Informatics

# Self-Elective Model Branching in Online Machine Learning on the Edge

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Software Engineering & Internet Computing**

eingereicht von

**Bernhard Schwarz, BSc**
Matrikelnummer 01325096

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Ivona Brandić

Wien, 14. Oktober 2020

| Bernhard Schwarz | Ivona Brandić |
|---|---|

TU WIEN Informatics

# Self-Elective Model Branching in Online Machine Learning on the Edge

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Software Engineering & Internet Computing**

by

**Bernhard Schwarz, BSc**
Registration Number 01325096

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Ivona Brandić

Vienna, 14th October, 2020

Bernhard Schwarz Ivona Brandić

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

# Erklärung zur Verfassung der Arbeit

Bernhard Schwarz, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 14. Oktober 2020

Bernhard Schwarz

# Danksagung

An erster Stelle möchte ich Ivona Brandić für die Möglichkeit, diese Idee im Rahmen dieser Diplomarbeit verfolgen zu können, danken.

Mein Dank gilt auch Atakan Aral, durch dessen Forschung diese Arbeit inspiriert wurde und der mir hilfsbereit bei Fragen beiseitestand.

Weiters danke ich Adriatik Nikaj und Thomas Rausch, deren Rat und Unterstützung mir im rechten Moment Halt gab.

Vielen Dank an Jianmo Ni für das rasche Bereitstellen des FitRec Datasets in einem anderen Dateiformat.

# Acknowledgements

First and foremost, I want to thank Ivona Brandić for giving me the opportunity to pursue this idea.

Atakan Aral's work inspired this thesis and I'm thankful for his help with any questions I had.

I'm grateful to Adriatik Nikaj and Thomas Rausch for their insight and support that helped me along the way.

Thanks also to Jianmo Ni for graciously providing the FitRec dataset in the file format I needed upon request.

# Kurzfassung

In der heutigen technologischen Landschaft werden immer mehr Geräte und Sensoren mit dem Internet verbunden. Um die neuen Anforderungen, die dadurch entstehen, zu decken, werden Rechenressourcen an den Rand des Netzwerks – auch *Edge* genannt – verlagert, was eine Dezentralisierung zur Folge hat. *Machine Learning*, eine Technologie, die für Anwendungsfälle mit enormen Datenaufkommen besonders geeignet ist, stellt hier keine Ausnahme dar. In der Praxis müssen die eingesetzten Modelle allerdings in vielen Fällen kontinuierlich angepasst werden, da sich auch die zu verarbeitenden Daten ändern können. Dadurch wird eine Synchronisierung der Modelle zwischen den Edge-Knoten erforderlich, um Konsistenz zu gewähren. Durch die statistische Heterogenität, die in solchen Einsatzszenarios oft vorliegt, können aber deutliche Diskrepanzen zwischen globaler und lokaler Modellqualität entstehen, die verheerende Folgen in kritischen Anwendungsfällen wie der medizinischen Versorgung verursachen können. Ansätze zur sogenannten *Model Personalization*, die das globale Modell an die Daten, die bei den einzelnen Knoten aufliegen, anpassen, werden als Lösung für dieses Problem erforscht.

In dieser Diplomarbeit wird eine neue Methode zur Model Personalization vorgestellt, die einen beliebten Federated-Learning-Algorithmus dahingehend modifiziert, bisher verworfene Nebenprodukte des Synchronisierungsprozesses zu verwenden, um die lokale Inferenzqualität zu verbessern, sowie schneller auf Ereignisse zu reagieren. Diese Modifikationen erlauben es den Edge-Knoten, lokal angepasste Modelle für Inferenzzwecke temporär weiterzuführen. Zur Validierung des vorgeschlagenen Ansatzes wird ein Algorithmus entworfen, implementiert und in einer simulierten Testumgebung zur Lösung einer Forecast-Aufgabe mittels mehrerer synthetischer und echter Datensätzen getestet. Dabei wird die Genauigkeit der Vorhersagen gemessen und mit den Vorhersagen des ursprünglichen Algorithmus verglichen. Diese numerischen Experimente liefern gemischte, aber Erfolg versprechende Ergebnisse.

