after the i^{th} epoch of training, the split is performed if:*

$Tr_i - Tr_j < \tau$ *where* $j = i - K$

For our empirical investigation, constants were established at $K = 3$ and $\tau = 0.05$. This meticulous design and the adaptive strategy underscore our commitment to refining the algorithm's efficiency and accuracy, ensuring it remains at the forefront of distributed machine learning methodologies.

3.4 Numerical Results

In this part, we aim to meticulously detail the results derived from the experimental simulations conducted on two distinct algorithms. These experiments were methodically carried out utilizing the CIFAR10 and the Fashion-MNIST datasets, applying them to a standard CNN model framework. It is pertinent to articulate that the term 'epochs' within the context of our experiments refers to the total number of computations executed. Specifically, a single computation corresponds to the training of one model for one epoch.

For the purpose of these experiments, certain constants were established. Notably, the constants selected for the study were $K = 3$ and $\tau = 0.05$. These parameters were thoughtfully chosen based on their relevance and potential impact on the outcomes of the experiments. The inclusion of these constants serves to provide a structured framework within which the experimental results can be accurately assessed and interpreted.

The experimental design and its execution are fundamental to the integrity and reliability of the results obtained. As such, a meticulous approach was adopted in the planning and execution stages, ensuring that each step of the process was carefully considered and appropriately implemented. The aim of this rigorous methodology is to furnish clear, comprehensible, and actionable insights based on the data collected from the simulations.

This analysis not only contributes to the existing body of knowledge on the subject but also lays the groundwork for future research in this area. By presenting these findings, we hope to offer valuable perspectives and stimulate further discussion among researchers and practitioners in the field.

3.4.1 Non-adaptive RWS with Homogenous data distribution

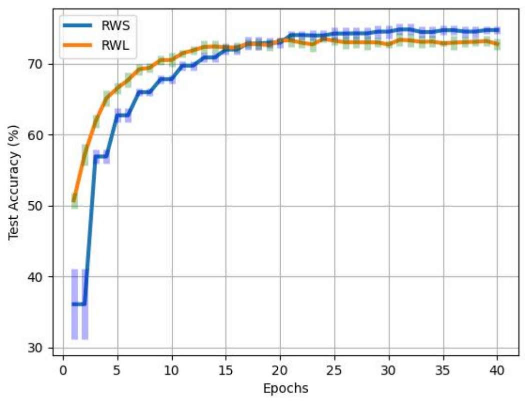

Figure 2: The result of the non-adaptive RWS algorithm with a homogenous data distribution among $N = 5$ nodes for $M = 2$ models using the CIFAR-10 dataset compared to the standard RWL.

The non-adaptive RWS algorithm was implemented on $M = 2$ models using a homogenous data distribution among $N = 5$ nodes. The results were consistent with our expectations, as the algorithm had lower accuracy than the RWL algorithm during the initial epochs. However, it eventually surpassed the RWL algorithm in terms of accuracy in the long run. Additionally, the standard deviation was found to be almost identical in both algorithms.

3.4.2 Adaptive RWS with Homogenous data distribution

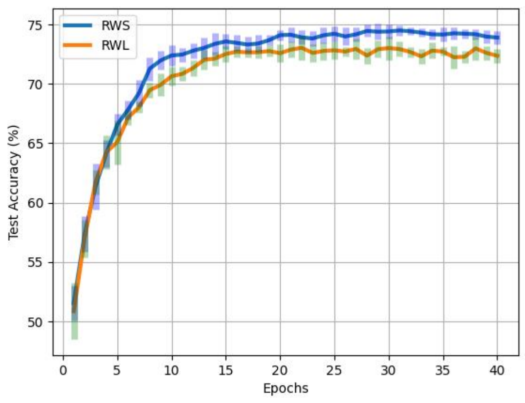

Figure 3: The result of the Adaptive RWS algorithm with a homogenous data distribution among $N = 5$ nodes for $M = 2$ models using the CIFAR-10 dataset compared to the standard RWL. $N = 5$ nodes

The results obtained from the adaptive RWS algorithm for $M = 2$ using a homogenous data distribution among $N = 5$ nodes were in accordance with the expectations. During the initial epochs, the accuracy was consistent with the RWL algorithm, which indicates a significant improvement with respect to the non-adaptive scenario. Furthermore, in the later epochs, the adaptive RWS algorithm outperformed the RWL algorithm in terms of accuracy. Upon analysis, it was determined that the adaptive RWS reaches superior accuracy compared to RWL in just

5 epochs, in contrast to the non-adaptive RWS, which necessitates 20 epochs to achieve similar results. Moreover, the standard deviation was almost the same as that of the RWL algorithm.

