WFNAS: Weight-Agnostic Federated Neural Architecture Search

by

Om Thakkar

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2021 by the
Graduate Supervisory Committee:

Rida Bazzi, Chair
Baoxin Li
Yu Zhang

ARIZONA STATE UNIVERSITY

December 2021

ABSTRACT

Federated Learning (FL) is envisaged to be a promising solution for collaboratively training a machine learning model while keeping the training data decentralized and private. Instead of sharing raw data to the central entity, the participating client devices share focused updates for aggregation to ensure global convergence of the model. Owing to the shortcomings of manually handcrafted neural network architectures, the research community is striving to develop Neural Architecture Search (NAS) approaches to automatically search for optimal networks that fit the clients' data. Despite the inaccessibility of clients' data in an FL setting, the federated NAS literature has recently witnessed great progress to apply these NAS techniques to an FL setting. However, one of the key bottlenecks of Federated Learning is the cost of communication between clients and the server, and the state-of-the-art federated NAS techniques search for networks with millions of parameters that require several rounds of communication to find the optimal weight parameters. Also, deploying a network having millions of parameters on edge devices (which are the typical participants in an FL process) is infeasible due to its computational limitations and increased latency. Thus, this work proposes *Weight-Agnostic Federated Neural Architecture Search (WFNAS)*, a novel evolutionary framework to search for well-performing and minimally connected weight-agnostic network architectures in an FL setting. As the connectivity of the networks themselves is the solution, there is no need for weight training and hyperparameter tuning, reducing the communication overhead significantly. The experiments indicate a gain of nearly 40% for orthogonal (vertical FL) data distributions compared to local training. This work is the first federated NAS technique in the literature for vertical FL. Although the experiments are performed in a resource-constrained environment, the aim of this thesis is to show a new direction of research to the FL community.

DEDICATION

Dedicated to my parents, Mr. Rakesh Thakkar and Mrs. Bhavna Thakkar for their unfathomable love, endless support, and sacrifices that serve as a constant motivation for me to become a better version of myself every day.

ACKNOWLEDGEMENTS

I would like to sincerely thank Dr. Rida Bazzi for his constant encouragement and support during the course of this work. It has been an absolute pleasure to learn from him while engaging in intellectually stimulating brainstorming sessions. This work would not have been possible without his guidance. I would also like to thank Dr. Baoxin Li and Dr. Yu Zhang for serving on my thesis committee and for providing valuable feedback that helped improve the quality of my work.

I am also extremely thankful to several individuals and organizations for providing me with the required computing resources and infrastructural support - Agave Cluster at ASU, Dr. Ming Zhao & Yitao Chen (VISA Research Lab at ASU), Dr. Lina Karam (IVU Lab at ASU), Renate Mittelmann (School of Mathematical and Statistical Sciences), and BYENG Lab at ASU.

The journey of this work met a lot of challenges and saw days of highs and lows, but I am immensely grateful to my parents, family, and friends for their constant motivation and faith in me. Their support and well-wishes served as a catalyst in the completion of this work.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

CHAPTER

1 INTRODUCTION 1

2 BACKGROUND AND RELATED WORK 8

2.1 Federated Learning 9

2.2 Neural Architecture Search 18

2.3 Federated Neural Architecture Search 26

3 IN SEARCH OF WFNAS 35

3.1 Dataset 35

3.2 Data Distribution among clients 35

3.3 Trivial exchange of networks 37

3.4 Estimation of rewards 38

3.5 Comparison of estimators 38

3.5.1 Mean 39

3.5.2 Normal Distribution 39

3.5.3 Hand-crafted features for regression model 40

3.5.4 Network Matrix as features 41

3.5.5 Rewards as features 43

3.5.6 Estimation Framework 47

3.6 Weighted averaging of rewards 47

3.7 Choosing which networks to share? 48

3.8 Historical networks 51

3.9 Separately evolving imported networks 51

CHAPTER Page

3.10 Estimator for Local Rewards 52

3.11 Informed Sharing 53

4 WFNAS: THE PROPOSED APPROACH 55

5 EXPERIMENTAL RESULTS AND DISCUSSION 59

5.1 Dataset & Image Preprocessing 59

5.2 Experimental Setup 60

5.3 Orthogonal data distribution 61

5.3.1 Effect of sharing percentages 62

5.3.2 Effect of sharing period 65

5.4 non-IID data distribution 66

5.5 Communication Overhead 69

6 CONCLUSION AND FUTURE WORK 71

REFERENCES 73

LIST OF TABLES

Table Page

2.1 Differences Between Cross-device and Cross-silo Fl [23] 10

3.1 Client A: Experiment Results for Choosing Networks for Sharing 50

3.2 Client B: Experiment Results for Choosing Networks for Sharing 50

5.1 Mnist Test Dataset 60

5.2 No. Of Samples per Class in Orthogonal Data-distribution 61

5.3 Comparison of Test Accuracy for Different Values of p 63

5.4 Comparison of Test Accuracy for Different Values of q 64

5.5 Comparison of Test Accuracy for Different Values of E 66

5.6 No. Of Samples per Class in Non-iid Data-distribution 66

5.7 Comparison of Test Accuracy for Non-iid Data Distribution 67

5.8 Empirical Values of Attributes of the Champion Networks 70

LIST OF FIGURES

Figure Page

2.1 Overview of Nas Process [12] . . . 18

2.2 Controller Rnn Sampling a Simple Convolutional Neural Network [61] . 21

2.3 Overview of Weight-agnostic Neural Network Search [14] . . . 23

2.4 Neat Algorithm . . . 25

2.5 Search Space Considered in Fednas [17] . . . 27

2.6 Visualization of Sampling Child Networks from a Parent Network [15] . 31

2.7 Algorithm to Perform Local Search at the Clients [15] . . . 32

2.8 Direct Federated Nas Algorithm [15] . . . 32

3.1 Example of Non-iid Data Distribution . . . 36

3.2 Example of Orthogonal Data Distribution . . . 37

3.3 Normal Distribution Vs Regression Model Using Hand-crafted Features 40

3.4 Normal Distribution Vs Different Models Using Network Matrix as Features . . . 43

3.5 Mean Squared Error (Mse) (Client A) . . . 45

3.6 R2 Score (Client A) . . . 45

3.7 Mean Squared Error (Mse) (Client B) . . . 46

3.8 R2 Score (Client B) . . . 46

3.9 Estimation Framework . . . 47

3.10 Using Estimator for Governing Local Evolution . . . 53

4.1 Exchange of Networks in Wfnas . . . 57

5.1 Example Image from Mnist Dataset . . . 59

5.2 Effect of Sharing Percentage - p . . . 63

5.3 Effect of Sharing Percentage - q . . . 64

5.4 Effect of Sharing Period - E . . . 65

Figure Page

5.5 No-sharing Vs 20-20-10 for Non-iid Data Distribution 67

5.6 Comparison of Per-class Test Accuracy: No-sharing Vs 20-20-10 Sharing (Client A) 68

5.7 Comparison of Per-class Test Accuracy: No-sharing Vs 20-20-10 Sharing (Client B) 69

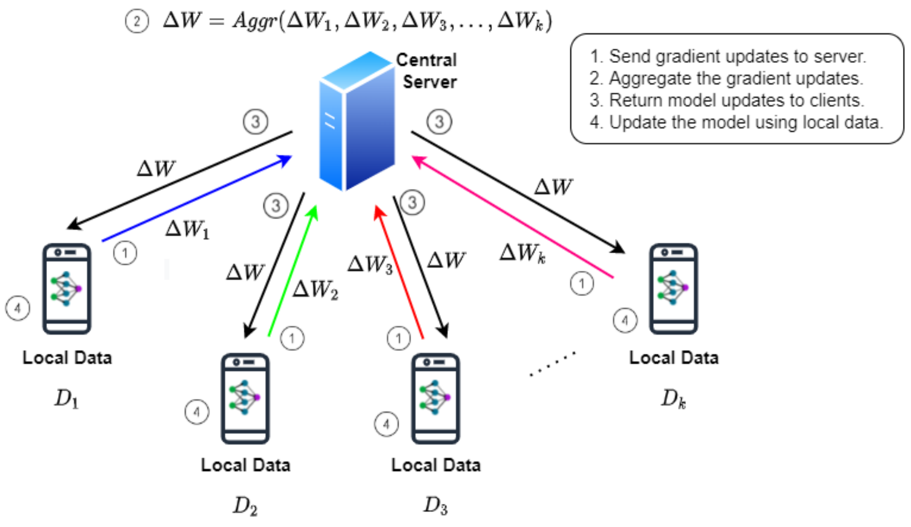
Chapter 1

INTRODUCTION

With the advent of data-hungry algorithms capable of performing convoluted computer vision tasks ranging from healthcare [22] to automotive [47] to manufacturing [54], it is of paramount importance to talk about the privacy and security concerns of the data being used. Conventional machine learning and deep learning algorithms are trained by centralizing all the data in a single machine or a data center. However, is the practice of centralizing data to perform training always feasible and "privacy-preserving"? Furthermore, challenges like high costs of storing large amounts of data, increased communication overheads, confidentiality concerns, and legal constraints are compelling enough to reconsider the assumptions of the standard practices to train a machine learning model. Federated Learning (FL) is a promising and reliable approach to address these challenges.

Federated Learning is a machine learning setting where multiple clients collaboratively solve a machine learning problem, under the orchestration of a central server where each client's data is only stored locally and does not leave the clients' device. Instead, appropriate statistics representative of the clients' local data such as gradient updates is shared for immediate aggregation at the central server in order to achieve the learning objective [23]. It is noteworthy that these focused updates shared by the client are reduced to contain only the minimum information required for the global model to progressively move towards convergence. An illustration of a typical federated learning workflow is shown below.

The key idea behind the Federated Learning approach, that is the ability to learn from the invisible data distributed among various clients in an unknown fashion is


② $\Delta W = Aggr(\Delta W_1, \Delta W_2, \Delta W_3, \dots, \Delta W_k)$
Central Server
1. Send gradient updates to server.
2. Aggregate the gradient updates.
3. Return model updates to clients.
4. Update the model using local data.
ΔW
ΔW_1
ΔW_2
ΔW_3
ΔW_k
Local Data D_1
Local Data D_2
Local Data D_3
Local Data D_k

A typical federated learning workflow

a long-standing goal for many research communities. There have been several early efforts to exploit the user data while preserving user privacy by leveraging cryptography techniques [41], [57], but Kairouz et al. [23] note that there is no single work in the literature yet that encompasses all the different challenges posed by federated learning. One of the primary reasons for this is that the open research problems in federated learning are fairly interdisciplinary and require the knowledge of distributed optimization, cryptography, differential privacy, compressed sensing, information theory, statistics, etc.

In this thesis, we particularly focus on the unique requirements to automatically search for optimal neural architectures in a federated learning setting. Hand-crafting neural network architectures is a time-consuming process. For example, the CNN architectures are a result of an effort of decades that have a strong bias towards computer vision tasks. Furthermore, this challenge is fueled by the invisibility of clients' data in an FL setting, rendering handcrafted network architectures unfavorable for federated learning. To circumvent this, Neural Architecture Search (NAS) is a favorable and well-studied technique to automatically search for optimal neural network architectures using various optimization algorithms for a specific task on a specific dataset. There are three methods primarily used to perform NAS: *evolutionary algorithms* [38], *reinforcement learning* [61], and *gradient descent* [29]. Although the recent progress in the NAS literature (in a non-federated setting) [18, 39, 11, 38, 2, 35, 29, 53, 30] is quite phenomenal, this thesis addresses the unique challenges that come into play while using the standalone NAS techniques in a federated learning environment.

Researchers in the FL community refer to the intersection of the above two areas as Federated Neural Architecture Search. This area has recently started receiving attention in the research community and very few papers have been published that address this exact problem. *FedNAS* [17] is one of the pioneering works in this realm to help the scattered workers collaboratively search for a better network architecture with higher performance accuracy. *Differentially-Private Federated Neural Architecture Search* [44] is built on top of *FedNAS* by adding noise to the gradient updates sent to the server. However, the aforementioned two papers follow a two-stage approach: *1) search* and *2) parameter retraining*, which is quite computationally exhaustive and infeasible in an FL setting. To address this problem, Garg et al. [15] proposed an approach called *Direct Federated Neural Architecture Search*, which is capable of performing the NAS task in one step. This approach is based on *Stochastic Neural Architecture Search (SNAS)* [53] and *Discrete Stochastic Neural Architecture Search (DSNAS)* [20]. The existing works elucidated so far are gradient-descent based approaches. Zhu et al. [58] propose a multi-objective evolutionary offline federated learning that aims to minimize the communication overhead by simplifying the network structure and improving performance accuracy with weight-training in federated learning. In [59], an evolutionary approach to *real-time* federated neural architecture search that focuses on optimizing the model performance is proposed. All the aforementioned federated NAS approaches are applicable only for horizontal FL setting, where data is distributed by samples (horizontal), and not by features (vertical). According to Zhu et al. [60], there is no single work in the literature of federated NAS for a vertical FL setting yet. A detailed discussion of all the aforementioned papers is provided in Chapter 2.

Out of the many core challenges of federated learning as identified and outlined in [23], communication is a primary bottleneck since wireless links and other end-user internet connections typically operate at lower rates than intra- or inter-datacenter links and can potentially be expensive and unreliable. One of the key drawbacks of the existing Federated NAS techniques discussed above is that the neural networks searched by these techniques are very expensive in terms of size and memory. Such neural networks may not be feasible in a real-world setting to perform real-time inference after the deployment of the model. It is common for devices participating in the FL setup to be edge devices, IoT sensors, or mobile phones which have memory and computational constraints. This indicates a need for a Federated NAS technique that can search for much simpler networks that are easily deployable and are capable of performing real-time inference with low latency.

The existing works in the literature, be it gradient-based, reinforcement learning-based or evolutionary, focus on a joint-optimization of the weights w and architecture α, which makes the problem harder to solve especially in a communication-bound federated learning environment because the clients are required to send gradient updates for both the parameters for aggregation at the central entity. However, it was recently demonstrated that the machine learning task at hand can be solved using the innate characteristics of the network architecture itself [14]. In other words, a randomly chosen single weight value can be used for all the connections to solve the machine learning problem solely on the basis of the connectivity of neurons in the network. Such networks are termed as "Weight Agnostic Neural Networks" (WANNs) and have not only shown performance accuracy comparable to the state-of-the-art networks for reinforcement learning and image classification tasks, but the searched networks are orders of magnitude smaller than the traditional networks. This makes WANNs highly favorable for deployment at edge devices for real-time inference in a federated learning setting. WANNs use the very popular and well-developed evolutionary algorithm named *NeuroEvolution of Augmenting Topologies (NEAT)* [45] to evolve the networks of the current generation to obtain the next generation of high-performance network topologies.

In this thesis, we extend the idea of WANNs to the realm of federated learning and propose a novel infrastructure for neural architecture search that can eliminate the weight-training and hyperparameter tuning process. For the sake of simplicity and due to resource constraints, we restrict our study to a 2-client FL setting, but we provide information to easily extend our work to a multi-client FL setting as a part of future work. For the ease of understanding, let the participating clients be denoted by "A" and "B". Both the clients evolve their networks locally based on their respective local data using a search process similar to [14]. After every iteration, the low-performing networks are eliminated as a part of the evolution process. However, the networks that are low-performing on Client A's data may not be low-performing on Client B's data and vice versa. Therefore, in order to validate the performance of the locally evolved networks on the other client's data, we propose the following idea: *A sends $p\%$ of its network architectures to B and receives the evaluated reward for those networks. To obtain the reward for the remaining $(100 - x)\%$ networks on the local data of B, we use a robust estimation technique to predict the performance of the remaining $(100-x)\%$ of networks on the local data of B. The networks of A are ranked based on their evaluations on A and B's data and the best-performing $q\%$ networks* *are sent to Client B again.* The same approach applies for Client B. We evaluate our approach on two types of FL settings: *i) vertical (orthogonal data distribution)* and ii) *horizontal (non-IID data distribution)* and compare it with the performance accuracy of standalone (local) training using the same data distributions. Our experiments demonstrate a gain of nearly 40% points for vertical FL and about 2% points for horizontal FL even though MNIST performs well in standalone training.

Evidently, our proposed network setup has two major differences in comparison to the standard FL architecture: 1) Since the networks searched by our proposed approach are weight-agnostic, no gradient updates are shared by the client. Instead, each client shares a certain percentage of networks evolved locally. 2) There is no central entity in the proposed network, instead each client shares its networks with the other clients in a decentralized manner.

The contributions of this work can be outlined as follows:

- A novel *weight-agnostic* neural architecture search framework is proposed for a federated learning environment for the first time in the literature. The proposed approach is capable of searching for lightweight networks (in terms of size and memory) that demonstrate performance accuracy comparable to the non-federated setting.

- This work serves as the first federated learning NAS technique that works for vertically distributed data. In addition, we also evaluate our approach for horizontally distributed data in a non-IID fashion.

- Extensive experiments have been conducted to test the hypothesis for all the independent aspects of the proposed approach. For example, choosing which networks to share, tuning the sharing percentage and sharing period for high-performance networks, etc.