# Abstract

Computational resources are moving closer to the edge of the network in response to the ever increasing number of connected devices and sensors in today's technological landscape. Machine learning applications that rely on large amounts of data are no exception. In many practical applications, the data changes over time, which necessitates continuous updates to the used machine learning models in the form of online learning, requiring that edge nodes synchronize their models periodically for consistency. Additionally, the statistical heterogeneity that is common in the geographically distributed edge setting can cause large discrepancies between the accuracies of such globally optimized models at the global and local level, which can have devastating consequences in critical applications such as medical care. Model personalization approaches that adjust the model to local data are actively being researched to remedy this.

This thesis introduces a novel model personalization scheme based on a popular federated learning algorithm that seeks to utilize the currently discarded byproducts of the synchronization process to improve local inference accuracy and allow for more timely adaptation to localized phenomenons. In the process, edge nodes can decide to temporarily decouple from the synchronized model for their inference tasks and instead use locally continued branches. To validate the proposed approach, an algorithm is designed, implemented and tested in a simulated environment on univariate forecasting tasks using several synthetic and real-world datasets and the accuracy measured against the baseline algorithm. The numerical experiments yield mixed, but promising results.

# Contents

CHAPTER 1

# Introduction

*Cloud computing* is a popular method of flexibly provisioning computing resources as needed over the Internet. Instead of buying and operating hardware, virtualized resources are rented in the datacenters of a cloud provider. In recent years, the ever increasing number of connected devices and sensors has created a demand for computing resources closer to the edge of the network, away from the centralized datacenters used in cloud computing. This paradigm known as *edge computing* enables applications that rely on the reduced latency allowed by closer physical proximity or the ability to transmit large amounts of data that would otherwise prove impossible or cost prohibitive in traditional cloud computing. Edge infrastructure deployments can range from simple devices to micro datacenters, using similar enabling technologies as the cloud. Machine learning techniques are defined by their ability to approximate arbitrary functions when given enough data. This makes them a natural fit for this new landscape, often outperforming traditional data processing methods. Machine learning uses models to encode the knowledge required to solve tasks. These models are trained by gradually adjusting the parameters they consist of until satisfactory performance is achieved. Model performance is measured in different metrics depending on the type of task, e.g. models used to forecast time series are deemed more accurate the smaller the errors between observed and predicted values are. Once a model has received enough training to be considered accurate enough, it can be deployed into production. However, in many practical applications, the live data will at some point start to diverge from the training data, causing the accuracy of the model to degrade. An email spam filter that is unable to deal with newly emerging methods designed to circumvent it would be an example of this phenomenon commonly known as *dataset shift*[1]. By incrementally updating the model as it is exposed to new data, *online machine learning* can mitigate the negative impact of dataset shift. Models deployed at different edge nodes can be trained independently in this fashion, but should periodically be synchronized with others to produce a more consistent and robust global model based on the data encountered across *all* nodes. Optimizing model performance throughout the synchronization process in this distributed setting is an open research question which this thesis seeks to address.

Machine learning techniques today play a crucial role in many applications that were not possible before and it is likely that new possibilities will crop up as the technology matures even further. Therefore, improving the accuracy of machine learning models has clear benefits. Self-driving cars and smart electric grid are examples of applications where model accuracy is critical. Smart grids possess new capabilities for energy control that are estimated to allow for significant cost reductions by means which include supply and demand forecasting or theft detection. Other examples include the integration of electric vehicles into the smart grid, augmented and virtual reality[2] and traffic management[3].