3.4.3 Adaptive RWS with Dirichlet distribution

In this particular subsection, the data are distributed among the nodes using a Dirichlet distribution.

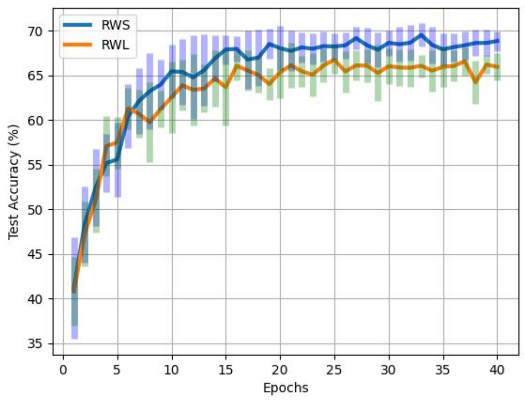

Figure 4: The result of the Adaptive RWS algorithm using a Dirichlet data distribution among $N = 5$ nodes for $M = 2$ using the CIFAR-10 dataset compared to the standard RWL.

The results obtained through the adaptive RWS algorithm utilizing the Dirichlet data distribution among $N = 5$ nodes with $M = 2$ models are akin to those observed with homogeneous data distribution. It is noteworthy, however, that the standard deviation is higher in the former case due to the data distribution. Additionally, comparing this figure with its predecessor reveals a slight increase in the difference between RWS and RWL. Moreover, the standard deviation was almost the same as that of the RWL algorithm.

3.4.4 Adaptive RWS with 2 classes per Node

In this case, we observe an extreme data distribution among the nodes. Our approach involved utilizing $N = \frac{1}{2}C$ nodes, where C represents how many classes the dataset has. Consequently, we assigned all the samples of two randomly selected classes to every node. As we are working with the CIFAR-10 dataset, which has 10 classes, we maintained a constant number of nodes, with N remaining at 5.

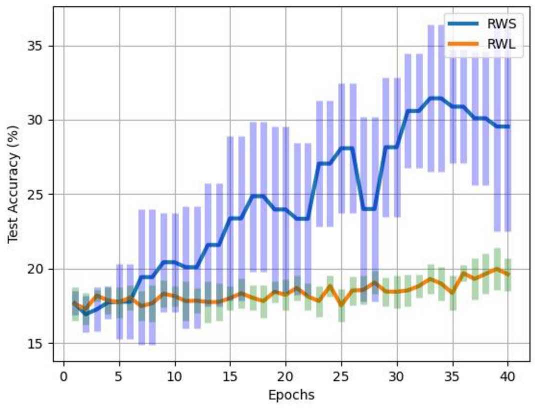

Figure 5: The result of the Adaptive RWS algorithm using 2 classes per node data distribution among $N = 5$ nodes for $M = 2$ using the CIFAR-10 dataset compared to the standard RWL.

The results of the study conducted on the Adaptive RWS algorithm utilized to process 2 classes per node data distribution for $M = 2$ indicate that an increase in the extremity of the distribution leads to a concomitant increase in the standard deviation. Although the high standard deviation may appear to be a concern, the RWS algorithm still outperforms the RWL algorithm in terms of accuracy.

3.4.5 Increasing the number of nodes

When increasing the number of nodes, we expect to need more models to have the best accuracy. This is due to the fact that we need more models to learn enough information with more nodes.