The rest of the thesis is organized as follows: In **Chapter 2**, we provide the necessary background and a detailed discussion of the state-of-the-art federated NAS approaches. We also identify the shortcomings and potential avenues of the existing approaches with an intention and expectation that this thesis serves as a reference for readers looking to contribute to the research community of federated learning. In **Chapter 3**, we present the various experiments we conducted that led us to WFNAS. We elaborate on the different parts of our proposed approach to empirically reason and justify our choices. We explored several aspects of the approach like how many networks to share with the other client, how often to share, how to choose which networks to share, what features to use for training the machine-learning based estimator for predicting the performance of networks that are not shared with the other client on their data, what strategies to use for improving the overall performance, etc. In **Chapter 4**, we present our proposed approach based on the findings and conclusions from the experiments discussed in Chapter 3. In **Chapter 5**, we present the results of our proposed approach with various different configurations of sharing percentage and sharing period for both horizontal (non-IID) and vertical (orthogonal) FL settings. **Chapter 6** concludes with a summary of our work and discusses potential directions of future work.

Chapter 2

BACKGROUND AND RELATED WORK

In this chapter, we provide a detailed discussion of the state-of-the-art approaches in three areas that serve as the background for this work. i) *Federated Learning*, ii) *Neural Architecture Search*, and iii) *Federated Neural Architecture Search.*

In i), we provide a background of the realm of federated learning and identify potential avenues for future research in various aspects of federated learning. In ii), we discuss the well-known state-of-the-art (non-federated) techniques to perform neural architecture search. In this subsection, we also discuss the NAS technique used by WANNs. This technique is adopted for the local search of WFNAS. Finally, in iii), a discussion of the state-of-the-art techniques at the intersection of federated learning and neural architecture search is provided.

Each related area is presented as a separate large section of this chapter to comprehensively understand the existing works and open research problems.

2.1 Federated Learning

In the past decade, the leaders in the industry have demonstrated an increasing interest in adopting and deploying federated learning-based solutions at scale [13] to adhere to privacy and legal requirements enforced by the government. For instance, Google uses federated learning techniques in the GBoard mobile keyboard [36, 16, 56, 6, 37] to enable training of machine learning models on-device to learn tasks like "next word recommendations". Apple used the federated learning technique in iOS 13 to make the "Hey Siri" vocal classifier more robust with minimal to no compromise of user privacy. Furthermore, Intel Labs have created a project called OpenFL [40] and is working in collaboration with the University of Pennsylvania to develop AI models for the identification of brain tumors by enabling a federation of 29 international healthcare and research institutions.

Kairouz et al. [23] is a comprehensively curated article identifying the open research problems in Federated Learning. The authors curate a new definition of *Federated Learning (FL)* that emerged over the time after FL was first proposed. "*Federated Learning is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client's raw data is stored locally and not exchanged or transferred; instead focused updates for immediate aggregation are used to achieve the learning objective.*" When FL was first introduced by McMahan et. al [31] in 2016, a set of challenges were defined for an ideal scenario: "An unbalanced and non-IID data partitioning across a massive number of unreliable devices with limited communication bandwidth." The authors mention that there is no single work in the literature yet that addresses all the challenges collectively.

The two primary variants of FL are *Cross-device* and *Cross-silo* FL. Both the

Cross-silo FL	**Cross-device FL**
Clients are *different organizations* (e.g. medical or financial)	Clients are very large number of *mobile or IoT devices*
Typically 2-100 clients	Massively parallel, up to 10^{10} clients
Each client has identity to access it specifically	Clients cannot be indexed directly.
Stateful - each client may participate in each round of computation	*Stateless* - each client will likely participate only once in a task
Relatively few failures	Highly unreliable - 5% or more clients are expected to fail in one round of computation

Table 2.1: Differences Between Cross-device and Cross-silo Fl [23]

variants mainly differ based on the environment/use-case for which they are used. A summary of differences between both the variants of FL is provided in Table. 2.1.

According to [23], a high-level workflow of a typical FL setting is the following:

1. *Problem Identification*
2. *Client Instrumentation to locally store training data*
3. *Simulation prototyping (optional) on proxy dataset*
4. *Federated model training*
5. *Federated model evaluation*

6. *Deployment*

A typical FL training process can be described as follows:

- Server repeats the following steps until training is stopped:
    1. **Client selection**
        - Eligibility requirements: plugged in, on an unmetered wi-fi and idle.
    2. **Broadcast**
        - Selected clients download current model weights & a training program.
    3. **Client computation**
        - executes training program on local data
    4. **Aggregation**
        - server aggregates the device updates and drops stragglers.
    5. **Model update**
        - server locally updates shared model based on the aggregated update.

The various FL settings achieved by relaxing the core FL assumptions are summarized below:

Fully Decentralized / Peer-to-Peer Distributed Learning

The usage of a single server also represents a single point of failure. Hence, a fully decentralized version of FL is preferred, where the network is represented as a connected graph with nodes as clients and edges as communication channel.

- Open research problems
    1. Robustness to limited availability of clients (dropping out or rejoining) and limited reliability of network.

2. Achieving faster convergence under non-IID distributions.
3. Translating existing compressed communication schemes from centralized to fully decentralized without negatively impacting convergence.

Cross-silo FL

Companies/Organizations train a model collectively without sharing their data directly. This could be due to legal constraints or confidentiality impositions. Data is partitioned by features, unlike cross-device FL where data is partitioned by example.

- Open research problems

  1. Development of incentive mechanisms for honest and long-term participation of clients in the training process.
  2. Formal privacy guarantees using differential privacy.

Split Learning

Splitting the execution of an ML model on a per-layer basis between the clients and the server such that identifying information like true labels stay with the clients.

- Open research problems

  1. Enabling split learning on edge devices.
  2. Formal privacy guarantees using differential privacy.

Moreover, new set of challenges arise when adapting standard ML workflows and pipelines in a typical FL setting. Several challenges have been identified as below:

- **Hyperparameter Tuning**

  1. Due to the communication overhead that comes because of *Federated Learning*, it is not feasible to run many rounds on resource-constrained mobile devices.

    2. Moreover, FL introduces many more hyperparameters of its own such as no. of clients per round and no. of local steps per round increasing the search space significantly.
    3. Longer than usual training time becomes a bottleneck in an FL setting.

- **Neural Architecture Search**
    1. Predefined architectures may not be the best choice when data generated by users are invisible to the model developers.
    2. Apart from the state-of-the-art approaches for NAS such as Reinforcement Learning, the authors particularly emphasize on a recently published *Weight Agnostic Neural Networks (WANN)* that search for efficient neural architectures with only a single weight value by leveraging the evolutionary algorithms.

The authors in [23] explicitly encourage the use of WANN since it can be applied without collaborative weight-it is training among devices.

Furthermore, it is quite evident that *Communication* is the primary bottleneck in any FL setting. Therefore, the authors also focus on various compression techniques emphasizing on how they can be used for a FL setting.

- **Gradient compression**: reducing the size of object communication from client to server since the upload bandwidth of clients is significantly smaller than download bandwidth.
- **Model broadcast compression**: reducing the size of model from server to clients since it is a redundant process for server to be performed for all clients.
- **Local computation reduction**: modifying the local training process to be computationally efficient.

Several open research problems identified to mitigate the bottleneck of *communication* are as follows:

1. Developing a more compact model for efficient inference. The existing techniques are difficult to map to a FL setting.

2. Techniques like *secure aggregation* and *differential privacy* are not designed to work with compressed communications. Hence, joint design of compression methods compatible with Secure Aggregation & DP is an open problem.

Following this, the authors also introduce the existing work in FL from the aspect of privacy-preserving machine learning. In an *ideal world* of FL, each actor in the system learns nothing more than the information needed to play their role and each actor is fully aware of the extent of personal information that will be revealed in the training process. Therefore, the privacy aspects of FL computation of function f can be categorized into three groups: (1) How is f computed? *(Secure MPC and TEEs)*, (2) What is computed? *(DP)* and (3) Verifiability? *(ZKP)*. In what follows, we provide a brief overview of all the three aspects and enumerate the open research areas in each of the domains as identified in [23].

Secure Multi-Party Computation (MPC)

A sub-field of cryptography with a goal of having a set of parties compute an agreed-upon function of their private inputs that only reveals the intended output to each of the parties.

- Open research problems

    1. Porting the existing protocols in Secure MPC to a FL setting is not straightforward and is an open area of research.

Homomorphic Encryption

These schemes allow certain mathematical operators to perform computation on cipher texts without prior decryption. This encryption scheme can be used to enable MPC by enabling participants to compute functions on data while keeping their data *private*.

- Open research problems

    1. "Who must hold the secret key of the Homomorphic Encryption scheme in an FL setting?" is the biggest open research problem.

Trusted Execution Environments (TEEs)

TEEs are isolated environments within the device/cloud where part of the code of FL process can be executed with high level of trust in surrounding environment.

- Open research problems

    1. Current TEEs have memory limitations & can only provide access to CPU resources.
    2. TEEs are unable to fully exclude all the side-channel attacks.
    3. How should FL functions be partitioned across TEEs & client devices?

Distributed Differential Privacy

One of the primary limitations of employing *local differential privacy* is the need for a trusted aggregator, which is unlikely in an FL setting. Hence, distributed DP can be employed with two different approaches: *(1) Secure Aggregation (2) Secure Shuffling*.

- Open research problems

1. We discussed *Moments Accountant* technique before which has been developed for a central model of DP, but no privacy accounting techniques exist for a distributed setting of DP.
2. Determining the amount of noise to be added to meet the DP constraint is difficult due to very unstable environment of an FL setting where the clients dropout frequently during the training process.

Zero-Knowledge Proofs (ZKPs)

ZKPs are cryptographic protocols that enable a *prover* to prove statements to a *verifier* such that the *verifier* learns no information from the proof except that what is proved is true.

- Open research problems

  1. ZKPs are typically interactive but there are non-interactive protocols for some applications. FL demands non-interactive ZKPs such that the secret is shared among the participating clients.

Finally, the authors provide a discussion on the open research problems in FL setting from the aspect of robustness to attacks and failures. An adversary can be capable of introducing attacks like *Model Poisoning*, *Data Poisoning* and *Evasion Attacks*. An **important open research direction** identified while reading through the paper is the use of **PixelDP DNN** [28] architecture discussed before as the training program in an FL setting. This can be a promising research direction as the PixelDP is capable of providing certified robustness guarantees against norm-bounded adversarial attacks. Moreover, other than malicious attacks, non-malicious failure nodes such as *Client reporting failures, data-pipelining failures and noisy model updates* can also cause problems in an FL setting.

- Open research problems from the aspect of non-malicious failure nodes

    1. Developing Secure Aggregation strategies that aggregates over multiple computation rounds can be used to mitigate the client reporting failures.
    2. Developing general-purpose debugging methods for ML that do not directly inspect raw data can be used to overcome the data-pipeline failures.
    3. Developing training methods that are robust to small to moderate amounts of noise is an open challenge in an FL setting to avoid noisy model updates.

2.2 Neural Architecture Search

Neural Architecture Search (NAS) has received considerable attention from the research community with a motivation that the conventional network architectures used in practice are *handcrafted* by human experts which is a *time-consuming* and *error-prone* process. Also, *handcrafted* network architectures might not necessarily be the best for tasks other than for which they were designed. Recent works in NAS as discussed subsequently in this subsection have already outperformed manually designed architectures on tasks like *Image Classification, Object Detection* and *Semantic Segmentation*. In what follows, we provide a background and literature review of the state-of-the-art NAS techniques.

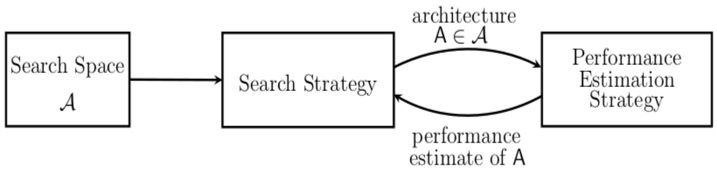

Search Space
$\mathcal{A}$
Search Strategy
architecture
$\mathsf{A} \in \mathcal{A}$
Performance Estimation Strategy
performance estimate of A

Figure 2.1: Overview of Nas Process [12]

Fig. 2.1 demonstrates a typical neural architecture search process as presented in [12]. NAS methods are categorized in three dimensions: *(1) Search Space, (2) Search Strategy and (3) Performance Estimation Strategy.* More specifically, a search strategy selects an architecture **A** from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of **A** to the search strategy.

Search Space

Starting from a very simplistic network namely *Chain-structured NNs*, the authors in [12] indicate the typical parameters of a search space such as *maximum number* *of layers, types of operations for every layer and hyperparameters associated with the operations.* As apparent, the search space grows exponential as the complexity of the NNs increases by considering more complex networks like *Residual Networks*. The use of repeated motifs/cells can reduce the search space significantly and such motifs can be stacked in various ways to create an architecture. However, an open problem is how to decide what number of cells to be used and how should they be connected to build a robust and high-performing model.

Search Strategy

Numerous different search strategies can be used to explore the space of neural architectures such as *Random Search* [3] where the parameters are randomly chosen from the search space and the best performing set of parameters on validation set is chosen. This essentially represents a trade-off between *exploration* and *exploitation*. Furthermore, techniques using *Reinforcement Learning* [29] and *Evolutionary Algorithms* [38] yield promising solutions with better performance and smaller models.

- Open research problems

  1. Developing NAS methods for multi-task problems and for multi-objective problems, in which measures of resource efficiency are used as objectives along with predictive performance on unseen data.
  2. Furthermore, RL approaches can be extended to learn policies that are conditioned on a state that encodes task properties/resource requirements.

Performance Estimation Strategy

To rapidly evaluate the search strategies and take sound decisions for choosing a better architecture, there is a need for efficient performance estimation strategies. Several state-of-the-art approaches for performance estimation are as follows:

- **Lower fidelity estimates [61]**: Training for fewer epochs (the number of passes of the entire training dataset), on subset of data, downscaled data, etc. However, a drawback of this approach is that it often leads to underfitting and relative ranking changes dramatically if the difference between cheap approximation and full evaluation is too big.

- **Learning Curve Exploration [46]**: Extrapolating performance based on fewer epochs. However, a shortcoming of this approach is making good predictions in a relatively large search space is difficult.

- **Weight Inheritance [39]**: Using pretrained weights instead of training from scratch, but it may lead to overly complex architectures.

- **One-Shot Architecture Search [43]**: treats all the architectures as different sub-graphs of one super-graph and shares weights between the architectures that have edges of this supergraph in common. However, a drawback of this technique is that it underestimates the actual performance of the best architecture. Currently, it is unclear if it can be used for ranking.

One of the pioneering works in the field of NAS by Zoph et al. [61] is based on an interesting observation that the structure and connectivity of a neural network can be typically specified by a variable-length string. This observation makes it possible to use *Recurrent Neural Networks (RNNs)* to generate model descriptions (or variable strings) of neural networks and train this RNN with reinforcement learning to maximize the expected accuracy of the generated architectures on a validation set.

Consider an example of predicting feed-forward neural networks with only convolutional layers, the authors use the controller to generate their hyperparameters as a sequence of tokens using RNNs as shown in Fig. 2.2.

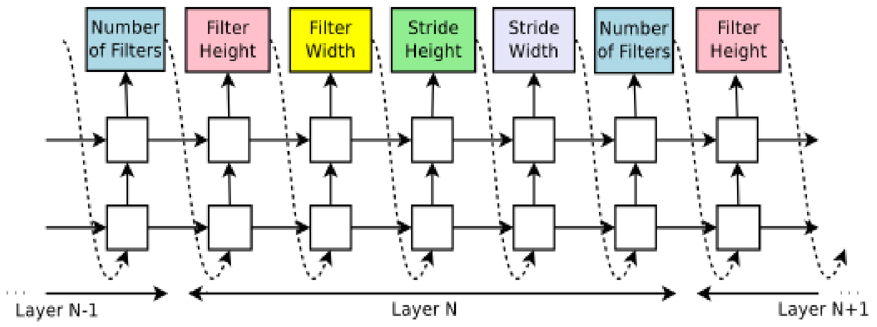


Figure 2.2: Controller Rnn Sampling a Simple Convolutional Neural Network [61]

The NAS process is trained using the well-known REINFORCE [52] algorithm. Moreover, in order to accelerate the training process, the authors use the parameter server architecture with asynchronous updates. The experiments presented in the paper use the CIFAR-10 dataset [24] for the image classification task. The training setup is quite heavy in terms of computational resources. 800 concurrent GPUs are used for distributed training of 800 networks in parallel. The authors compare the performance of the networks produced by their approach with the best performing human-curated architectures in the literature. The results indicate that the neural networks suggested by the proposed approach are better than most existing architectures in terms of performance accuracy except for one particular DenseNet architecture [21]. The implementation of this approach is included under the name of *NASCell* into TensorFlow [1] for use.