Many current synchronization schemes utilize a single centralized server, often called *parameter server*, to aggregate model updates from the nodes. By only sending updates and not the full data to the parameter server, transmission costs are reduced and the data stays on the nodes, which has positive effects on privacy. Once the parameter server has received enough updates, it assembles the next global update and broadcasts it back to the nodes. Since this communication round trip is time consuming, it is infeasible to synchronize the models in real-time. Instead, nodes will typically only periodically synchronize with the parameter server depending on the scheme. However, this delay has consequences in the online edge setting, where the data encountered can vary significantly between nodes, while the underlying learning algorithms expect the source distributions of the data at each node to be independent and identically distributed (IID). When data samples violate these assumptions, they are called non-IID. Research in the relatively young field of Federated Learning (FL) has yielded methods to deal with such non-IID data to a certain degree, although this issue has only recently been given more attention as the prime objective of federated learning was to enable machine learning on mobile devices such as smartphones. The statistically heterogeneous data distributions that are prevalent in this setting can also lead to situations where the globally optimized model performs poorly on the data present at individual nodes.

Many synchronization schemes in this setting are based on the *FedAvg* (Federated Averaging) algorithm introduced by McMahan et. al[4], which remains the benchmark to this day. *FedAvg* has been empirically shown to be able to deal with non-IID data, although the model it produces can diverge under certain conditions of high statistical heterogeneity[5]. In addition, high accuracy of the model on the global overall distribution does not necessarily reflect on the performance at the node level[6], which could lead to dramatic consequences in certain applications such as medical diagnosis[7]. To improve the inference accuracy at the node level, *model personalization* through local model fine tuning is a popular method. In this approach, the global model is first lightly trained on local data by the node before it is used in inference to achieve a better fit[8]. In the online learning setting, such personalization would have to be carried out at the start of each iteration. This is not only costly, but also neglects the possibility of finding a high performing model as part of regular participation in the global model consensus. Current approaches replace the nodes' locally tuned models with the latest global model without questioning whether this will actually have a positive impact on inference accuracy.

Reacting to dataset shift is a core goal of online learning. This thesis thus seeks to answer the following question: *Can inference results be improved if nodes are allowed to keep training and using older global models?* Intuitively, temporarily decoupling from the synchronization process should allow nodes to better respond to localized phenomenons, as new insights they gain from local data are not diluted by global knowledge. When a node decides to reject the current global model, it effectively creates a model branch dating back to the previous iteration. The nodes themselves are in the best position to make this decision, as only they have knowledge of their local data partitions.

The model branching approach is empirically evaluated on univariate time series forecasting tasks over three real-world datasets and one generated synthetic dataset for better control over dataset shift and data distributions. In these experiments, notable forecast accuracy improvements are achieved on the synthetic dataset as well as one of the real-world datasets.

This thesis makes the following contributions:

- It introduces an online learning algorithm that utilizes a novel model personalization approach to improve local inference quality.
- The proposed model branching method provides incentive to join the federation to nodes with well-performing local models.
- The created simulation testbed makes it possible to empirically test branching strategies in the edge setting.

The thesis is organized as follows. In Chapter 2 related work is discussed to give an overview of existing approaches. Chapter 3 introduces fundamental concepts and terms that are necessary for understanding. In Chapter 4, the model branching method is described and the rational behind it presented. Chapter 5 outlines the experiment setup for the empirical verification of the method and Chapter 6 shows and discusses the results of analysis of outcomes. Finally, conclusions and future research opportunities are given in Chapter 7.

CHAPTER 2

# Related Work

The setting this thesis addresses matches cross-silo Federated Learning as described by Kairouz et al.[8], wherein the participating nodes are either organizations such as hospitals or geo-distributed data centers. The network link from the node to the central orchestration server is usually fast and stable, but the data generated by end users or devices communicating with the nodes might be large, limiting the possibility of moving it around.