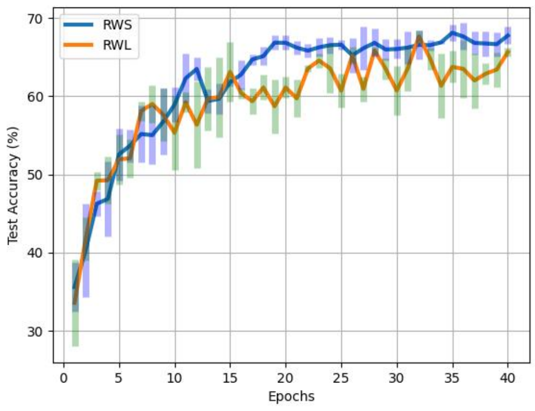

Figure 6: The result of the Adaptive RWS algorithm using a Dirichlet data distribution among $N = 10$ nodes for $M = 2$ using the CIFAR-10 dataset compared to the standard RWL.

As we can see from the figure, using 2 models with a higher number of nodes leads to an accuracy that is not clearly better, so we will now try with 3 models instead of 2.

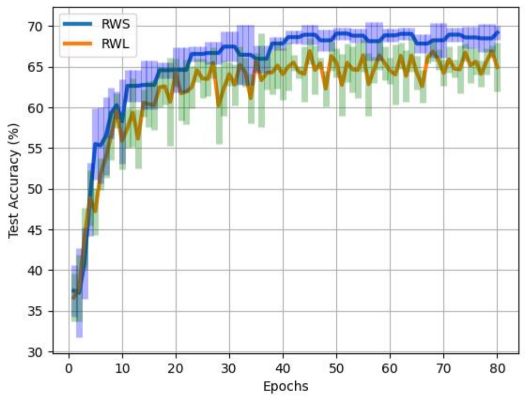

Figure 7: The result of the Adaptive RWS algorithm using a Dirichlet data distribution among $N = 10$ nodes for $M = 3$ using the CIFAR-10 dataset compared to the standard RWL.

From this plot, the improvement obtained using 3 models instead of 2 is clear; in this case, the accuracy of the RWS algorithm is always higher than the RWL one. We can also observe that the standard deviation obtained using the RWS algorithm is lower than the one obtained with the RWL one.

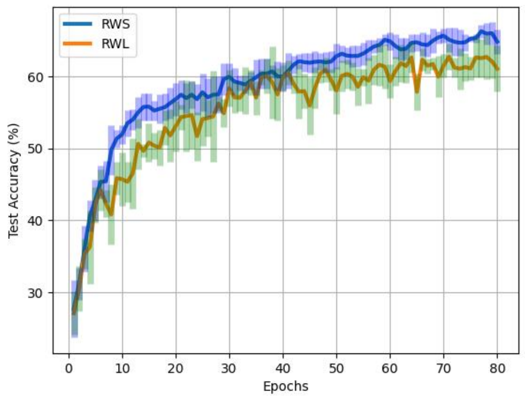

Figure 8: The result of the Adaptive RWS algorithm using a Dirichlet data distribution among $N = 30$ nodes for $M = 3$ using the CIFAR-10 dataset compared to the standard RWL.

In this figure, we have again increased the number of nodes to 30 without changing the number of models. The accuracy of the RWS algorithm is always higher than that of the RWL one. We can observe that also, in this case, the standard deviation obtained using the RWS algorithm is lower than the one obtained with the RWL one.

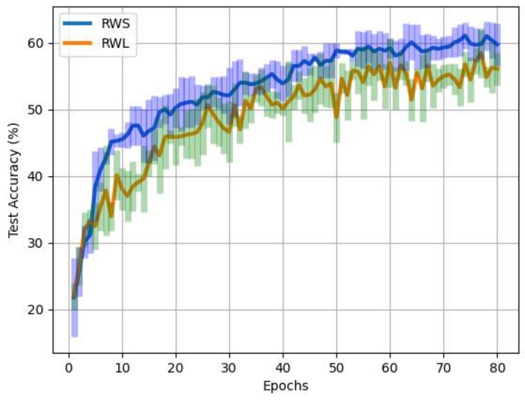

Figure 9: The result of the Adaptive RWS algorithm using a Dirichlet data distribution among $N = 50$ nodes for $M = 3$ using the CIFAR-10 dataset compared to the standard RWL.

In this figure, we have again increased the number of nodes to 50 without changing the number of models. The accuracy of the RWS algorithm is always higher than that of the RWL one. We can observe that, differently from the previous cases, the standard deviation obtained using the RWS algorithm is the same as that obtained with the RWL one.

We can conclude that even after increasing the number of nodes, the RWS algorithm is capable of performing better than the RWL algorithm, with some small changes to the number of models used.