Following this, in 2019, Liu et al. [29] proposed *DARTS (Differentiable Architecture Search)* that approaches the problem of neural architecture search from a different viewpoint by relaxing the search space from discrete to a continuous domain. This enabled the search technique to optimize the network architecture w.r.t a validation set using gradient descent. The authors consider an overparameterized computation cell as the building block of a convolutional neural network (CNN) and perform a bilevel optimization of training and validation loss to find the connection at each layer with the highest expectation. Other connections having a relatively weak expectation in the overparameterized cell are subsequently removed. The obtained computation cell is later stacked and retrained to produce an optimal CNN architecture.

The parameter retraining in *DARTS* is a major overhead and in some way a contradiction to the norm in machine learning that no optimization is allowed in the evaluation stage. This led to *SNAS (Stochastic Neural Architecture Search)* [53] that proposes a one-round backpropagation technique to get rid of the parameter retraining phase while maintaining the completeness and differentiability of the NAS pipeline. In order to simultaneously optimize the weights w and the architecture encoding parameter α using gradient descent, SNAS relaxes the discrete one-hot random variable used to sample networks to a continuous random variable. However, this relaxation requires holding the whole parent network in memory, which prevents this approach from being applicable to large networks. As a solution, Hu et al. [20] propose a mathematical trick that yields a similar search process of networks in discrete space that involves holding only the sampled network in memory as opposed to the whole network.

Weight-Agnostic Neural Networks (WANNs)

WANNs are motivated by the observations of the precocial species whose young already possess certain abilities from the moment of birth such as the behavior of escaping from the predators. Using the same analogy, this work focuses on the possibility to learn the task at hand using the innate characteristics of the network architecture. In order to achieve this objective, it is important to de-emphasize the importance of weights from a network. This can be achieved by using a single value of weight for all the neurons in the network.

The NAS search for optimal WANNs is inspired by the well-established neuroevolution algorithm called NEAT (NeuroEvolution for Augmenting Topologies) [45] and is used for evolving the local generations in WFNAS. However, instead of simultaneously optimizing the network topology and its weights as is done in the conventional NEAT algorithm, only the topological search operators are considered and the weight parameters are ignored to produce weight-agnostic neural architectures. More particularly, all the networks use a randomly chosen shared weight value to discard the impact of the weight parameter in the optimization problem, making it simpler. The steps to perform the local search are outlined below:

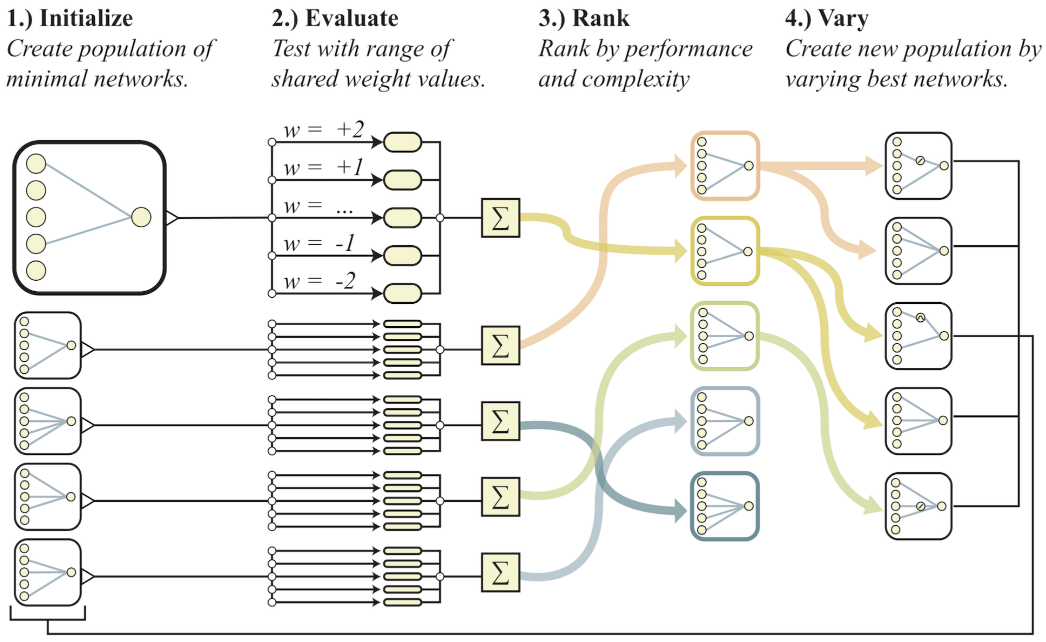

1.) Initialize
Create population of minimal networks.
2.) Evaluate
Test with range of shared weight values.
3.) Rank
Rank by performance and complexity
4.) Vary
Create new population by varying best networks.
w = +2
w = +1
w = ...
w = -1
w = -2
Σ

Figure 2.3: Overview of Weight-agnostic Neural Network Search [14]

1. Initialize a population of N networks that are minimally connected.

2. Evaluate these networks on 1000 samples randomly selected from the local training set, and assign the obtained reward to the corresponding network. The reward is nothing but the negative of the softmax cross-entropy loss value and the

task is to maximize the reward (i.e. minimize the loss). This step is repeated for 6 different weight values as follows: [-2.0, -1.0, -0.5, 0.5, 1.0, 2.0] and therefore, the reward for each network is a [1 x 6] array.

3. Based on the assigned reward for each network, rank these N networks on the basis of performance and complexity. More particularly, we use the following metric for ranking: i) *fitness* (average reward over the 6 different rollouts) (higher is better), ii) *fitMax* (maximum reward among the 6 different rollouts) (higher is better), and iii) *nConn* (the number of connections) (lower is better). In order to promote simpler networks, *fitness* and *nConn* are chosen with an 80% probability and *fitness* and *fitMax* are chosen with a 20% probability.

4. Discard the lower-ranking networks and evolve/vary the higher ranking networks using NEAT algorithm (described below). We use either of the three topological operators for mutation: *i) insertion of a node, ii) addition of a connection between previously unconnected nodes*, and *iii) changing the activation function of a hidden node.*

A visual representation of the above enumerated steps is provided in Fig. 2.3 borrowed from the original paper that proposed (non-federated) WANNs [14].

Overview of NEAT Algorithm

In this subsection, we provide an overview of the evolutionary algorithm (NEAT) using during the local search process [45] as represented in Fig. 2.4. The algorithm takes the parent population of N networks as input that need to be evolved/varied using the topological operators. Following that, the networks are ranked according to the Pareto Dominance (NSGA sort) using *fitness, fitMax* and the *number of connections.* As mentioned above, the factors are chosen for ranking using a probability.

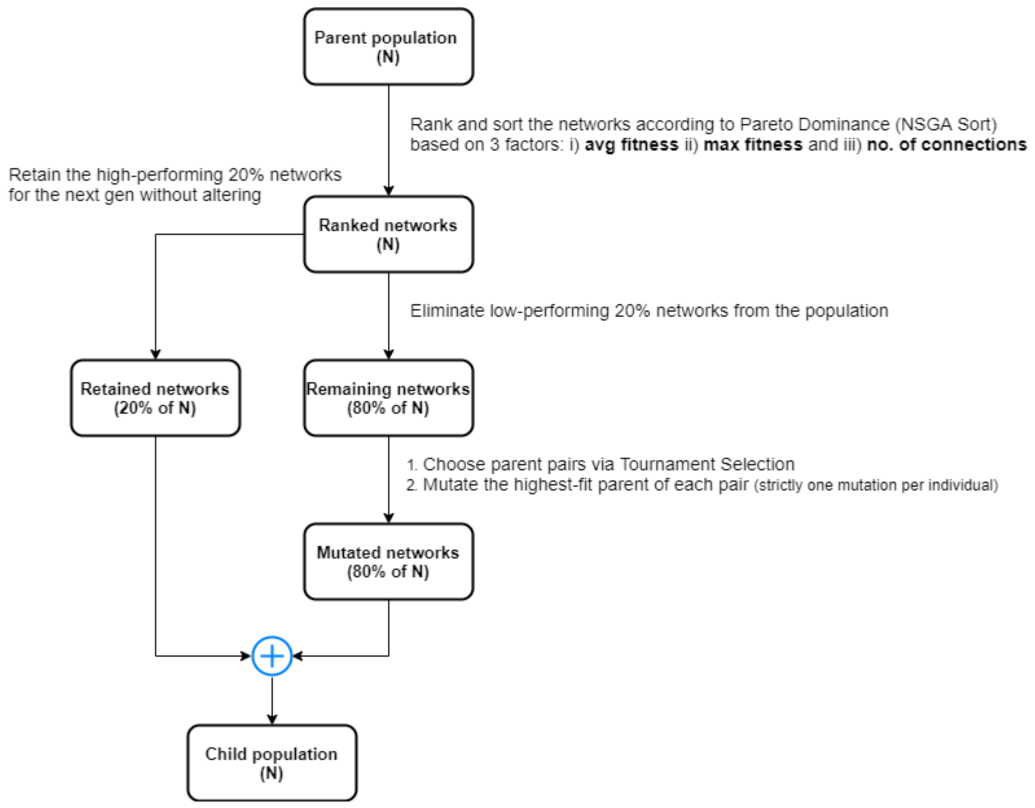

Parent population (N)
Rank and sort the networks according to Pareto Dominance (NSGA Sort) based on 3 factors: i) avg fitness ii) max fitness and iii) no. of connections
Retain the high-performing 20% networks for the next gen without altering
Ranked networks (N)
Eliminate low-performing 20% networks from the population
Retained networks (20% of N)
Remaining networks (80% of N)
1. Choose parent pairs via Tournament Selection
2. Mutate the highest-fit parent of each pair (strictly one mutation per individual)
Mutated networks (80% of N)
Child population (N)

Figure 2.4: Neat Algorithm

The elite 20% of the networks are retained (without any alteration) for the next generation and the last 20% of the networks are dropped. Mutation is performed on the remaining 80% networks (including the elite) chosen using Tournament Selection [32] to obtain new networks (strictly one mutation per individual) which are then combined with the retained elite networks to get the child population of N networks.

2.3 Federated Neural Architecture Search

The realm of Federated NAS is about efficiently searching for neural network architectures in an FL environment. This task is intrinsically more difficult due to the decentralization of data in an FL setting. In this section, we discuss current research in the literature on Federated NAS. Although WFNAS is an evolutionary approach, we mention and elaborate on both evolutionary and non-evolutionary state-of-the-art approaches.

Federated Deep Learning via Neural Architecture Search

He et al. [17] propose one of the pioneering works in Federated Neural Architecture Search, where the authors argue that since the data-distribution is invisible to the researchers in a federated learning environment, it is difficult to find the most suitable neural network that can work well on the machine learning problem at hand. That said, the authors promote the use of *Automating Federated Learning (AutoFL)* via NAS technique to help scattered workers collaboratively search for a better architecture with higher performance accuracy.

A cross-organization scenario is considered where each client in the network is a GPU-equipped edge server located in an organization. The authors then propose *FedNAS* to search for high-performance network architectures among the edge servers collaboratively. The search space considered in this work follows the mixed-operation search space defined in DARTS [29] and MiLeNAS [19], where the search is performed in two shared convolutional cells as shown in Fig. 2.5. After the search stage, the searched cells are stacked to build the entire model architecture. The search space is an overparameterized Directed Acyclic Graph (DAG) where all the possible forward connections are considered between any two nodes and the one with the highest weight is chosen to obtain the final architecture.

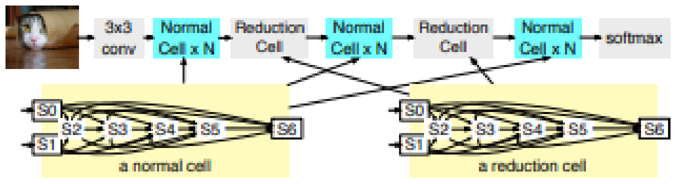


Figure 2.5: Search Space Considered in Fednas [17]

MiLeNAS [19] is used to perform the local search at the client level as it is efficient in terms of search time and can be easily distributed. Each client searches locally by utilizing the mixed-level optimization technique MiLeNAS as follows:

$$w = w - \eta_w \Delta_w L_{tr}(w, \alpha)$$

$$\alpha = \alpha - \eta_\alpha (\Delta_\alpha L_{tr}(w, \alpha) + \lambda \Delta_\alpha L_{val}(w, \alpha))$$

where, w represents the network weights and α is the neural architecture encoding parameter. $L_{tr}(w, \alpha)$ and $L_{val}(w, \alpha)$ denote the training and validation loss respectively.

Following the local search, each client sends its updated w and α to the server for weighted aggregation. The central server aggregates the gradients received from all the participating clients as below:

$$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{N_k}{N} w_{t+1}^{k} \qquad \alpha_{t+1} \leftarrow \sum_{k=1}^{K} \frac{N_k}{N} \alpha_{t+1}^{k}$$

where, N_k is the number of samples with client k and N is the total number of samples. The server then sends the updated w and α to the clients for the next round of searching.

The experiment setup used for this work uses a distributed computing network of 17 nodes equipped with GPUs, where 16 nodes are clients and the remaining node is the central server. The authors use the CIFAR-10 [24] dataset and generate a non-IID data distribution by randomly splitting the data among the clients using the *Dirichlet Distribution*. The experimental results indicate that the architecture searched by FedNAS obtains a test accuracy 4% higher than FedAvg [31]. A shortcoming of FedNAS is that it is a two-stage process: *search* and *parameter retraining*, which is very time consuming. Merging the search and evaluation into one stage is a potential avenue for future work.

Differentially-private Federated Neural Architecture Search

On top of *FedNAS* discussed above, Singh et al. [44] propose a *differentially-private* federated learning algorithm for *neural architecture search* supported by theoretical guarantees for privacy preservation. Experimental results demonstrate the capability to search for highly-performance neural network architectures while preserving the privacy of individual clients. The primary motive of the paper is to address the problem of potential privacy violations of user's sensitive data with an argument that although the federated neural architecture search does not directly use the client's data, studies [4] have shown that intermediate results such as gradients if leveraged cleverly are capable of revealing private information in federated learning.

The problem formulation for the standard *Federated Neural Architecture Search* (without using differential privacy) has been shown in the paper as below:

$$\begin{aligned} \min_A \quad & \sum_{k=1}^{K} L(D_k^{(val)}, \alpha, w^*(\alpha)) \\ \text{s.t.} \quad & w^*(\alpha) = \operatorname{argmin}_w \sum_{k=1}^{K} L(D_k^{(tr)}, \alpha, w) \end{aligned}$$

where, $D_k^{(tr)}$ and $D_k^{(val)}$ denote the training and validation dataset belonging to client k, α denotes the architecture, w denotes the weights of the model whose architecture is α, w^* denotes the optimal weights of architecture α on training data and L denotes the loss function. Given a configuration α of the architecture, the authors use the gradient-based federated learning approach to learn the optimal weights $w^*(\alpha)$ on the training data and then evaluate $w^*(\alpha)$ on the validation data of each client and aggregate the evaluation results. The validation performance is used to select the best architecture. It is noteworthy that the authors have used the idea of differentiable NAS as proposed in [29], where an overparameterized cell/network is considered with all the possible forward operations between any two nodes, which are optimized together with the weight parameters to achieve the best performance on the validation set.

Now, to address the problem of privacy discussed earlier, the authors use *Differential Privacy* that guarantees that the log-likelihood ratio of the outputs of the algorithm under two databases differing in a single individuals data is smaller than ϵ (a parameter to measure the strength of privacy). For this work, the authors add random noise drawn to the gradient updates of each client from a univariate Gaussian distribution.

The authors use the CIFAR-10 [24] dataset to perform experiments using the proposed algorithm. The search space similar to [17] is considered. During the search phase, the training and validation are split equally (i.e. 25k for training and 25k for validation). Once the search is completed, the searched cell is stacked multiple times to form a neural network and retrained using all the 50k samples as the training set. The experimental results indicate the although the proposed approach performs slightly worse than *FedNAS* [17] for the same number of clients, it is able to provide more privacy guarantees.

Direct Federated Neural Architecture Search

A major shortcoming of approaches like *FedNAS* [17] is that it utilizes a two-stage complex process of searching followed by parameter retraining making the entire process very computationally expensive and infeasible for a real-life federated learning environment. Garg et al. [15] propose *Direct Federated Neural Architecture Search*, a one-stage hardware agnostic computationally lightweight method to search for ready-to-deploy neural network models. The main contributions of this work can be summarized as below:

1. First work in the literature to combine federated NAS for both *cross-silo* and *cross-device* federated learning.
2. An efficient plug-and-play solution to search for ready-to-deploy network in the federated setting.
3. For the same number of communication rounds, the proposed approach achieves an average test accuracy improvement of 10% compared to the predefined models with FedAvg [31].

Instead of searching for the best operations in an overparameterized cell and then retraining the stacked architecture, the authors propose to sample the full network architectures and represent the search space with a set of one-hot random variables as shown below in Fig. 2.6 that are multipled as a mask to select operations in the graph. Child networks are sampled from the parent network, which discards the need to hold the entire parent network in memory for training.