In this setting, data cannot be assumed to be IID, as explored in the previous section. The impact non-IID training data can have on the accuracy of models is significant, Zhao et al.[9] demonstrate a decrease in model accuracy of up to 55 % in neural networks trained using highly skewed non-IID training data, while Hsieh et al.[10] report a 39 % decrease in their experiments. This data heterogeneity also raises the question whether a single global model is the answer to the individual optimization tasks at the node level. Changing behavior of the data producers and environments further complicate ML deployments on the edge, necessitating continued model refinements, e.g. through online learning, to prevent service quality degradation[2]. The question thus becomes how to provide nodes with the most optimal model for the data they will see, while also building consensus on the globally best model that can then serve as a checkpoint and be used as the starting point for newly joining nodes.

It is important to note that the focus of this thesis lies specifically on a setting in which the models are already used for live inference while they are being trained. Although the resulting global is still a priority, the quality of local inference results is crucial. Most research on dealing with non-IID data however is primarily focused on reaching well performing models only as the end result. In those FL scenarios where the number of participating nodes is very large, selection algorithms will pick a subset of nodes to contribute to the global model during each iteration. Furthermore, a node will typically only be chosen once in the training of a model, necessitating stateless local algorithms[8].

This is in contrast to the setting at hand, where nodes are expected to participate in all iterations and are able to store state.

Jiang et al.[6] argue for the importance of addressing all three of the following goals: (1) Improved Personalized Model (2) Solid Initial Model and (3) Fast Convergence. Improving personalized models will benefit most participating nodes by improving inference accuracies, while a solid initial model ensures nodes with limited or no data to personalize with still experience good performance. Fast convergence on a high quality model remains as crucial as ever.

## 2.1 Model Convergence

Given high statistical heterogeneity, the ability to converge on a well performing model is critical and must be solved first. The *FedAvg*[4] algorithm enjoys continued success as the default federated optimization algorithm of choice. It has been empirically shown to be able to deal with some degree of non-IID data[4], but the global model can diverge under high statistical heterogeneity[5]. Various efforts have been made to develop theoretical convergence guarantees for *FedAvg* or a relaxed variant *LocalSGD*[1][12][13], although they rely on assumptions that do not hold in FL[11]. Li et al.[11] establish convergence bounds for *FedAvg* when used on strongly convex problems without assuming IID data and full device participation under the condition that the learning rate decays over time. Convergence guarantees on non-convex problems in the federated setting are however still outstanding. This continued failure to establish theoretical guarantees is critiqued in the literature as a weakness of the algorithm.

The *FedProx* framework introduced by Li et al.[14] generalizes *FedAvg* and makes small adjustments that allows them to provide theoretical convergence guarantees in the presence of non-IID data not only in convex but also non-convex problems. The eponymous proximal term is added to the local optimization objective of the nodes, restricting the impact of local updates to the model. Although *FedAvg* can be emulated in *FedProx* by using certain parameters, these convergence guarantees do not extend to *FedAvg* as the proximal term required in their convergence theory is then absent[11].

Xie et al.[15] propose *SLSGD* (Secure Local SGD) to deal with non-IID data under the added threat of data poisoning carried out by a small number of nodes. Instead of averaging parameters to update the global model, it is updated using a moving average to secure it against data poisoning. In *SLSGD*, nodes do not send gradients but parameters to the central server, similar to earlier *FedAvg* implementations. The authors prove convergence empirically and provide theoretical guarantees for convex and a subset of non-convex problems.

Chen et al.[16] propose an asynchronous FL framework for the online learning use case in which edge devices operate on continuously streaming local data. In their experiments, the asynchronous approach shows promise when dealing with the heterogeneous loads at

[1]Equivalent to *FedAvg* under the assumption that data is IID and all participants are active[11]

different participating devices as well as stragglers and dropouts. *ASO-Fed* converges for both convex and non-convex problems on non-IID data under a bounded gradient dissimilarity assumption.