3.4.6 Changing dataset

All the previous simulations were done with the CIFAR-10 dataset, so now we will test our algorithm using another dataset to verify that it is working in different scenarios. To accomplish this, we have chosen to use the Fashion-MNIST dataset.

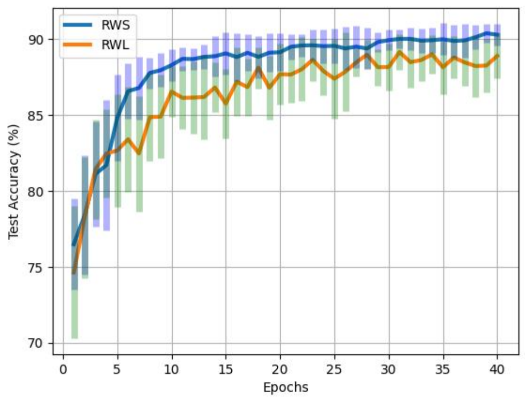

Figure 10: The result of the Adaptive RWS algorithm using a Dirichlet data distribution among $N = 10$ nodes for $M = 3$ using the Fashion-MNIST dataset compared to the standard RWL.

This first figure shows that the accuracy obtained using the RWS algorithm is higher than the one obtained with the RWL algorithm. We have also noticed that RWS converges faster. Regarding the standard deviation, the 2 algorithms obtain almost the same values.

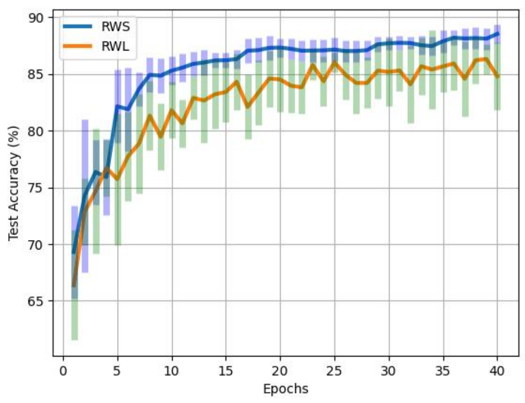

Figure 11: The result of the Adaptive RWS algorithm using a Dirichlet data distribution among $N = 30$ nodes for $M = 3$ using the Fashion-MNIST dataset compared to the standard RWL.

In this figure, we have the results obtained by increasing the number of nodes to 30; the behavior is pretty similar to the one with 10 nodes, except for a smaller standard deviation in the RWS algorithm.

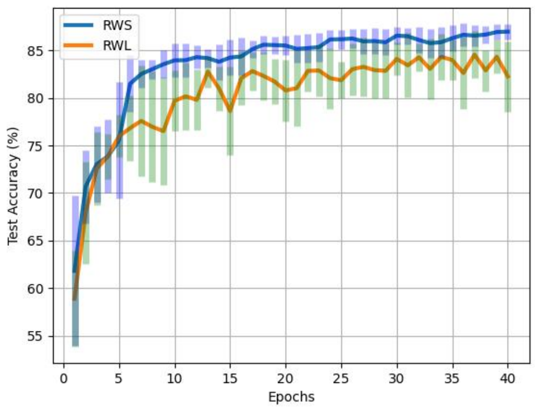

Figure 12: The result of the Adaptive RWS algorithm using a Dirichlet data distribution among $N = 50$ nodes for $M = 3$ using the Fashion-MNIST dataset compared to the standard RWL.

In this figure, we have the results obtained by increasing the number of nodes to 50. The behavior is pretty similar to the figures with 10 and 30 nodes, the standard deviation, as in the 30 nodes case, is smaller in the RWS algorithm.

From the last 3 figures, we can conclude that even after changing the dataset, the RWS algorithm performs better than the RWL algorithm.

CHAPTER 4

CONCLUSION

In this thesis, we presented an innovative methodology for decentralized learning called Random Walking Spiders (RWS), which addresses the limitations inherent in the traditional Random Walk Learning (RWL) approach. Through rigorous exploration and empirical analysis, the study demonstrates the effectiveness of RWS in improving learning accuracy and efficiency across distributed nodes, particularly in environments marked by heterogeneous data distribution. The introduction of both non-adaptive and adaptive RWS methodologies underscores the adaptability and potential of RWS to respond effectively to dynamic learning environments and performance metrics.