The local objective is to find the best architecture α to optimize the expected performance on the local data. Mathematically representing the local objective:

$$\mathbb{E}_{Z \sim p_\alpha(Z)}[\mathcal{L}_w(x_i, y_i, Z)]$$

where, Z is a matrix of all structural decisions, α is the architecture encoding, $p_\alpha(Z)$ is the distribution of architectures and $\mathcal{L}$ is the loss function calculated using the

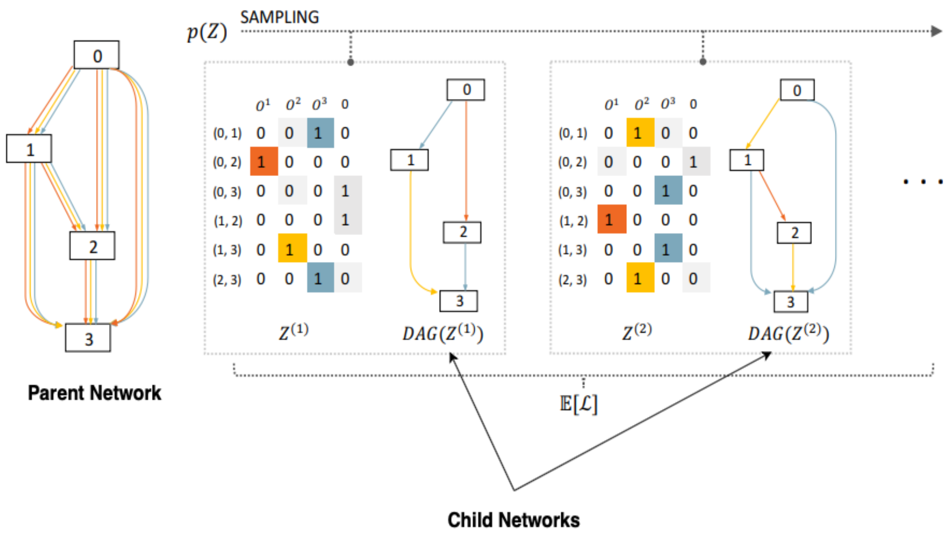
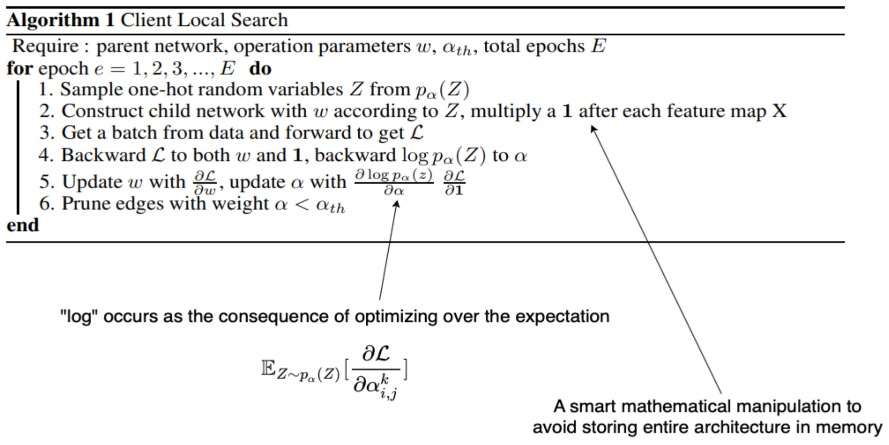


Figure 2.6: Visualization of Sampling Child Networks from a Parent Network [15]

data sample (x_i, y_i) for architecture represented using Z. The global objective is to maximize the predictive performance by collectively learning model parameters w and architecture α.

The architecture search technique used in this paper is based on DSNAS [20]. For every epoch, a child network represented by a matrix of randomly generated one-hot vectors is sampled and a forward pass using the local data is performed to obtain the loss L. The gradients for the weight and architecture parameters are calculated using the formula shown in Fig. 2.8. The operation with the highest architecture parameter is selected between any pair of nodes.

In *Direct Federated NAS*, uses the algorithm in Fig. 2.7 to find a ready-to-deploy network using the algorithm defined in Fig. 2.8. For every round, K clients are chosen from the pool of available clients and the local search is performed at each client. Following this, each client sends their updated w and α for aggregation. The server aggregates the client updates and sends the updated w and α to the newly selected K clients and the process repeats until convergence.



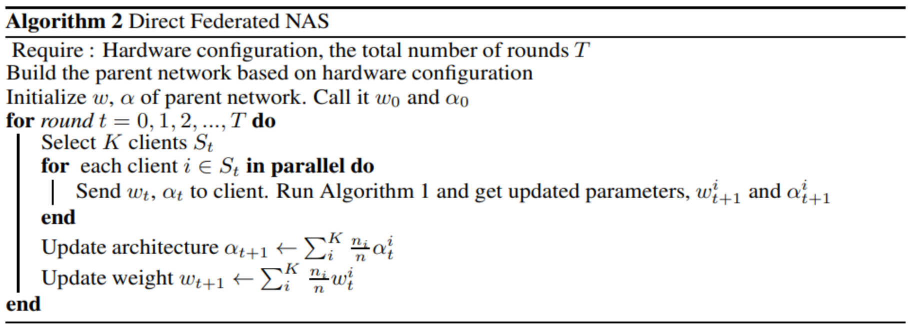
```
Algorithm 1 Client Local Search
Require : parent network, operation parameters $w$, $\alpha_{th}$, total epochs $E$
for epoch $e = 1, 2, 3, ..., E$ do
    1. Sample one-hot random variables $Z$ from $p_\alpha(Z)$
    2. Construct child network with $w$ according to $Z$, multiply a $\mathbf{1}$ after each feature map X
    3. Get a batch from data and forward to get $\mathcal{L}$
    4. Backward $\mathcal{L}$ to both $w$ and $\mathbf{1}$, backward $\log p_\alpha(Z)$ to $\alpha$
    5. Update $w$ with $\frac{\partial \mathcal{L}}{\partial w}$, update $\alpha$ with $\frac{\partial \log p_\alpha(z)}{\partial \alpha} \frac{\partial \mathcal{L}}{\partial \mathbf{1}}$
    6. Prune edges with weight $\alpha < \alpha_{th}$
end
```

"log" occurs as the consequence of optimizing over the expectation

$$\mathbb{E}_{Z\sim p_\alpha(Z)}[\frac{\partial \mathcal{L}}{\partial \alpha_{i,j}^k}]$$

A smart mathematical manipulation to avoid storing entire architecture in memory

Figure 2.7: Algorithm to Perform Local Search at the Clients [15]



```
Algorithm 2 Direct Federated NAS
Require : Hardware configuration, the total number of rounds $T$
Build the parent network based on hardware configuration
Initialize $w$, $\alpha$ of parent network. Call it $w_0$ and $\alpha_0$
for round $t = 0, 1, 2, ..., T$ do
    Select $K$ clients $S_t$
    for each client $i \in S_t$ in parallel do
        Send $w_t$, $\alpha_t$ to client. Run Algorithm 1 and get updated parameters, $w_{t+1}^i$ and $\alpha_{t+1}^i$
    end
    Update architecture $\alpha_{t+1} \leftarrow \sum_i^K \frac{n_i}{n} \alpha_t^i$
    Update weight $w_{t+1} \leftarrow \sum_i^K \frac{n_i}{n} w_t^i$
end
```

Figure 2.8: Direct Federated Nas Algorithm [15]

The authors perform extensive experiments for the cross-silo and cross-device federated learning settings using CIFAR-10 [24] and CINIC-10 [8] (an extended version of CIFAR-10) datasets. There are 3 and 4 candidate blocks for cross-device and cross-silo respectively and they are stacked in the original architecture 20 times, which means that the search space comprises of 3^{20} and 4^{20} architectures for cross-device and cross-silo respectively. Results indicate a superior performance compared to *FedNAS* [17] with a significant reduction in total time to obtain a ready-to-deploy network. At the same time, the parameter size of the searched network is increased significantly.

Multi-objective Evolutionary Federated Learning

Besides the gradient-based federated NAS techniques discussed above, several evolutionary federated NAS approaches have been proposed in the literature. Zhu et al. [58] propose to optimize the structure of the neural networks (Multi-layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs)) using a multi-objective evolutionary algorithm to minimize the communication as well as improve the performance accuracy. The proposed federated NAS framework is a typical offline approach because the search and the weight-training happens sequentially. The multi-objective federated NAS algorithm can be summarized as follows:

1. Initialize a parent population of N networks.
2. Evolve/Vary the parent population by performing genetic operations on the networks like crossover and mutation to obtain the child population of networks.
3. Send the entire child population to all the participating clients for fitness evaluations in federated learning on all their local data.
4. Combine the parent and the child population together and select the best N networks to obtain the new parent population of networks using environmental selection.
5. Repeat steps 2 to 4 until convergence.
6. Using the best network obtained after step 5, train the weights in federated learning by using aggregation techniques like FedAvg [31].

Since the NAS evolutionary algorithm is a two-step process, the overall communication overhead of this approach is quite significant. Also, the idea of sharing all the networks from the central entity to all the participating client nodes may be improved by using a sampling technique.

Real-time Federated Evolutionary Neural Architecture Search

Addressing the shortcomings of the offline evolutionary federated NAS approach discussed above, Zhu et al. [59] propose a novel real-time federated evolutionary NAS technique with a motivation to reduce the uniquely high demands of communication resources of federated learning caused primarily due to the frequent and bulky exchange of model parameters between the server and the clients. The authors also mention that training and deploying deep learning workloads that require a large amount of computation power is unrealistic for edge devices typically used in federated learning environments.

In the proposed approach, the authors use a double-sampling technique that samples sub-networks or parts of a network from a predefined super-network. These sub-networks are then sent to the randomly samples clients based on their availability and connectivity without replacement. It is emphasized by the authors that this approach has two-fold advantages: i) As only a subset of the entire network is sent to the clients, the communication overhead is significantly reduced and ii) sampling clients without replacement allows duplication of efforts for training a sub-network. Furthermore, since each client has a different sub-network, the weighted aggregation of updates from the client as done in [31] is not possible. Therefore, the global models for each client are reconstructed by filling in the sub-networks that are not updated by the respective client. After reconstructing the global model, they are easily aggregated as each client now has the same global model (with different weight and architecture parameters).

Extensive experiments are conducted using five different datasets - CIFAR10 [24], CIFAR100 [25], SVHN [33], Pathmnist [55], and Tiny ImageNet [9]. The results indicate that the performance accuracy is comparable to state-of-the-art baseline models with a much simpler network in terms of computational complexity.

Chapter 3

IN SEARCH OF WFNAS

In this chapter, we discuss several experiments that we performed in search of our proposed approach - WFNAS. Every subsection of this chapter deals with a question about a particular part of the proposed approach and talks about the findings that justify our choices and led to the creation of the final WFNAS.

3.1 Dataset

We perform experiments on the MNIST benchmark dataset [27] consisting of images of handwritten digits in order to have a fair comparison with the base (non-federated) approach (WANNs [14]). The MNIST dataset consists of $60,000$ training samples and $10,000$ testing samples for handwritten digits from 0 to 9 (10 classes). All the images in the MNIST dataset are [28 x 28] grayscale (black and white) images. We did not consider other datasets for experimentation due to resource and time constraints.

3.2 Data Distribution among clients

For emulating a federated learning environment, we consider two types of data distributions in our experiments: i) *non-IID (non-identical and independent distribution)* and ii) *mutually exclusive division of data by classes.*

non-IID Data Distribution

We generate a non-IID data distribution of the MNIST dataset by splitting the $60,000$ training samples among the participating clients, indexed with k. Similar to

FedNAS [17], we use the *dirichlet distribution* to divide the data in an unbalanced manner. More particularly, for each client, we sample the distribution of proportions $p_c \sim \mathrm{Dir}_J(0.5)$ and allocate a $p_{c,k}$ proportion of the training samples of class c to the client k. An example of a non-IID data distribution used to perform the experiments for the 2-client setting is shown in Fig. 3.1.

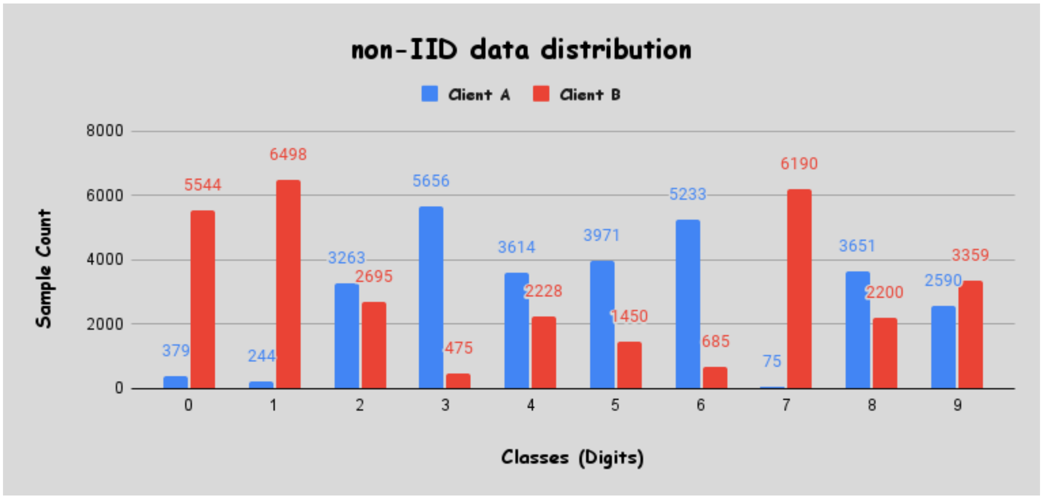


Figure 3.1: Example of Non-iid Data Distribution

Mutually Exclusive / Orthogonal Data Distribution

The MNIST dataset is considered fairly simple to learn as it consists of two-dimensional images of handwitten digits which are easy to distinguish. Therefore, the clients may learn how to predict the class correctly even with low number of samples without any exchange of networks in the case of non-IID data distribution. In order to prove the effectiveness of sharing networks using our proposed approach, we also consider the extreme case of data distribution which is dividing the entire dataset by classes in a mutually exclusive manner. In particular, we consider a 2-client setting, where one client has the data for all the even digits and the other client

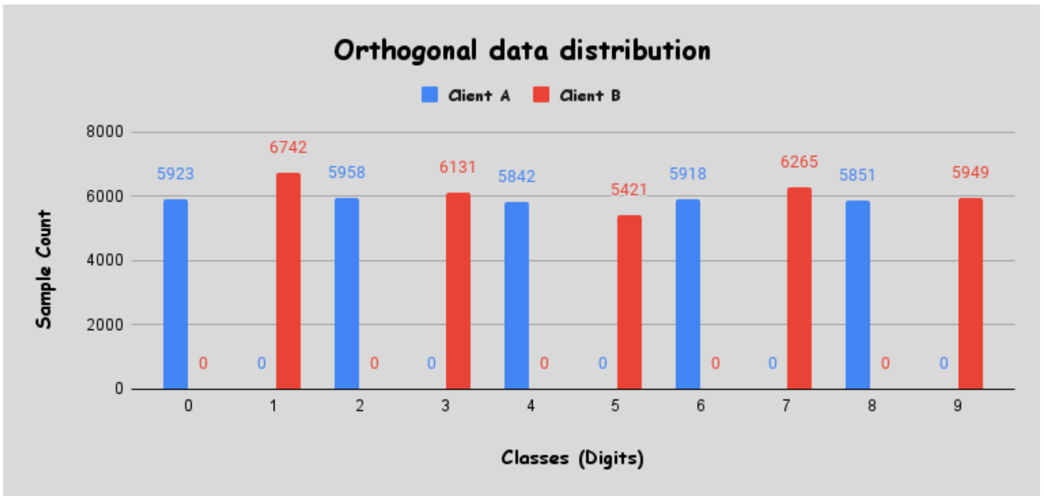


Figure 3.2: Example of Orthogonal Data Distribution

has the data for all the odd digits as shown in Fig. 3.2.

3.3 Trivial exchange of networks

In order to implement a federated version of the WANNs, we began by experimenting with a simple exchange of networks with an expectation that merely sharing the locally evolved networks might help. So, each client would do the following: *evolve the networks locally (based on the local data) and at the time of the global exchange or the sharing period, the clients simply exchange a certain percentage of their locally evolved networks with each other. Following the global exchange, all the networks are ranked based on the local data, and the best N networks are chosen for the next generation.*

However, in the case of orthogonal data distribution, it was observed that the clients only learned the data on which they are trained locally and fail to perform on the other client's data despite mixing the networks. It was speculated that the trivial mixing of networks gave no chance to the networks that do well on the other data regardless of the percentage of networks shared. This indicated that there was a need for networks to be validated on the other client's data. Since the data cannot leave the client's device in a federated learning setting, the other alternative was to ask the client to validate the networks itself.

3.4 Estimation of rewards

One of the primary purposes of the global exchange of networks between the clients is to validate the networks on the other clients' local data in order to ensure the convergence of local evolution on the entire training dataset. The most straightforward way to perform validation would be to send all the local networks (sharing percentage = 100%) from one client to all the other clients participating in the training process and receive the evaluation of those networks on their local data in return. However, sending all the networks for validation may not be feasible due to reasons such as increased communication overhead (growing in quadratic time for a two-client setting) and increased chances of privacy leak (learning information about the local data from the network topologies).