Another way to address the problem of non-IID data distributions is to enrich local datasets with extra data to make them more IID. Jeong et al.[17] follow through on this idea using a generative model, more specifically a Generative Adversarial Network (GAN), that is trained centrally on data samples uploaded by the nodes. The nodes then use the resulting GAN to augment their real training dataset with synthetic data. However, sharing local data this way not only carries increased network costs, it also violates the privacy assumption of FL[14]. Zhao et al.[9] use a small, globally shared dataset that is then mixed with the local data during training at the nodes. Assembling this dataset without drawing from private data however requires effort and might not always be possible[14]. In their experiments, the authors create this dataset by making sure all classes in their image classification task are fairly represented, but for other tasks such as detection and prediction ensuring that the shared dataset is IID might prove more challenging.

## 2.2 Model Optimization

Approaches using multiple models have been shown to be able to outperform even the best possible global model in terms of inference accuracy when the dataset is non-IID[8]. Yu et al.[18] show that even in common tasks like next-word prediction the global models pursued in FL often fail to outperform models that participants trained independently on only their local data. They attribute this partially to the robust aggregation methods used to limit the influence of outliers and malicious participants on the global model. The authors further demonstrate in their experiments that all participating nodes improve inference accuracy on their data by locally adapting the global model.

### 2.2.1 Fine-Tuning

Local fine-tuning is the current predominant personalization approach in FL[5]. In fine-tuning methods, nodes commonly first train the global model on local data for one or several steps of SGD before using it for inference.

*Transfer Learning (TL)* is a similar method of adapting a model with good performance to local tasks more commonly used outside of FL[8]. In TL, the existing layers of such a model are frozen, i.e. they are prevented from changing, and new trainable layers added to predict the new task. Li and Wang[7] use a combination of TL and knowledge distillation to perform federated learning in a setting where all participants have unique model designs to solve a classification problem.

*Meta-learning* approaches have found use in the field of FL primarily in the form of *Model-Agnostic Meta-Learning (MAML)*[19]. Finn et al.[19] phrase the goal as being able to learn the parameters of a model in a way that allows fast adaptation to a new task using only few data points and gradient steps. They propose that the internal representation of some models makes them more suitable for such adaptation than others, which has been confirmed by others[6].

Non-IID data distributions across nodes as well as sudden dataset shift could well be understood as such tasks that necessitate model adaptation. Jiang et al.[6] point out the connection between *FedAvg*[4] and *MAML* (as do Khodak et al.[20] and Fallah et al.[21]), showing that models trained using *FedAvg* do indeed have much higher propensities for personalization than centrally trained models.

Liang et al.[22] introduce *LG-FedAvg*, which jointly learns purely local representations of the data on each node and a global model across all nodes. When the local representation is of a lower dimension than the source data, the global model can potentially be smaller, i.e. consist of fewer parameters, thereby reducing the number of parameters that need to be updated and communicated during synchronization. This ensemble approach of using local and global models effectively achieves fine-tuning by the local updates to the model used to transform data into local representation, which the authors show enable it to react better to statistical heterogeneity as well as dataset shift. The loss function used incorporates both the local and global model, which Liang et al. argue synchronizes the local representation learned at the nodes; the local models cannot overfit to local data since the accuracy of the joint global model would then suffer. Their experiments show *LG-FedAvg* outperforming *FedAvg*[4] and *FedProx*[14] in an online setting where a node with drastically different data is introduced which is achieved by rotating the MNIST[2] dataset by 90 degrees in a digit classification task. *LG-FedAvg* also relieved catastrophic forgetting in this scenario compared to the baseline algorithms.

### 2.2.2 Multi-Task Learning

When viewing nodes and their local problems as separate tasks, research and techniques of the field of *Multi-Task Learning (MTL)*, which deals with learning models for multiple related task simultaneously, should also be considered[8].

Wang et al.[23] recognize the issue of massive datasets and expensive communication, especially when data distributions on different machines do not belong to the same source distribution. They achieve results comparable to centralized MTL approaches in a distributed setup while improving communication efficiency. Baytas et al.[24] describe the importance of solving the problem of non-IID data in distributed ML in edge settings and tackle it using MTL techniques. Their framework provides a more principled way compared to other heuristic or approximation based solutions for distributed MTL and present a linear convergence rate for convex problems.