This research contributes significantly to the machine learning domain by proposing a novel strategy for decentralized learning that overcomes the challenges of traditional approaches. It opens new avenues for future investigations aiming at optimizing distributed learning systems. The detailed examination of challenges and the iterative enhancements introduced through the RWS strategy underscores the evolving nature of machine learning technologies. Such advancements have a profound impact on the methodologies for processing and learning data across decentralized networks.

The findings of this study reveal that the RWS strategy substantially improves the accuracy and efficiency of decentralized learning models, especially in scenarios characterized by uneven data distribution among nodes. The methodology's adaptability and responsiveness to changing dynamics and performance indicators make it a promising approach for practical applications in decentralized learning environments. This approach heralds a future where collaborative and distributed learning models are not only more efficient and adaptable but also more accessible. In conclusion, this thesis not only advances academic knowledge but also lays the groundwork for practical implementations in decentralized learning environments. It promises a future where collaborative and distributed learning models achieve greater efficiency, adaptability, and accessibility, "Representing a substantial advancement in machine learning."

CITED LITERATURE

Bibliography

[1] Hu, L., Chen, J., Vaughan, J., Yang, H., Wang, K., Sudjianto, A., and Nair, V. N.: Supervised machine learning techniques: An overview with applications to banking, 2020.

[2] Tiwari, A.: Supervised learning: From theory to applications. In Artificial Intelligence and Machine Learning for EDGE Computing, pages 23–32. Elsevier, 2022.

[3] Barlow, H. B.: Unsupervised learning. Neural computation, 1(3):295–311, 1989.

[4] Ghahramani, Z.: Unsupervised learning. In Summer school on machine learning, pages 72–112. Springer, 2003.

[5] Celebi, M. E. and Aydin, K.: Unsupervised learning algorithms, volume 9. Springer, 2016.

[6] Kaelbling, L. P., Littman, M. L., and Moore, A. W.: Reinforcement learning: A survey. Journal of artificial intelligence research, 4:237–285, 1996.

[7] Li, Y.: Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274, 2017.

[8] Bebis, G. and Georgiopoulos, M.: Feed-forward neural networks. Ieee Potentials, 13(4):27–31, 1994.

[9] Fine, T. L.: Feedforward neural network methodology. Springer Science & Business Media, 1999.

[10] O'shea, K. and Nash, R.: An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458, 2015.

[11] Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J.: A survey of convolutional neural networks: analysis, applications, and prospects. IEEE transactions on neural networks and learning systems, 33(12):6999–7019, 2021.

[12] Sakib, S., Ahmed, N., Kabir, A. J., and Ahmed, H.: An overview of convolutional neural network: Its architecture and applications. 2019.

[13] Datta, L.: A survey on activation functions and their relation with xavier and he normal initialization. arXiv preprint arXiv:2004.06632, 2020.

[14] Datta, L.: A survey on activation functions and their relation with xavier and he normal initialization. arXiv preprint arXiv:2004.06632, 2020.

[15] Mao, A., Mohri, M., and Zhong, Y.: Cross-entropy loss functions: Theoretical analysis and applications. In International conference on Machine learning, pages 23803–23828. PMLR, 2023.

[16] Zhang, Z. and Sabuncu, M.: Generalized cross entropy loss for training deep neural networks with noisy labels. Advances in neural information processing systems, 31, 2018.

[17] Liu, W., Wen, Y., Yu, Z., and Yang, M.: Large-margin softmax loss for convolutional neural networks. arXiv preprint arXiv:1612.02295, 2016.

[18] Gentile, C. and Warmuth, M. K.: Linear hinge loss and average margin. Advances in neural information processing systems, 11, 1998.

[19] Toro-Vizcarrondo, C. and Wallace, T. D.: A test of the mean square error criterion for restrictions in linear regression. Journal of the American Statistical Association, 63(322):558–572, 1968.

[20] Allen, D. M.: Mean square error of prediction as a criterion for selecting variables. Technometrics, 13(3):469–475, 1971.

[21] Hecht-Nielsen, R.: Theory of the backpropagation neural network. In Neural networks for perception, pages 65–93. Elsevier, 1992.