As a solution, we share only a certain percentage of the total local networks. Let the sharing percentage be denoted by p. Now, we use an estimator that can answer the following question: *Based on the performance of the networks that were shared with the other client on their local data, how would the networks that were not shared perform on the other client's data?* To the best of our knowledge, this problem has not been studied previously and our proposed approach for estimation is a novel contribution to the literature.

3.5 Comparison of estimators

In this section, we provide a detailed discussion and comparison of the various estimation techniques that were considered. At the end, we decide on the best features for the estimator to be used in WFNAS.

3.5.1 Mean

Since the networks in one generation do not differ by much and their rewards values do not have a huge variance, using the mean of the reward values as an estimate was the first approximation to evaluate the benefit of estimation. The idea is the following: *In each local iteration, client A evolves its own networks and evaluates them on its local data. However, to validate its networks on the local data of client B, A sends p% of network architectures to B and receives the evaluations for those p% networks in return. Now, to estimate the evaluations for the remaining (100 - p)% networks on the local data of B, the mean of the received reward values is used.* The same process is performed at Client B as well.

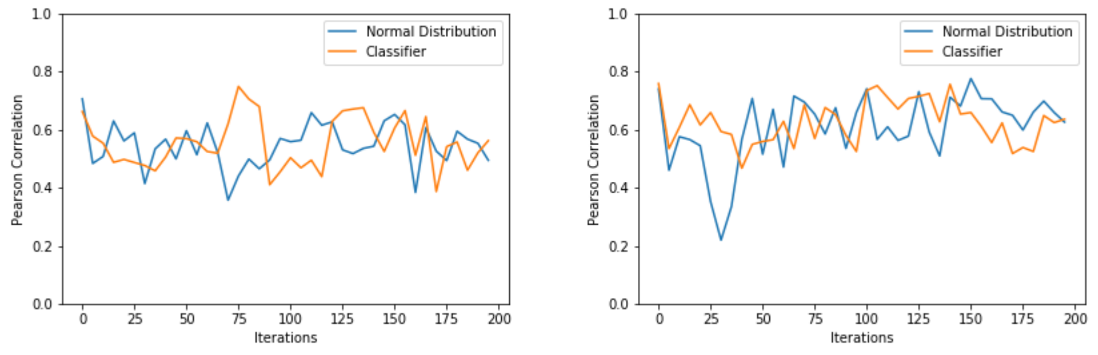
Using the mean as an estimate showed a slight improvement in the results when training with mutually exclusive (orthogonal) data distribution as compared to only local training, which indicated that estimation is beneficial and using a better estimate can help reduce the number of networks that need to be shared for validation.

3.5.2 Normal Distribution

Using the normal distribution is another cheap estimation technique we used. In this approach, the mean and the standard deviation of the received reward values are evaluated which are then used to fit a normal distribution. Following that, to estimate how the remaining networks would perform on the other client's data, random values are drawn from that normal distribution. This technique demonstrated better experimental results than using the *mean* as an estimator.

3.5.3 Hand-crafted features for regression model

The first two estimation techniques are weak. Instead, we needed a model that relates network characteristics to estimated performance on the other client's data. As an effort in that direction, we used a Decision Tree-based regression model [34] that takes as input two manually handcrafted features of the networks shared with the other client: i) *The average number of connections per node in each layer* and ii) *The number of nodes in each layer* as $\mathbf{X}$ and the reward values for these networks on the other client's data as $\mathbf{y}$. For the networks that are not shared, the same hand-crafted features are calculated and the reward values for these networks are predicted using the trained regression model.



(a) Client A (b) Client B

Figure 3.3: Normal Distribution Vs Regression Model Using Hand-crafted Features

We performed an experiment for the non-IID data distribution (20% sharing after every 10 local iterations) to compare the estimation strength of hand-crafted features for the regression model (labeled as "classifier" in Fig. 3.3) and the reward values estimated using the normal distribution. More particularly, the plots in Fig. 3.3 are values of Pearson correlation [50] of both the approaches with the ideal ranking (the true ranking of networks if all the networks were shared with the other client for validation). The results indicate that using the machine learning-based approach performs no better than random values drawn from a normal distribution and the reason for this behavior is insufficient variety of features. In addition, the overall training time increased significantly with this approach because of the time spent in creating the feature vectors. Therefore, this approach is not favorable to be used in the actually training workload.

3.5.4 Network Matrix as features

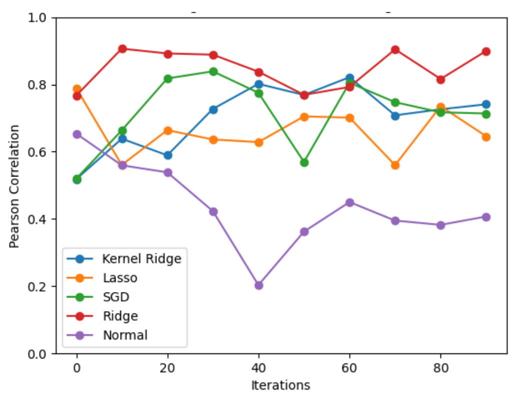
Each network is represented as an adjacency matrix A where the rows indicate the "connections from" and the columns indicate the "connections to". An entry a_{ij} in the matrix A is 1 if nodes i and j are connected and 0 if they are not connected. The idea of using the entire networks as features was based on the intuition that instead of trying to use hand-crafted features, maybe the classifier/estimator learns important information from the networks themselves. The matrix is flattened to obtain a vector, which is then used as the feature vector for the regression model.

This approach demonstrated a better performance compared to the estimation using manually hand-crafted features and is significantly faster. However, one drawback of this approach is that as the iterations progress, the networks become bigger and therefore the matrices consume more memory eventually creating *Out of Memory (OOM)* issues in our testing environment.

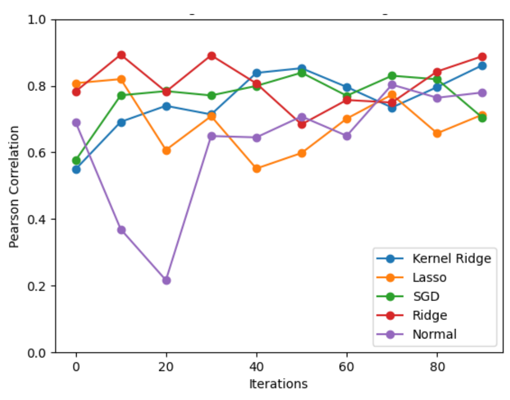
As a solution, we used the sparse matrix representation in *COO*rdinate format provided by the *scipy* [48] library in Python to represent the input features. The *COO*rdinate format essentially stores information about the indices having non-zero values in the original matrix using just three vectors: *row, column, data*, which significantly reduces the memory consumed by the matrix. Since the compression is lossless, we obtained the same performance with approximately **300x** less memory consumption.

Once we have the compressed features - **X_train** (*networks shared with the other client*) and **y_train** (*reward of shared networks on the other client's data*), they are used to train a regression model such as a *Support Vector Regressor* or a *Ridge Regressor*. Following that, in order to obtain **y_pred** (*prediction of the reward of networks not shared with the other client*), **X_test** (*networks not shared with the other client*) is provided as an input to the trained regression model.

We performed an experiment for the non-IID data distribution (20% sharing after every 10 local iterations) to evaluate the performance of different regression models (such as Kernel Ridge, Lasso, SGD, and Ridge) trained using the network matrices as features and compare it with the estimation accuracy of Normal Distribution. The plot in Fig. 3.4 is a comparison of the Pearson correlation (higher is better) [50] of rankings of networks obtained using different estimation techniques with the ideal ranking (the true ranking of networks if all the networks were shared with the other client for validation) after every global exchange. It is evident from the results that using the network matrix as a feature yields a better estimate of the rewards as compared to the normal distribution. Also, this study indicates that the choice of the regression model makes a considerable difference in performance. In this case, the "Ridge" regression model has superior performance than other regression models for both clients. The disparity in the performance of the estimators at both clients is due to the data distribution.



(a) Client A



(b) Client B

Figure 3.4: Normal Distribution Vs Different Models Using Network Matrix as Features

3.5.5 Rewards as features

The mean squared error (MSE) of the regression model with networks as features was still considerably high, given that our aim is to reduce the sharing percentage as much as possible. Therefore, we used the local reward values themselves as training features. More particularly, the *reward of shared networks on local data* will be the **X_train** and the *reward of shared networks on other client's data* will be the **y_train** to train the regression model. Now, to estimate *the performance of the networks that are not shared with the other client* (**y_pred**), the *reward of networks not shared on the local data* (**X_test**) are provided as an input to the trained model.

Network Matrix vs Rewards as features

The *mean, normal distribution* and *hand-crafted features* were considered in the very initial stage of this work and due to their shortcomings as mentioned in their respective subsections, they were not considered for future experimentation. However, these trivial estimators showed the way to more sophisticated estimation techniques using networks and rewards as features.

Now, in order to decide which among the networks and rewards is a better choice, we performed an experiment for the non-IID data distribution in a 2-client setting where both the clients share 20% of their networks after every 10 local iterations. During every exchange, the reward values were estimated using both the types of features and then compared to the ideal reward (true evaluations of the networks not shared with the other client). Mean-squared error (MSE) *(lower is better)* and R2-score (coefficient of determination) *(higher is better)* are used as metrics to evaluate the effectiveness of both the features that are fed as an input to the Support Vector Regressor.

The experimental results in Fig. 3.5, Fig. 3.6, Fig. 3.7, and Fig. 3.8 clearly show that using the *rewards* as features to train the regression model is better than using the *network matrix* as features. Furthermore, this study allowed us to be able to share a smaller percentage of networks with the other client reducing the overall communication overhead.

Client A: Network Matrix vs Rewards

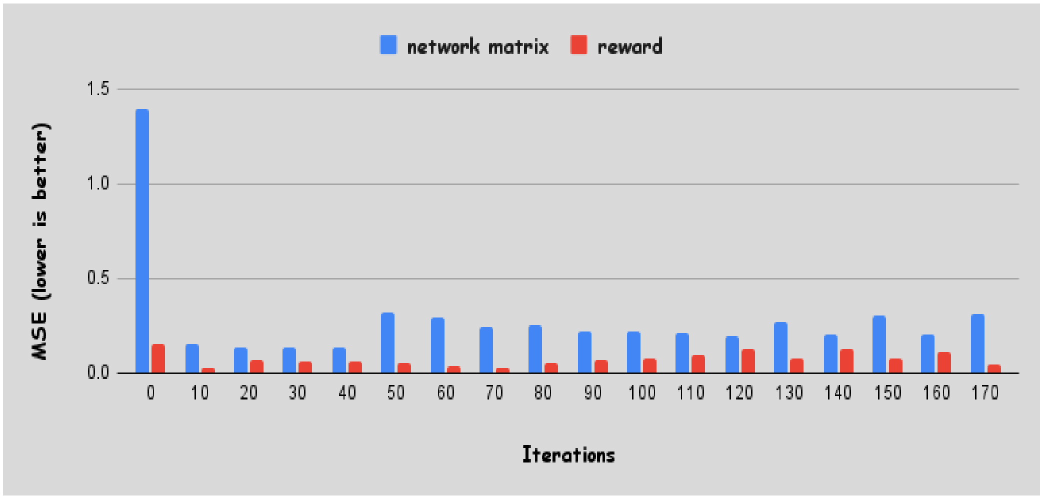


Figure 3.5: Mean Squared Error (Mse) (Client A)

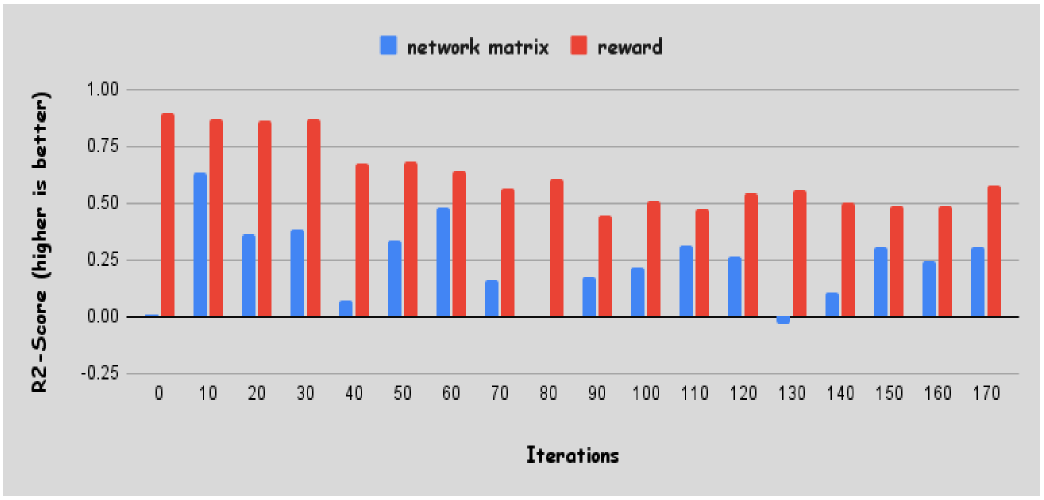


Figure 3.6: R2 Score (Client A)

Client B: Network Matrix vs Rewards

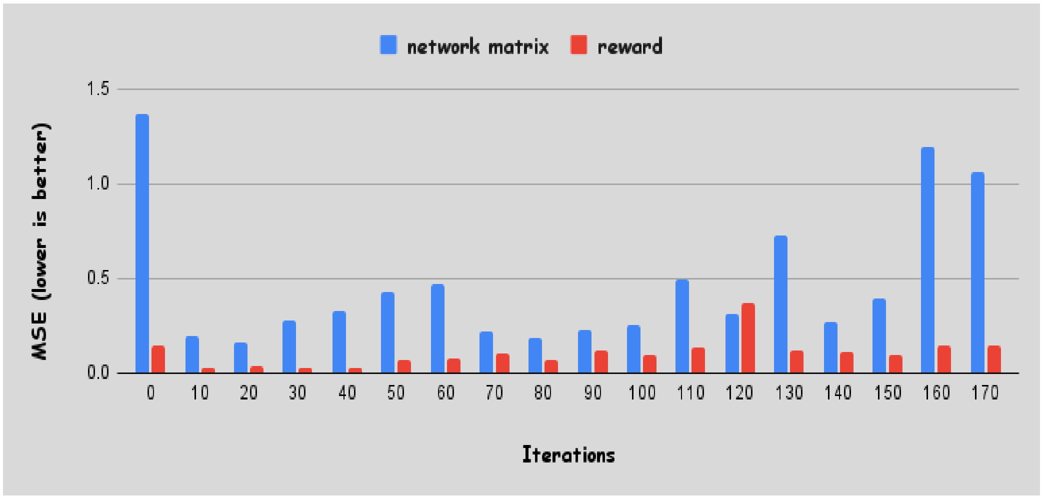


Figure 3.7: Mean Squared Error (Mse) (Client B)

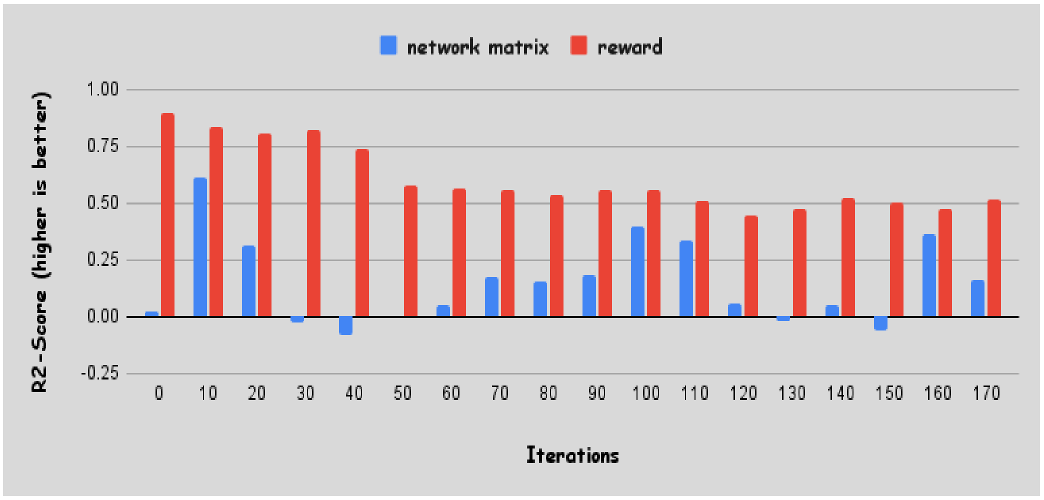


Figure 3.8: R2 Score (Client B)

3.5.6 Estimation Framework

A visual representation of the estimation framework used in the proposed approach is shown in Fig. 3.9.