Smith et al.[25] make the connection between MTL and FL, claiming that MTL is a natural fit for the challenge of statistical heterogeneity by learning separate models at each node. Their *MOCHA* algorithm for distributed MTL does not rely on prior knowledge of

[2]`http://yann.lecun.com/exdb/mnist/`

the relationships between tasks and can handle dropouts as well as straggling participants. They further show that *MOCHA* converges both empirically and theoretically on convex problems.

Current MTL approaches do not seem to be ready to handle non-convex problems, which are typical problems solved by deep learning.

### 2.2.3 Model Mixing

Hanzely and Richtarik[26] propose to change the optimization formulation of FL to seek a trade-off between a traditional global model and purely local models trained without outside influence to improve performance and convergence in a heterogeneous data setting. They use a mixing parameter to prioritize either local models for small values of the parameter or convergence to the optimal global model for high values. The authors also give theoretical convergence guarantees they state could be extended to the non-convex case.

Mansour et al.[27] present three personalization algorithms that can be used either standalone as well as together, including an approach they call model interpolation which resembles model mixing. In their approach, they train a local and a global model and then use an interpolation of the two. Deng et al.[5] follow a very similar approach where each node trains a local model, incorporating the global model using a multiplier to weight the mixing process towards either side. Finding the right mixing parameter is in itself an optimization problem. As Deng et al. note, when the data is distributed uniformly across nodes it should be trending towards zero, as the global model is preferable in such cases. However, under high statistical heterogeneity it is more beneficial to the nodes to primarily rely on their own data, making use of some limited amount of knowledge from the other nodes to improve generalization. Both papers show the value of their proposed implementations empirically, however Deng et al. also show the impact of model mixing on the theoretical generalization bound.

### 2.2.4 Fully Decentralized Learning

In fully decentralized learning, there are no coordinating central servers overseeing the learning process and nodes are usually communicating in a peer-to-peer fashion. Although much of the older work focuses on consensus on a single model, advances have been made to collaboratively learn personalized models.

Rather than having nodes try to find consensus on a global model, Vanhaesebrouck et al.[28] suggest an approach where each node learns a personalized model according to local objectives. They assume that a network graph is given wherein neighboring nodes have similar objectives, i.e. similar data, but only the direct neighbors are known to the nodes. The algorithms introduced by Vanhaesebrouck et al. regularizes model parameters by smoothing them out across the network graph, which they prove for convex objectives leads to a state in which all nodes have an optimal model. Bellet et al.[29] improve on the work of Vanhaesebrouck et al. by analyzing the trade-off between utility and privacy when allowing local datasets to remain private to the nodes. Zantedeschi et al.[30] further propose an optimization procedure that doesn't require the network graph to be known ahead of time, but instead learns it in an alternating manner together with the models.

## 2.3 Other Work

Geographically distributed deployments of ML techniques on the edge present further challenges beyond non-IID data distributions and dataset shift.

Many edge nodes are naturally constrained in their usage of local computation power or network functions, i.e. smartphones or other battery-powered devices. Wang et al.[31] explore a central control algorithm to reach the most efficient trade-off between local updates to the models, which consume computation resources, and global model aggregation steps, which requires use of the communication resource of the network. Their proposed algorithm seeks to set the global aggregation frequency in a manner such that the less constrained resource is utilized preferentially. The authors provide formal convergence bounds for gradient-descent based FL of convex problems in the presence of non-IID data distributions and arbitrary numbers of local updates between global aggregation steps, which serves as the basis for their control algorithm. Further, they empirically confirm its effectiveness in a series of experiments where it is compared to existing algorithms.