[22] Zhang, Z.: Derivation of backpropagation in convolutional neural network (cnn). University of Tennessee, Knoxville, TN, 22:23, 2016.

[23] Amari, S.-i.: Backpropagation and stochastic gradient descent method. Neurocomputing, 5(4-5):185–196, 1993.

[24] Zhang, Z.: Improved adam optimizer for deep neural networks. In 2018 IEEE/ACM 26th international symposium on quality of service (IWQoS), pages 1–2. Ieee, 2018.

[25] Babu, D. V., Karthikeyan, C., Kumar, A., et al.: Performance analysis of cost and accuracy for whale swarm and rmsprop optimizer. In IOP Conference Series: Materials Science and Engineering, volume 993, page 012080. IOP Publishing, 2020.

[26] Zhang, N., Lei, D., and Zhao, J.: An improved adagrad gradient descent optimization algorithm. In 2018 Chinese Automation Congress (CAC), pages 2359–2362, 2018.

[27] Hawkins, D. M.: The problem of overfitting. Journal of Chemical Information and Computer Sciences, 44(1):1–12, 2004. PMID: 14741005.

[28] Xu, H., Chen, M., Meng, Z., Xu, Y., Wang, L., and Qiao, C.: Decentralized machine learning through experience-driven method in edge networks. IEEE Journal on Selected Areas in Communications, 40(2):515–531, 2022.

[29] Smajić, A., Grandits, M., and Ecker, G. F.: Privacy-preserving techniques for decentralized and secure machine learning in drug discovery. Drug Discovery Today, 28(12):103820, 2023.

[30] Li, H., Shou, G., Hu, Y., and Guo, Z.: Mobile edge computing: Progress and challenges. In 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), pages 83–84, 2016.

[31] Zaeem, R. N. and Barber, K. S.: The effect of the gdpr on privacy policies: Recent progress and future promise. ACM Trans. Manage. Inf. Syst., 12(1), dec 2020.

[32] Liang, X., Javid, A. M., Skoglund, M., and Chatterjee, S.: Learning without forgetting for decentralized neural nets with low communication overhead. In 2020 28th European Signal Processing Conference (EUSIPCO), pages 2185–2189, 2021.

CITED LITERATURE (continued)

[33] Kang, J., Xiong, Z., Jiang, C., Liu, Y., Guo, S., Zhang, Y., Niyato, D., Leung, C., and Miao, C.: Scalable and communication-efficient decentralized federated edge learning with multi-blockchain framework. In Blockchain and Trustworthy Systems, eds. Z. Zheng, H.-N. Dai, X. Fu, and B. Chen, pages 152–165, Singapore, 2020. Springer Singapore.

[34] Alsagheer, D., Xu, L., and Shi, W.: Decentralized machine learning governance: Overview, opportunities, and challenges. IEEE Access, 11:96718–96732, 2023.

[35] Mammen, P. M.: Federated learning: Opportunities and challenges. CoRR, abs/2101.05428, 2021.

[36] Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., and Gao, Y.: A survey on federated learning. Knowledge-Based Systems, 216:106775, 2021.

[37] Li, Q., Wen, Z., Wu, Z., Hu, S., Wang, N., Li, Y., Liu, X., and He, B.: A survey on federated learning systems: Vision, hype and reality for data privacy and protection. IEEE Transactions on Knowledge and Data Engineering, 35(4):3347–3366, 2023.

[38] Li, T.: Scalable and trustworthy learning in heterogeneous networks, 2023.

[39] Boyd, S. P., Ghosh, A., Prabhakar, B., and Shah, D.: Randomized gossip algorithms, 2006.

[40] Nedic, A. and Ozdaglar, A.: Distributed subgradient methods for multi-agent optimization, 2009.

[41] A. Koloskova, S. S. and Jaggi, M.: Decentralized stochastic optimization and gossip algorithms with compressed communication, 2019.

[42] T. Aysal, M. E. Yildiz, A. S. and Scaglione, A.: Broadcast gossip algorithms for consensus, 2009.

[43] J. C. Duchi, A. A. and Wainwright, M.: Dual averaging for distributed optimization: Convergence analysis and network scaling, 2012.

[44] D. Kempe, A. D. and Gehrke, J.: Gossip-based computation of aggregate information, 2003.