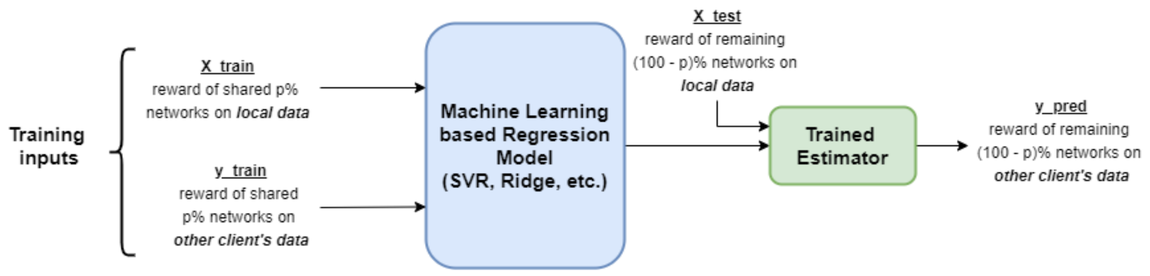

Training inputs
X_train
reward of shared p% networks on *local data*
y_train
reward of shared p% networks on *other client's data*
Machine Learning based Regression Model (SVR, Ridge, etc.)
X_test
reward of remaining (100 - p)% networks on *local data*
Trained Estimator
y_pred
reward of remaining (100 - p)% networks on *other client's data*

Figure 3.9: Estimation Framework

3.6 Weighted averaging of rewards

Following the above discussion of estimation of rewards, the next question that comes up is how to combine the evaluations on both the clients in order to rank the networks locally. The most straightforward way to evaluate a network is to consider a simple average of the network's evaluation on both the clients' data and use that reward while ranking the networks. For example, to evaluate the reward of network i, r_i in an K-client FL setting,

$$\mathrm{r}_i = \frac{1}{K}\sum_{k=0}^{K} r_i(k)$$

where, $r_i(k)$ represents the reward of network i on client k's data.

However, an assumption for the above case is that we are implicitly giving equal importance to the evaluation of all the clients. But, for instance, consider the case of non-IID data distribution (for example, the distribution shown in Section 3.2), where client A has only 75 samples of digit 7, whereas client B has 6190 samples of digit

7. Intuitively, it makes more sense to give more weight to the evaluation of client B compared to the evaluation of client A for digit 7. Therefore, we propose to use a per-class weighted reward as below:

$$\mathrm{r}_i = \frac{1}{C}\sum_{k=0}^{K}\sum_{c=0}^{C}\frac{K_{kc}}{Kc}r_i(kc)$$

where, C denotes the number of classes (in our case, 10), K_{kc} denotes the number of samples of class c with client k and $r_i(kc)$ represents the reward of network i on the data of class c on client k.

Note that the approach for weighted averaging assumes that each client has information about the number of samples with every other client, which indicates leakage of information. Empirical results demonstrated a higher performance accuracy with weighted averaging of rewards during the global exchange as compared to simple averaging.

A limitation of weighted averaging considered in this work that it is still a simple average of the number of classes. Perhaps, there should be a more sophisticated approach to weigh the rewards, which was not explored further in this work.

3.7 Choosing which networks to share?

At the time of the global exchange, $p\%$ networks are shared with the other client for validation and combination purposes. However, it is important to decide which networks to choose for sharing as they impact the estimation as well as the combination performed after the global exchange and therefore the overall performance. In order to investigate the matter, we performed a detailed analysis on how to choose the networks for sharing. Intuitively, it might seem that choosing the best-performing networks locally should be the way to go, but, our study indicates that choosing randomly from all over the population of networks is better.

Consider the experiment setup where each client shares 20% of networks chosen randomly from its entire population of networks after every 5 local generations. Now, the idea is to see whether the networks that survive at the other client after the exchange are from the group of high-performing local networks or from the group of low-performing local networks or a mix of both.

In order to do that, 20% of networks are chosen randomly, evaluated on the local data and both the networks and their evaluations are shared with the other client. The other client that receives the networks, evaluates them on its own local data and performs weighted averaging of the reward based on the number of samples per class with both clients to obtain the final reward. Using this reward, the networks are ranked and the best networks are chosen from the pool of its local networks and the imported networks (evaluated on both the client's data). We consider different bands of low and high-performing networks $(10\%, 20\%, 30\%, 40\%)$ for the sake of reliability of our results and calculate the following ratios:

- Ratio A: $\frac{\textbf{\# of survivors from low}}{\textbf{Total \# of survivors}}$
- Ratio B: $\frac{\textbf{\# of survivors from high}}{\textbf{Total \# of survivors}}$
- Ratio C: $\frac{\textbf{\# of survivors from low that were sent}}{\textbf{\# of low that were sent}}$
- Ratio D: $\frac{\textbf{\# of survivors from high that were sent}}{\textbf{\# of high that were sent}}$

The above ratios can serve as a metric to determine the distribution of the network ranks at one client that survive at the other client. The results shown in Table 3.1 and Table 3.2 are an average of the ratios obtained over 100 iterations (20 exchanges).

Low and High Rank % Band	**Ratio A**	**Ratio B**	**Ratio C**	**Ratio D**
10%	0.08	0.10	0.60	0.80
20%	0.16	0.20	0.75	0.91
30%	0.27	0.31	0.84	0.96
40%	0.36	0.43	0.79	0.92

Table 3.1: Client A: Experiment Results for Choosing Networks for Sharing

Low and High Rank % Band	**Ratio A**	**Ratio B**	**Ratio C**	**Ratio D**
10%	0.07	0.11	0.65	0.98
20%	0.16	0.22	0.70	0.91
30%	0.25	0.32	0.76	0.98
40%	0.35	0.44	0.74	0.95

Table 3.2: Client B: Experiment Results for Choosing Networks for Sharing

As the total # of survivors will be less than or equal to 20% (sharing percentage) of the total number of networks for this experiment, the ratios A and B are expected to increase with the increasing percentage of low and high networks considered due to randomness in selection of networks. Also, if either ratio C or D was consistently low for all the percentage bands, then it would indicate that considering networks from that percentage band is not a good idea. However, as evident from the above results, ratios C and D are consistently high which indicates that the networks that survive at the other end come from all over the population. This study lead us to adopt random selection for the shared networks.

3.8 Historical networks

As a part of the effort to understand areas of improvement in our approach, we observed that for the experiments with a configuration of sharing period greater than 1, we might potentially be losing out on good networks obtained during the local generations. Therefore, the idea of using *historical networks* was to retain a certain percentage of networks from every local generation and then at the time of global exchange, randomly choose networks for sharing from the pool of current generation of networks and the retained *historical networks*.

However, the approach did not show much promise based on the overall results and therefore the idea was eventually dropped for further experiments.

3.9 Separately evolving imported networks

For sharing period greater than 1, analyzing the performance of the best-performing networks of both the clients on individual classes indicated that sharing is not helping the classes that have a lower number of samples. An intuition for justifying the behavior was that the networks that were imported from the other client although they make it to the next generation based on the weighted validation of rewards, but eventually do not survive in the local evolution process as a certain percentage of networks are culled from every generation in the evolution process of the NEAT algorithm [45].

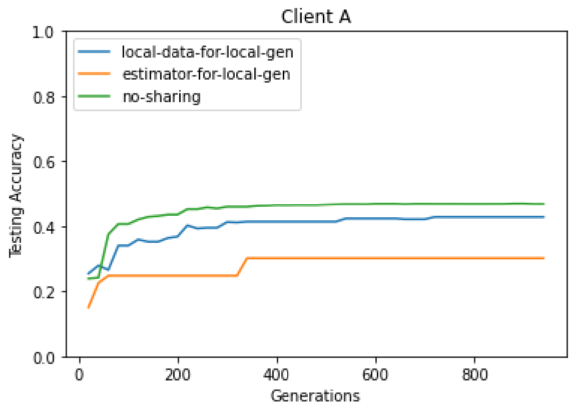
In order to make them survive intentionally, instead of choosing the best-performing networks right after the global exchange, an idea was to retain the imported networks and evolve them separately from the local networks using the local data. At the end of a set of local generations, the best-performing networks from the locally evolved local and imported networks are chosen to perform the global exchange. The expectation was that this will give a chance to the imported networks to evolve on the other client's data. It was expected that this approach can closely imitate the case where all the clients had access to the local data of all the other clients.

The results for this experimental approach indicated a higher overall performance accuracy on the test dataset, but it did not specifically help the classes having low number of samples. Therefore, this idea was dropped subsequently.

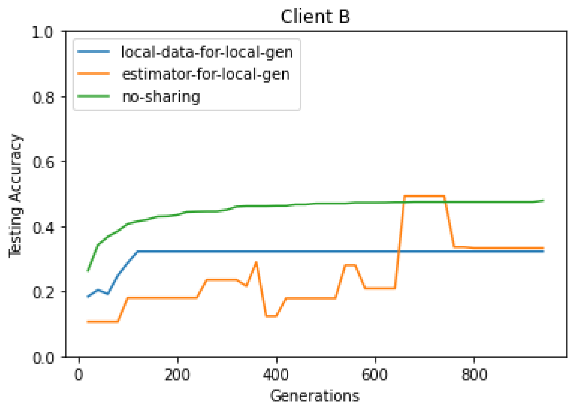
3.10 Estimator for Local Rewards

In order to reduce the total cost of communication between clients, it is necessary to increase the interval of exchange of networks between clients. At the time of the exchange, as discussed in Section 3.4, a machine learning based estimator is trained to predict the performance of the networks that are not shared with the other client on their data. The idea of this set of experiments was to utilize this trained estimator for prediction even during the local generations. For example, consider a configuration where both the clients exchange 20% of their networks after every 10 local generations. Now, after the clients perform an exchange during the 10th local generation, they have a trained estimator that is capable of predicting the performance of their local networks on the other client's data. Instead of evolving the networks by ranking them on the local data, the clients use this estimator to calculate the weighted average of the reward (as discussed in 3.6) to rank and evolve the networks until the next exchange. After the second exchange of networks during the 20th generation, the newly trained estimator is used for the next 10 local generations, so on and so forth.

The experimental results (also illustrated in Fig. 3.10) showed that the overall test



(a) Client A



(b) Client B

Figure 3.10: Using Estimator for Governing Local Evolution

performance reduced even in comparison to the evaluation only on local data during the local generations due to the following reasons: i) Since the reward values are used as features to train the estimator, it was observed that the estimator produces stale values that are outdated as the generations progress further and ii) the estimator itself is not perfect and relying on it beyond the global exchange seems to have limited benefit. Furthermore, the performance with and without estimator is worse than no-sharing indicating that a sharing period more than 1 needs further research.

3.11 Informed Sharing

The results from Section 3.7 indicated that randomly choosing networks to share with the other client is the best approach for two reasons: i) train the estimator well because of the variety of networks and ii) the experiments suggested that the survivor networks after the global exchange come from all over the population. These conclusions were made based on the ranking obtained from the local data of the client. However, the idea behind informed sharing is that it intuitively makes more sense to rank the data based on the entire dataset and then exchange the best-performing networks among the clients. Since the clients do not have access to each other's data, we perform two sets of exchanges as a part of the global exchange: i) randomly share p% networks to obtain the estimator and ii) rank the networks on the entire data using the estimator trained in the first exchange and share the best-performing q% networks. If the estimator is ideal, this approach seems more promising than random sharing in one exchange. A shortcoming of this approach is the increased communication overhead. However, this technique demonstrated a increased performance accuracy as expected and is therefore used in the proposed approach.

Chapter 4

WFNAS: THE PROPOSED APPROACH

Consider a federation of two clients A and B participating in a neural architecture search. Let the total number of generations/iterations be denoted by G and the interval at which the networks are exchanged be denoted by E. Therefore, the number of generations evolved locally before an exchange will be E and the total number of network exchanges will be $\frac{G}{E}$.

The steps of the proposed approach at Client A can be outlined as below. Similar steps are simultaneously performed at Client B.

1. Initialize a population of minimally connected networks at Client A.

2. For local generations (if $(g = 0)$ or $(g \bmod E) \neq 0$),

   (a) Evolve the networks of the current generation g and rank them based on the local data of Client A as shown in Fig. 2.3, however, instead of randomly choosing samples from the entire local dataset for evaluation of networks, choose randomly from each class and then average the evaluations on 10 classes. This averaged reward will be used to rank the networks for the next generation.

3. For global exchange of networks for validation (if $(g \bmod E) = 0$ and $g \neq 0$) (refer to Fig. 4.1),

   - *First Exchange*

     (a) Randomly select $p\%$ of the networks in the current generation g (where p is a parameter for the first exchange).

(b) Evaluate the selected and the remaining networks on each class of the local data of A.

(c) Send the selected networks with their per-class local reward to client B. Client B returns the per-class reward (evaluation) of shared networks of A on B's data.

(d) Similarly, client B sends $p\%$ of its networks and their per-class reward on B's data to A. Now, client A evaluates the received networks on local data of A and sends the per-class reward back to B.

(e) Using the evaluation of shared networks on both clients' data, train an estimator using the rewards per class as features for the regression model as discussed in Subsection 3.5.6. With the help of the trained model, estimate how the remaining (not shared) networks would perform on B's data.

(f) At this point, all the networks of the current generation have either an evaluation or an estimation for each class on both the client's data.

(g) Perform weighted averaging of rewards based on the number of samples per class with each client as discussed in Section 3.6.

- *Second Exchange*

(a) Rank the networks based on this weighted reward and send the best-performing $q\%$ networks to Client B (where q is a parameter for the second exchange).

(b) Similarly, client A will receive the best $q\%$ networks from client B.

(c) Rank the pool of networks (A's local + B's best $q\%$) again and choose the best-performing N networks for the next generation.

4. Repeat steps 2 and 3 for G generations.

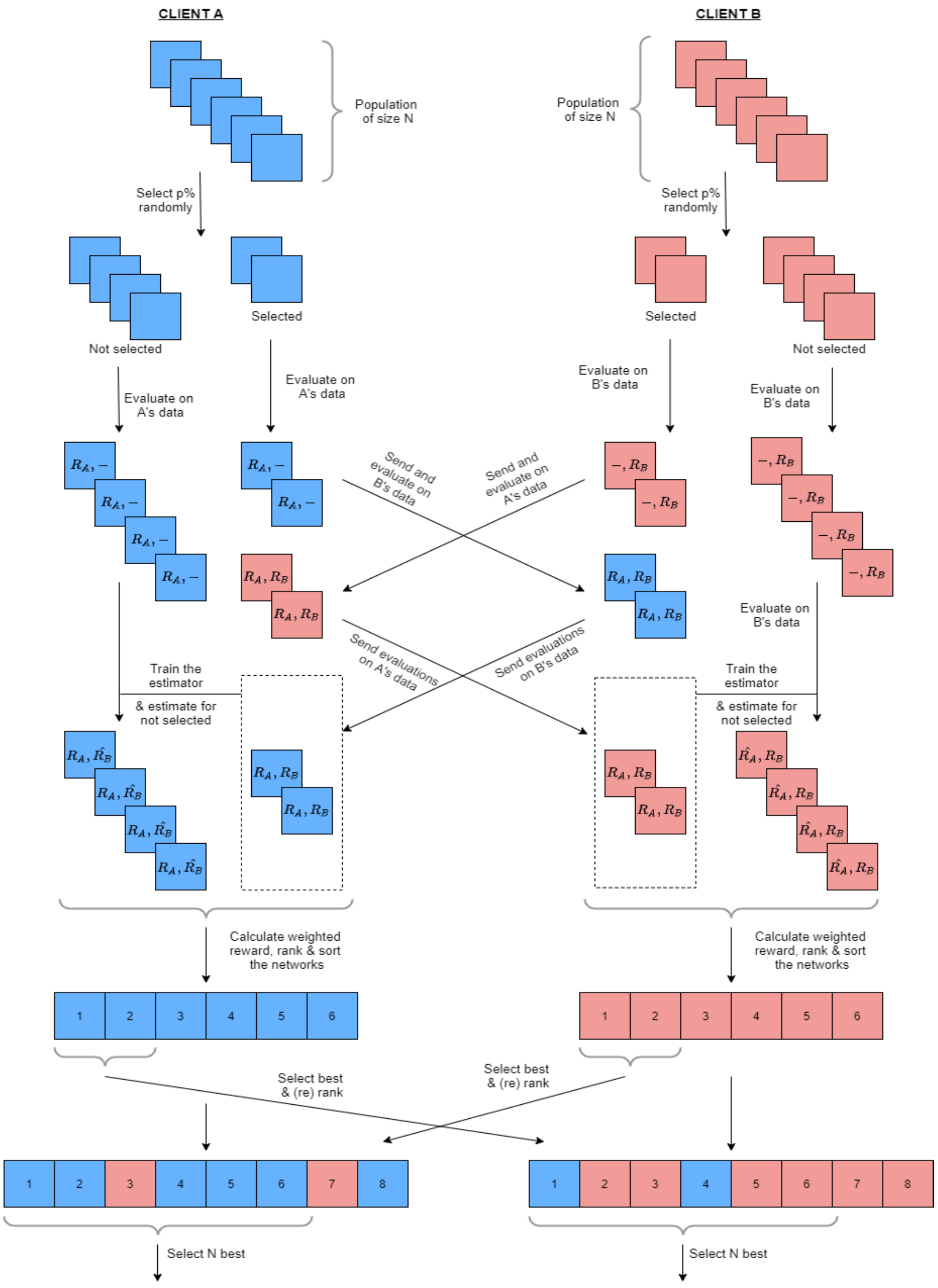


Figure 4.1: Exchange of Networks in Wfnas

Extending the proposed approach to an N-client FL setting

The proposed approach for a 2-client FL setting can be extended to an N-client FL setting by having each client exchange networks with every other client. Consequently, the estimator used to predict performance on the other client's data will be unique to each client and the final reward for ranking the local networks will be a weighted average of the reward obtained from all the clients based on the number of samples per class with each client. Note that with N clients, the number of exchanges scale quadratically - $O(n^2)$.