Aral et al.[2] also propose an adaptive synchronization rate, but utilize it to address generality and timeliness of models in an online learning edge setting. The authors identify three open challenges of this setting which they seek to address: (1) Intermittent connectivity or package loss experienced in mobile edge applications impedes normal model synchronization (2) Edge nodes are only privy to data from sources from within their catchment (3) The data is often non-stationary, leading to dataset shift that must be mitigated through frequent model refinements. They posit that during the model update aggregation phase at the parameter server, there is a point in time regarding generality and staleness where it is optimal to disregard potential updates from straggling participating nodes and instead assemble and broadcast a new global model. In their experiments, Aral et al. show that their proposed *SCEDA* algorithm effectively addresses the insularity challenge, reaching comparable results as centralized algorithms but with near-real time timeliness.

CHAPTER 3

# Preliminaries

In this section, fundamental concepts and terms that are used throughout the thesis are introduced. This high level overview serves to establish or refresh a knowledge base sufficient to understand both problem and proposed solution as described in the remaining chapters.

## 3.1 Deep Learning

Deep Neural Networks (DNNs) have become the representative machine learning method thanks to advances in algorithms and computer hardware. Consequently, DNNs have won many image classification, recognition and object detection contests starting in 2012[32]. While they rely on complex fundamentals, in practice it is relatively easy to understand the high level building blocks and develop models using DNNs that compare favorably to the previous state of the art.

Neural networks consist of layers of neurons, which are real-valued activation functions with multiple inputs. The individual inputs are weighted, i.e. multiplied with parameters, to adjust their influence on the output. Intuitively, the outputs of some neurons in the proceeding layer might be more important in this neuron than other inputs. During training, the parameters are slowly adjusted to reduce a given error or loss function that is applied to see how far off the model's prediction was from the actual truth. Before Stochastic Gradient Descent (SGD), Gradient Descent was used to calculate each step using the entirety of samples in the training dataset. SGD differs in that each step can instead be calculated using only a single data point or a small batch of samples, which might produce a less smooth path towards optimizing the network, but is cheaper to compute. Figure 3.1 shows an example neural network consisting of four layers of neurons in total. The first layer is the input layer accepting the input data and the last is the output layer which returns the model's prediction. The two layers in the middle are called hidden layers, because they are not visible from the outside; the name simply