[45] Xiao, L. and Boyd, S. P.: Fast linear iterations for distributed averaging, 2003.

[46] S. Boyd, A. Ghosh, B. P. and Shah, D.: Randomized gossip algorithms, 2006.

[47] X. Lian, W. Zhang, C. Z. and Liu, J.: Asynchronous decentralized parallel stochastic gradient descent, 2018.

[48] Y. Li, M. Yu, S. L. A. S. A. N. S. K. and Schwing, A. G.: Pipe-sgd: A decentralized pipelined sgd framework for distributed deep net training, 2018.

[49] Avidor, T. and Tal-Israel, N.: Locally asynchronous stochastic gradient descent for decentralized deep learning, 2022.

[50] Xia, F., Liu, J., Nie, H., Fu, Y., Wan, L., and Kong, X.: Random walks: A review of algorithms and applications. IEEE Transactions on Emerging Topics in Computational Intelligence, 4(2):95–107, April 2020.

CITED LITERATURE (continued)

[51] Li, X., Huang, K., Yang, W., Wang, S., and Zhang, Z.: On the convergence of fedavg on non-iid data, 2020.

[52] K., P. and et al.: Advances and open problems in federated learning, 2021.

[53] Li, T., Sahu, A. K., Talwalkar, A., and Smith, V.: Federated learning: Challenges, methods, and future directions, 2019. ArXiv.

[54] Ardic, A. B., Seferoglu, H., El Rouayheb, S., and Koyuncu, E.: Random walking snakes for decentralized learning at edge networks. In 2023 IEEE 29th International Symposium on Local and Metropolitan Area Networks (LANMAN), pages 1–6, 2023.

[55] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A.: Communication-efficient learning of deep networks from decentralized data, 2023.

VITA

NAME	Federico Ghiglione
EDUCATION	
	Master of Science in “Computer Science, University of Illinois at Chicago, May 2024, USA
	Specialization Degree in “ Artificial Intelligence and Data Analytics ”, Jul 2024, Polytechnic of Turin, Italy
	Bachelor’s Degree in Computer Engineering (CE) -
	Computer Engineering, Jul 2022, Polytechnic of Turin, Italy
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	2022 - IELTS examination (7.5/9)
	A.Y. 2023/24 One Year of study abroad in Chicago, Illinois
	A.Y. 2022/23. Lessons and exams attended exclusively in English
SCHOLARSHIPS	
Spring 2024	Research Assistantship (RA) position (10 hours/week) with full tuition waiver plus monthly stipend
Fall 2024	Italian scholarship for final project (thesis) at UIC
Fall 2024	Italian scholarship for TOP-UIC students
TECHNICAL SKILLS	
Basic level	Javascript
Average level	Java
Average level	C++
Average level	Machine Learning
Average level	Big Data Analysis(Spark and Hadoop)
Advanced level	Python

VITA (continued)

Advanced level	C
Advanced level	Relational Databases(SQL)
Advanced level	Neural Networks
Advanced level	Python for Neural Networks (pytorch)

WORK EXPERIENCE AND PROJECTS

March 2022 - Jul 2022	Android Developer Android developer internship at Feedback Italia srl, Developed android applications using Java and Kotlin
March 2023 - Jul 2023	Machine learning project Group project of machine learning: implementation of various machine learning models such as Gaussian models, SVM, Linear Regression models, GMM, in order to compare them on the same dataset. The dataset was also pre-processed with techniques such as PCA, LDA and z-normalization.
March 2023 - Jul 2023	Software Engineering project Making a web application using javascript with all the phase of a real software engineering project such as documentation, the design of the front end, the coding, and the testing.
May 2023 - July 2023	System and Device Programming project Use the operating system os-161 to write from scratch a certain number of system calls and test them
September 2023 - December 2023	Computer Vision project Implementation of a barcode detector using the OpenCV library with Python, without using other pre-built libraries already optimized for achieving this task

VITA (continued)

September 2023 - December 2023	Causal Inference and Learning project
	Investigate possible new algorithms for Causal inference, starting from the PC and the FCI algorithms
2022-2024	Other Experiences:
	Big data analysis using Spark and Map Reduce for Hadoop
	Arm Mips and 8086 programming
	Web app developing
	Multi-process and multi-thread C++ programming