Chapter 5

EXPERIMENTAL RESULTS AND DISCUSSION

In this chapter, we provide a detailed discussion of the experiments that highlight the effectiveness of our proposed approach. We begin by describing the dataset and the experimental setup, followed by several experiments performed to understand the impact of various parameters of the proposed approach.

5.1 Dataset & Image Preprocessing

As the primary goal of this study is to explore the feasibility of weight-agnostic neural network search in a federated learning setting, we use the MNIST dataset (similar to Gaier et al. [14]) consisting of images of handwritten digits (0 to 9) so that we can compare our results to the non-federated setting (we did not consider other datasets due to time and resource limitations). The details of the MNIST dataset are already mentioned in Section 3.1.

0
5
10
15
20
25
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0

(a) Original Mnist Image [28 X 28]   (b) Modified Mnist Image [16 X 16]

Figure 5.1: Example Image from Mnist Dataset

For the sake of consistency with the non-federated WANNs [14], all the images in the MNIST dataset are downsampled by reducing the size of the images from [28 x

28] to [16 x 16] followed by the deskew operation using the OpenCV library [5] and pixel intensity normalization between 0 and 1. In Fig. 5.1, a visual representation of a sample image of digit 5 before and after performing resize, crop and deskew operations is shown.

We test the results of all the experiments using the MNIST testing dataset (after applying the same image pre-processing steps mentioned above). It contains a total of $10,000$ samples (see Table 5.1 for distribution of samples among classes).

Classes (Digits)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
# of samples	980	1135	1032	1010	982	892	958	1028	974	1009

Table 5.1: Mnist Test Dataset

5.2 Experimental Setup

In this study, we restrict our experiments to a two-client federated learning setting due to computational resource limitations. Let the two clients be denoted by “A” and “B”. As discussed in Chapter 4, each client evolves N networks per generation locally for G generations and performs two sets of exchanges of networks ($p\%$ and $q\%$ of networks respectively) after every E local generations.

We implemented the experimental setup in two ways: i) on a single node and ii) in a distributed setting. In i) both clients “A” and “B” execute in parallel using mpi4py [7] library in Python leveraging the various cores on a single node/machine. In ii) each node represents a client (similar to a federated learning setting) that performs evolution of its networks locally and the communication/exchange of networks is done between the nodes using *scp (secure copy protocol)* [51] in a synchronous manner.

5.3 Orthogonal data distribution

In this section, we consider a mutually exclusive division of the entire MNIST dataset among two clients as discussed in Section 3.2. Client A has the data of $29,492$ samples of all the even digits (0, 2, 4, 6, 8) and client B has the data of $30,508$ samples all the odd digits (1, 3, 5, 7, 9) (see Table 5.2 for per-class distribution of samples between clients). Since client A has zero samples of odd digits and client B has zero samples of even digits, there is no way for them to learn how to classify odd and even digits respectively. However, we show that our proposed federated NAS algorithm is capable of searching for networks that perform well on all the digits in the testing set. To the best of our knowledge, this is the first federated NAS technique proposed in the literature for vertical FL and hence we do not have any prior results to compare to. Therefore, we compare the results of our proposed approach with the results of non-federated standalone NAS of clients on their local data. In what follows, we present results that show the effect of various parameters in our proposed algorithm such as the sharing percentages and the sharing period.

Classes (Digits)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
# of samples with Client A	5923	0	5958	0	5842	0	5918	0	5851	0
# of samples with Client B	0	6742	0	6131	0	5842	0	6265	0	5949

Table 5.2: No. Of Samples per Class in Orthogonal Data-distribution

5.3.1 Effect of sharing percentages

As discussed in Chapter 4, the algorithm for WFNAS involves two sets of exchanges where $p\%$ of networks from the total population are shared with the other client in the first exchange and $q\%$ of networks are shared in the second exchange. The goal is to minimize the sum of p and q as it accounts for the total communication overhead of the proposed approach. First, we perform experiments by fixing the value of q to find a good value of p, and once we have the desired value of p, we vary the percentage of networks shared in the second exchange to find a good value of q.

The first exchange

The first exchange of networks is very crucial as the performance of the estimator depends on it. The higher the number of networks shared in the first exchange, the better will the estimator be, as the machine learning-based regression model will have more training data. At the same time, we also want to reduce it as much as possible to reduce the overall communication cost. We fix the value of q to 40% and vary the values of p to search for optimal values using an approach similar to the grid search [26]. Thus, we consider the following values of p - [5%, 10%, 20%, 40%] and exchange the networks after every generation ($E = 1$) for a total of 2000 generations, where each client has $N = 960$ networks per generation. The value of N is chosen as 960 for consistency with the non-federated version of WANNs. In addition, we compare the results of all the four different configurations of p with the case where the clients do not perform any exchange of networks and only search for networks using their local data.

The results in Fig. 5.2 indicate a consistent trend of better performance with a higher value of p for both clients A and B as anticipated. Also, in the case of no-sharing, both the clients perform well only on the digits for which they are trained

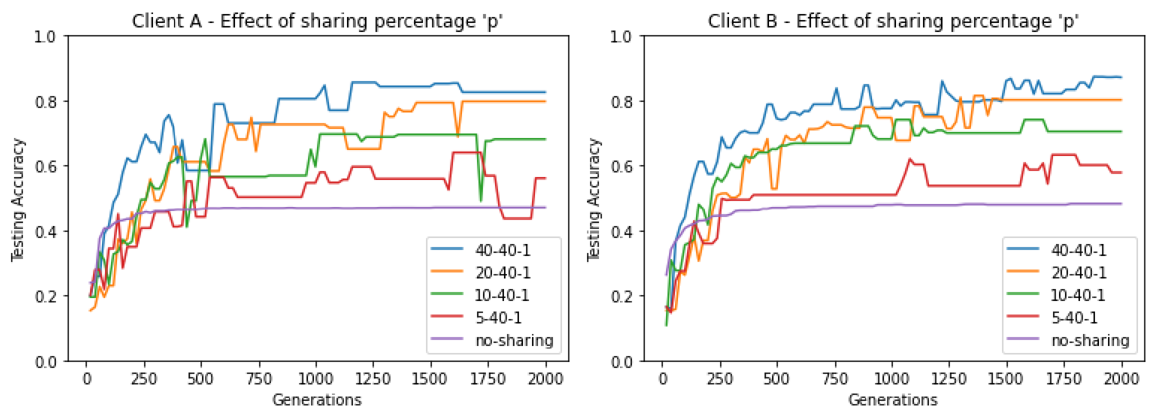


Figure 5.2: Effect of Sharing Percentage - p

locally, hence the test accuracy is less than 50%. On the contrary, the overall test performance for all the other configurations using our proposed approach is beyond 50% demonstrating the effectiveness of our algorithm. See Table 5.3 for reference.

Experiment Configuration (p-q-E)	**Test Accuracy**	
	Client A	**Client B**
no-sharing	47.09%	48.29%
5-40-1	56.12%	57.85%
10-40-1	68.08%	70.44%
20-40-1	79.67%	80.13%
40-40-1	**82.48%**	**87.08%**

Table 5.3: Comparison of Test Accuracy for Different Values of p

The second exchange

The second exchange is contingent on the performance of the estimator trained using the first exchange of networks, which requires choosing an optimal value of p. As the local networks shared in the second exchange will be mixed with the population of networks on the other end, it is expected that the higher the value of q, the better the overall performance will be. We fix the value of $p = 20$ as it seems to balance the trade-off between performance accuracy and communication overhead from Fig. 5.3 and vary the value of q as follows - [5%, 10%, 20%, 40%]. The values of other parameters are kept the same as earlier ($N = 960$, $G = 2000$, $E = 1$).

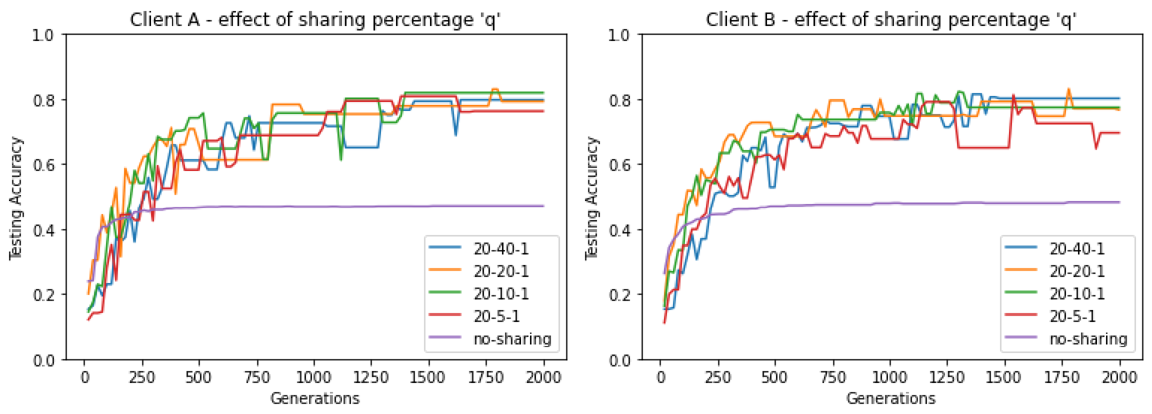


Figure 5.3: Effect of Sharing Percentage - q

Experiment Configuration (p-q-E)	**Test Accuracy**	
	Client A	**Client B**
no-sharing	47.09%	48.29%
20-5-1	76.21%	69.56%
20-10-1	81.84%	77.36%
20-20-1	79.19%	76.65%
20-40-1	79.67%	80.13%

Table 5.4: Comparison of Test Accuracy for Different Values of q

As evident from Fig. 5.3, in general, there is an increasing trend in the overall testing accuracy of both clients by increasing the sharing percentage q. In Table 5.4, we summarize the results obtained by varying the sharing percentage q.

5.3.2 *Effect of sharing period*

The motivation to study the effect of sharing period is that the higher the sharing period (E), the lower will the frequency of exchange of networks between the clients be, and thus, the communication cost will be lower. For orthogonal data distribution, evolving networks locally for 5 generations (using their local data) and then exchanging networks for validation at every 5th generation results in a significant drop in overall performance even below the baseline case of no-sharing as shown in Fig. 5.4 and Table 5.5.

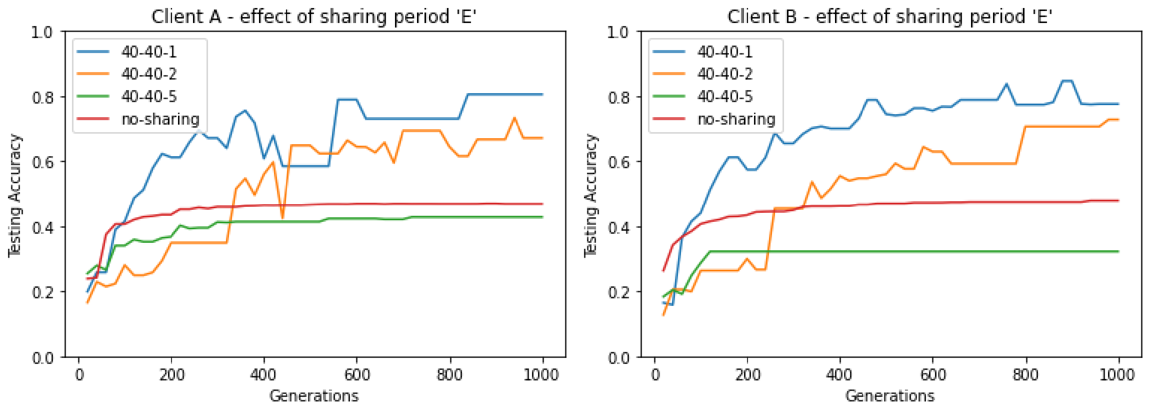


Figure 5.4: Effect of Sharing Period - E

Our intuitive explanation for this phenomenon is that after the exchange of networks, the imported networks do not survive in subsequent local generations. For example, after importing 40% of networks from client B (having data of odd digits) at client A (having data of even digits), the imported networks never get a chance to be evaluated on the data of odd digits for the next 5 local generations, which is why

Experiment Configuration (p-q-E)	**Test Accuracy**	
	Client A	**Client B**
no-sharing	47.09%	48.29%
40-40-1	82.48%	87.08%
40-40-2	67.11%	72.76%
40-40-5	42.77%	32.21%

Table 5.5: Comparison of Test Accuracy for Different Values of E

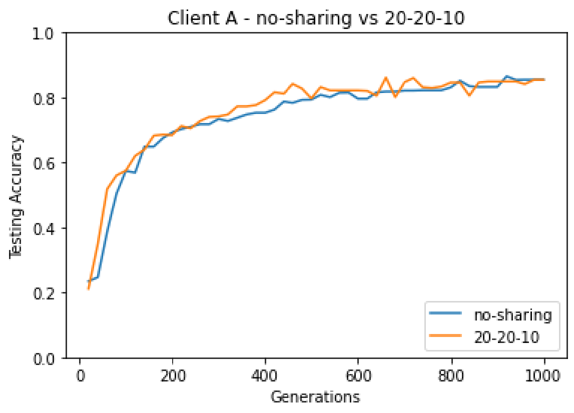
there is no way for them to survive until the next exchange.

5.4 non-IID data distribution

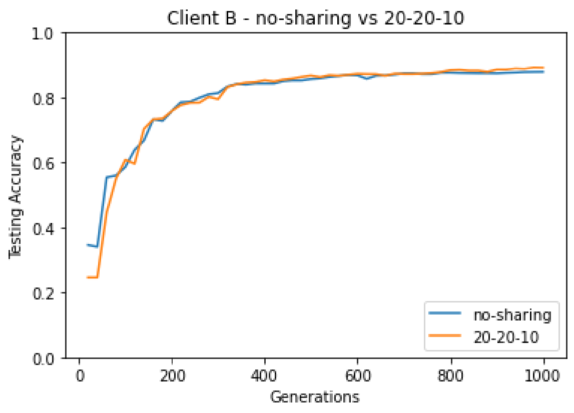
In this section, we discuss the experimental results when the data distribution among the clients is non-IID. Using Dirichlet distribution [49], we generate a non-IID data distribution (see Table 5.6 for per-class distribution of samples between clients) to perform experiments with our proposed approach where client A has a total of $28,676$ samples and client B has a total of $31,324$ samples.

Classes (Digits)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
# of samples with Client A	379	244	3263	5656	3614	3971	5233	75	3651	2590
# of samples with Client B	5544	6498	2695	475	2228	1450	685	6190	2200	3359

Table 5.6: No. Of Samples per Class in Non-iid Data-distribution



(a) Client A



(b) Client B

Figure 5.5: No-sharing Vs 20-20-10 for Non-iid Data Distribution

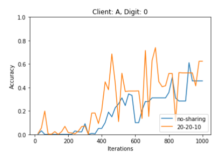
Experiment Configuration (p-q-E)	**Test Accuracy**	
	Client A	**Client B**
no-sharing	85.38%	87.81%
20-20-10	85.36%	**89.16%**

Table 5.7: Comparison of Test Accuracy for Non-iid Data Distribution

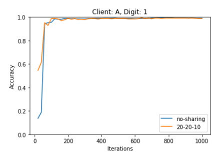
As evident from Table 5.6, there are several classes with high disparity in the distribution of data such as digit 7 with only 75 samples with client A and 6150 samples with client B. However, despite having lower number of samples for certain classes at both clients, we observe that the clients are able to search for well-performing networks with a test accuracy similar to non-federated WANNs even without requiring any exchange of networks. Figure 5.5 and Table 5.7 show a comparison for two configurations: i) no exchange of networks and ii) first exchange of 20% of networks ($p = 20$) and second exchange of 20% of networks ($q = 20$) between clients after every 10 local generations for 1000 generations ($G = 1000$) is performed using non-IID data distribution. Even though the baseline (standalone training) performance with MNIST is quite well, we see a gain of 2% in the overall test accuracy with WFNAS.

To examine the reason for a relatively smaller gain compared to vertically distributed data, we analyzed and compared the per-class test accuracy of both configurations as shown in Fig. 5.6 and Fig. 5.7 for clients A and B respectively. As evident from all the plots in Fig. 5.6 and Fig. 5.7, despite a high disparity in the number of samples for different classes between both the clients, the testing accuracy for each class in both "sharing" and "no-sharing" configuration is similar. This phenomenon is explained by the simplicity of the MNIST dataset, where a model can trained with a low number of samples.

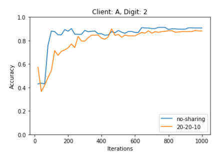
Client A



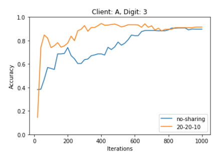
(a) Digit 0



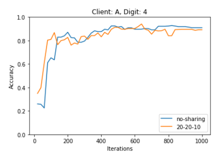
(b) Digit 1



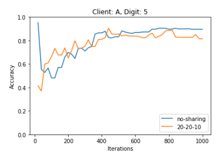
(c) Digit 2



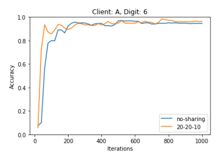
(d) Digit 3



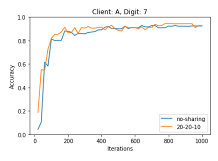
(e) Digit 4



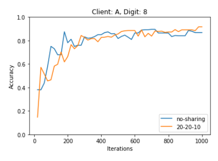
(f) Digit 5



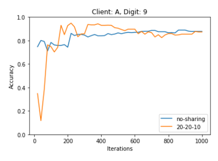
(g) Digit 6



(h) Digit 7



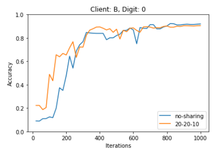
(i) Digit 8



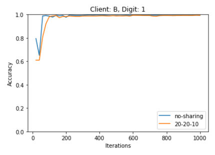
(j) Digit 9

Figure 5.6: Comparison of Per-class Test Accuracy: No-sharing Vs 20-20-10 Sharing (Client A)

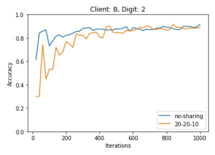
<u>*Client B*</u>



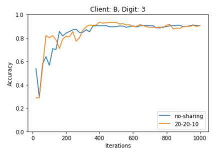
(a) Digit 0



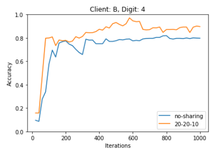
(b) Digit 1



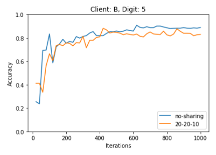
(c) Digit 2



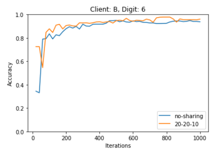
(d) Digit 3



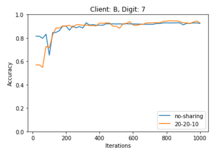
(e) Digit 4



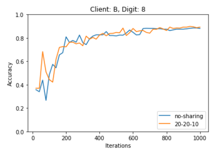
(f) Digit 5



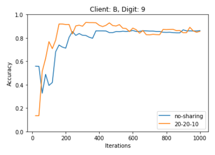
(g) Digit 6



(h) Digit 7



(i) Digit 8



(j) Digit 9

Figure 5.7: Comparison of Per-class Test Accuracy: No-sharing Vs 20-20-10 Sharing (Client B)

5.5 Communication Overhead

In this subsection, we provide an analysis of the communication overhead of our proposed approach. For a 2-client FL setting, consider an experiment configuration in which 40% of networks are shared in the first exchange ($p = 40$), 40% of networks shared in the second exchange ($q = 40$) and the exchange happens after every local generation ($E = 1$). The total number of networks at each client is 960 ($N = 960$). Although the size of the networks grows during the evolution process which is smaller than the size of the champion (final best-performing) network, we calculate the communication cost per exchange under the assumption that the clients always exchange networks of the same size as the champion network. Also, as evident from the experiments in Subsections 5.3.1 and 5.3.2, WFNAS is capable of performing well with lower values of p and q. However, this study will give us an approximation of the upper bound of communication overhead for a 2-client setting.

Attributes of the champion network	**Client A**	**Client B**
# connections	621	735
# neurons	419	449
# hidden layers	8	10
Size (compressed)	18 KB	21 KB

Table 5.8: Empirical Values of Attributes of the Champion Networks

The networks are represented using an adjacency matrix, where the rows and the columns indicate the nodes having *connections from* and the nodes having *connections to* respectively. In Table 5.8, we present the experimentally obtained values of several attributes of the champion networks with both clients: *number of connections, number of neurons, number of hidden layers*, and *compressed size.* We use the compressed size of the networks because the number of connections in the champion networks are low and hence the adjacency matrices are very sparse.

During one exchange for the experiment configuration considered above, each client shares **768** networks ($\frac{(p+q)*N}{100}$). The communication cost per exchange is **768 x (18 + 21) = 29.95 MB**. Therefore, the overall communication overhead for $G = 2000$ generations is **59.9 GB**.

Chapter 6

CONCLUSION AND FUTURE WORK

In this work, we proposed a novel weight-agnostic federated NAS approach and performed extensive experiments to prove the efficacy of WFNAS. The networks produced by our proposed approach have several hundred connections all sharing one randomly chosen weight value as opposed to the state-of-the-art CNN networks (both handcrafted and automatically searched) having millions of parameters (each connection having a trainable weight-value) which may be infeasible for deployment on edge devices. Both horizontal and vertical FL settings were considered for experimentation with the MNIST dataset, and we demonstrated that while the approach compares favorably to standalone training with both FL settings, higher gains are observed for the vertical FL setting. To the best of our knowledge, this is also the first NAS contribution to the literature of vertical FL. While the reported testing accuracy for MNIST is not on par with the state-of-the-art accuracy values of bulky CNN architectures, we aim that this work serves as a motivation for the community to explore this direction of research further.

Some directions for future work for this research are as follows:

- **WFNAS for a multi-client FL setting**: In this work, we restricted our experiments to a 2-client FL setting due to the limitation of resources at our disposal. However, we provide a framework and ideas for using the proposed WFNAS for a multi-client setting.
  - Instead of a client sharing $p\%$ and $q\%$ networks with the other client for validation and federated combination, for a multi-client setting, these local

  networks can be shared with all the participating clients in a one-to-many manner for validation on their data. Consequently, the estimator used to predict performance on the other client's data will be unique to each client and the final reward for ranking the local networks will be a weighted average of the reward obtained from all the clients based on the number of samples per class with each client.
  - To reduce the communication overhead that increases by increasing the number of clients, ideas such as sampling a certain number of clients for each round instead of validation from all the clients for every round of exchange can be implemented.

- **WFNAS on more sophisticated datasets**: In this work, we evaluated our approach on the MNIST dataset which despite being a fairly simple dataset, is a motivation to experiment with more sophisticated datasets like CIFAR10 [24] and ImageNet [10].
  - During this study, we experimented with the CIFAR10 for a while, but the memory requirements and time consumption was unreasonably higher even for the standard non-federated WANNs. In [42], the authors corroborate our observation and indicate that the implementation inefficiencies are the reason for the requirement of higher computational resources. More specifically, each network is implemented as a NumPy array, which grows as the evolution progresses and it is noteworthy that these arrays are highly sparse (more than 99% sparsity). A potential direction of future work is rebuilding the implementation code with an optimized representation of networks using frameworks like TensorFlow [1] to reduce the memory consumed by networks and enable support for GPUs.

REFERENCES

[1] Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: A system for large-scale machine learning”, in “12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)”, pp. 265–283 (2016).

[2] Bello, I., B. Zoph, V. Vasudevan and Q. V. Le, “Neural optimizer search with reinforcement learning”, in “International Conference on Machine Learning”, pp. 459–468 (PMLR, 2017).

[3] Bergstra, J. and Y. Bengio, “Random search for hyper-parameter optimization.”, Journal of machine learning research **13**, 2 (2012).

[4] Bhowmick, A., J. Duchi, J. Freudiger, G. Kapoor and R. Rogers, “Protection against reconstruction and its applications in private federated learning”, arXiv preprint arXiv:1812.00984 (2018).

[5] Bradski, A., *Learning OpenCV, [Computer Vision with OpenCV Library ; software that sees]* (O‘Reilly Media, 2008), 1. ed. edn., gary Bradski and Adrian Kaehler.

[6] Chen, M., R. Mathews, T. Ouyang and F. Beaufays, “Federated learning of out-of-vocabulary words”, arXiv preprint arXiv:1903.10635 (2019).

[7] Dalcín, L., R. Paz and M. Storti, “Mpi for python”, Journal of Parallel and Distributed Computing **65**, 9, 1108–1115 (2005).

[8] Darlow, L. N., E. J. Crowley, A. Antoniou and A. J. Storkey, “Cinic-10 is not imagenet or cifar-10”, arXiv preprint arXiv:1810.03505 (2018).

[9] Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database”, in “2009 IEEE conference on computer vision and pattern recognition”, pp. 248–255 (Ieee, 2009).

[10] Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database”, in “2009 IEEE conference on computer vision and pattern recognition”, pp. 248–255 (Ieee, 2009).

[11] Elsken, T., J. H. Metzen and F. Hutter, “Efficient multi-objective neural architecture search via lamarckian evolution”, arXiv preprint arXiv:1804.09081 (2018).

[12] Elsken, T., J. H. Metzen and F. Hutter, “Neural architecture search: A survey”, The Journal of Machine Learning Research **20**, 1, 1997–2017 (2019).

[13] Erlingsson, Ú., V. Pihur and A. Korolova, “Rappor: Randomized aggregatable privacy-preserving ordinal response”, in “Proceedings of the 2014 ACM SIGSAC conference on computer and communications security”, pp. 1054–1067 (2014).

[14] Gaier, A. and D. Ha, "Weight agnostic neural networks", arXiv preprint arXiv:1906.04358 (2019).

[15] Garg, A., A. K. Saha and D. Dutta, "Direct federated neural architecture search", arXiv preprint arXiv:2010.06223 (2020).

[16] Hard, A., K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon and D. Ramage, "Federated learning for mobile keyboard prediction", arXiv preprint arXiv:1811.03604 (2018).

[17] He, C., M. Annavaram and S. Avestimehr, "Fednas: Federated deep learning via neural architecture search", arXiv e-prints pp. arXiv–2004 (2020).

[18] He, C., H. Ye, L. Shen and T. Zhang, "Milenas: Efficient neural architecture search via mixed-level reformulation", in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition", pp. 11993–12002 (2020).

[19] He, C., H. Ye, L. Shen and T. Zhang, "Milenas: Efficient neural architecture search via mixed-level reformulation", in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition", pp. 11993–12002 (2020).

[20] Hu, S., S. Xie, H. Zheng, C. Liu, J. Shi, X. Liu and D. Lin, "Dsnas: Direct neural architecture search without parameter retraining", in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition", pp. 12084–12092 (2020).

[21] Huang, G., Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely connected convolutional networks", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 4700–4708 (2017).

[22] Jiang, F., Y. Jiang, H. Zhi, Y. Dong, H. Li, S. Ma, Y. Wang, Q. Dong, H. Shen and Y. Wang, "Artificial intelligence in healthcare: past, present and future", Stroke and vascular neurology **2**, 4 (2017).

[23] Kairouz, P., H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning", arXiv preprint arXiv:1912.04977 (2019).

[24] Krizhevsky, A., G. Hinton *et al.*, "Learning multiple layers of features from tiny images", (2009).

[25] Krizhevsky, A., G. Hinton *et al.*, "Learning multiple layers of features from tiny images", (2009).

[26] LaValle, S. M., M. S. Branicky and S. R. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps", The International Journal of Robotics Research **23**, 7-8, 673–692 (2004).

[27] LeCun, Y. and C. Cortes, "MNIST handwritten digit database", URL `http://yann.lecun.com/exdb/mnist/` (2010).

[28] Lecuyer, M., V. Atlidakis, R. Geambasu, D. Hsu and S. Jana, "Certified robustness to adversarial examples with differential privacy", in "2019 IEEE Symposium on Security and Privacy (SP)", pp. 656–672 (IEEE, 2019).

[29] Liu, H., K. Simonyan and Y. Yang, "Darts: Differentiable architecture search", arXiv preprint arXiv:1806.09055 (2018).

[30] Luo, R., F. Tian, T. Qin, E. Chen and T.-Y. Liu, "Neural architecture optimization", arXiv preprint arXiv:1808.07233 (2018).

[31] McMahan, B., E. Moore, D. Ramage, S. Hampson and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data", in "Artificial intelligence and statistics", pp. 1273–1282 (PMLR, 2017).

[32] Miller, B. L., D. E. Goldberg *et al.*, "Genetic algorithms, tournament selection, and the effects of noise", Complex systems **9**, 3, 193–212 (1995).

[33] Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning", (2011).

[34] Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine learning in Python", Journal of Machine Learning Research **12**, 2825–2830 (2011).

[35] Pham, H., M. Guan, B. Zoph, Q. Le and J. Dean, "Efficient neural architecture search via parameters sharing", in "International Conference on Machine Learning", pp. 4095–4104 (PMLR, 2018).

[36] Pichai, S., "Privacy should not be a luxury good", The New York Times (2019).

[37] Ramaswamy, S., R. Mathews, K. Rao and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard", arXiv preprint arXiv:1906.04329 (2019).

[38] Real, E., A. Aggarwal, Y. Huang and Q. V. Le, "Regularized evolution for image classifier architecture search", in "Proceedings of the aaai conference on artificial intelligence", vol. 33, pp. 4780–4789 (2019).

[39] Real, E., S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le and A. Kurakin, "Large-scale evolution of image classifiers", in "International Conference on Machine Learning", pp. 2902–2911 (PMLR, 2017).

[40] Reina, G. A., A. Gruzdev, P. Foley, O. Perepelkina, M. Sharma, I. Davidyuk, I. Trushkin, M. Radionov, A. Mokrov, D. Agapov *et al.*, "Openfl: An open-source framework for federated learning", arXiv preprint arXiv:2105.06413 (2021).

[41] Rivest, R. L., L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms", Foundations of secure computation **4**, 11, 169–180 (1978).

[42] Sääw, D. and F. Nijweide, "Exploring the feasibility of weight agnostic neural networks for low-end hardware", (2021).

[43] Saxena, S. and J. Verbeek, "Convolutional neural fabrics", Advances in neural information processing systems **29**, 4053–4061 (2016).

[44] Singh, I., H. Zhou, K. Yang, M. Ding, B. Lin and P. Xie, "Differentially-private federated neural architecture search", arXiv preprint arXiv:2006.10559 (2020).

[45] Stanley, K. O. and R. Miikkulainen, "Evolving neural networks through augmenting topologies", Evolutionary computation **10**, 2, 99–127 (2002).

[46] Swersky, K., J. Snoek and R. P. Adams, "Freeze-thaw bayesian optimization", arXiv preprint arXiv:1406.3896 (2014).

[47] Tong, W., A. Hussain, W. X. Bo and S. Maharjan, "Artificial intelligence for vehicle-to-everything: A survey", IEEE Access **7**, 10823–10843 (2019).

[48] Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", Nature Methods **17**, 261–272 (2020).

[49] Wikipedia contributors, "Dirichlet distribution — Wikipedia, the free encyclopedia", `https://en.wikipedia.org/w/index.php?title=Dirichlet_distribution`, [Online; accessed 12-October-2021] (2021).

[50] Wikipedia contributors, "Pearson correlation coefficient Wikipedia, the free encyclopedia", `https://en.wikipedia.org/w/index.php?title=Pearson_correlation_coefficient&oldid=1049506440` (2021).

[51] Wikipedia contributors, "Secure copy protocol — Wikipedia, the free encyclopedia", `https://en.wikipedia.org/w/index.php?title=Secure_copy_protocol&oldid=1049551380` (2021).

[52] Williams, R. J., "Simple statistical gradient-following algorithms for connectionist reinforcement learning", Machine learning **8**, 3, 229–256 (1992).

[53] Xie, S., H. Zheng, C. Liu and L. Lin, "Snas: stochastic neural architecture search", arXiv preprint arXiv:1812.09926 (2018).

[54] Yang, J., Y. Chen, W. Huang and Y. Li, "Survey on artificial intelligence for additive manufacturing", in "2017 23rd International Conference on Automation and Computing (ICAC)", pp. 1–6 (IEEE, 2017).

[55] Yang, J., R. Shi and B. Ni, “Medmnist classification decathlon: A lightweight automl benchmark for medical image analysis”, in “2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)”, pp. 191–195 (IEEE, 2021).

[56] Yang, T., G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage and F. Beaufays, “Applied federated learning: Improving google keyboard query suggestions”, arXiv preprint arXiv:1812.02903 (2018).

[57] Yao, A. C., “Protocols for secure computations”, in “23rd annual symposium on foundations of computer science (sfcs 1982)”, pp. 160–164 (IEEE, 1982).

[58] Zhu, H. and Y. Jin, “Multi-objective evolutionary federated learning”, IEEE transactions on neural networks and learning systems **31**, 4, 1310–1322 (2019).

[59] Zhu, H. and Y. Jin, “Real-time federated evolutionary neural architecture search”, IEEE Transactions on Evolutionary Computation (2021).

[60] Zhu, H., H. Zhang and Y. Jin, “From federated learning to federated neural architecture search: a survey”, Complex & Intelligent Systems **7**, 2, 639–657 (2021).

[61] Zoph, B. and Q. V. Le, “Neural architecture search with reinforcement learning”, arXiv preprint arXiv:1611.01578 (2016).