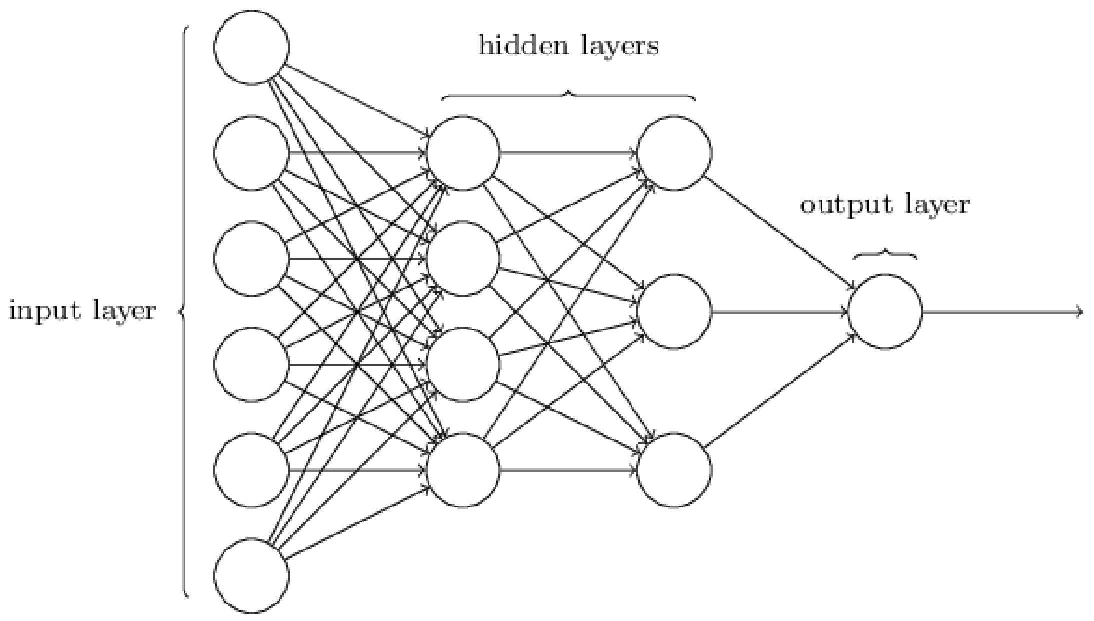


Figure 3.1: Example of a feedforward Deep Neural Network with multiple input neurons, two hidden layers and a single output neuron[33].

represents the fact that they are not input or output layers. Neural networks are called deep when there are at least two of such hidden layers[33].

The training is further modulated by an additional hyperparameter called learning rate, which modifies the adjustments to the model's weights. When the learning rate is too high, the training can continually overshoot the target, never getting closer while too low learning rates can leave the parameters stuck in a local optimum they cannot progress past.

## 3.2 Distributed Machine Learning

Some neural network models have too many parameters or require far too much training data to be reasonably trained on a single machine. These problems were the initial motivators for distributed model training on multiple computers. To aggregate the progress from the worker nodes, centralized parameter servers[34] are commonly used. First implementations saw the nodes sending their models' parameters to the parameter server, which aggregated and stored them, but today nodes commonly only send updates in the form of choice parameter gradients, which makes communication more efficient. Real time synchronization with the parameter server is infeasible due to the overhead it would incur, which spawned research into synchronization schemes to balance trade-offs.

Jiang et al.[35] identify the following three popular synchronization schemes:

**Asynchronous Parallel (ASP)** In asynchronous SGD procedures, workers fetch and push parameters without consideration for other worker nodes. Since fast nodes do not have to wait for slower ones, ASP systems can be faster, but lack convergence guarantees when worker speeds diverge too much.

**Bulk Synchronous Parallel (BSP)** Ensures that workers all operate on a consistent set of parameters by not letting fast workers proceed beyond a certain point until the parameter server receives an update from all workers. This scheme incurs a high synchronization overhead as fast workers must wait for straggling ones to catch up.

**Stale Synchronous Parallel (SSP)** Unlike BSP, fast workers can start the next iteration using stale parameters as long as they are not too far ahead of the straggling nodes under some measure of staleness. However, the updates from workers that are too far out of sync might still end up pushing a global parameter that is close to optimality further away again.

## 3.3 Federated Learning

Distributed machine learning lives in data centers and the cloud, but ML has found its way onto more constrained edge devices such as mobile phones. Federated Learning originated out of the desire to train ML models on smartphones where concerns over privacy and communication costs are limiting factors in sending the data off to a central data center. Horizontal FL, in which the data is partitioned by samples, especially bears many similarities to distributed ML at the surface level[36]. While distributed machine learning and FL are in fact very similar, one key difference is that in FL the data is typically private to the node and distributed in a non-IID (see Section 3.5) fashion[31][22]. An example for a common FL task given by McMahan et al.[4] is training a language model to improve keyboard input on smartphones by correcting words or predicting the next one. In such applications it is clear that the data would preferentially never leave the user's device for privacy, and algorithms such as *FedAvg* proposed by McMahan et al. facilitate this by learning a global model only on locally computed updates that are then aggregated in a central location. Figure 3.2 shows a typical training iteration in FL. The mobile phone setting also means that not all participating devices are available at all times, since battery life should not be negatively impacted and network connectivity is not always given. For this reason, FL algorithms often include a participant selection step at the start of each iteration. Further, current algorithms are limited to only hundreds of participating devices, as more cannot be utilized efficiently[37]. This is an issue in the mobile phone use case, but not as much in the micro data center setting this thesis is targeting.

## 3.4 Dataset Shift

Dataset shift is defined by Quionero-Candela et al. [1] as the discrepancy between the joint distributions of inputs and outputs of training and test data. They argue that

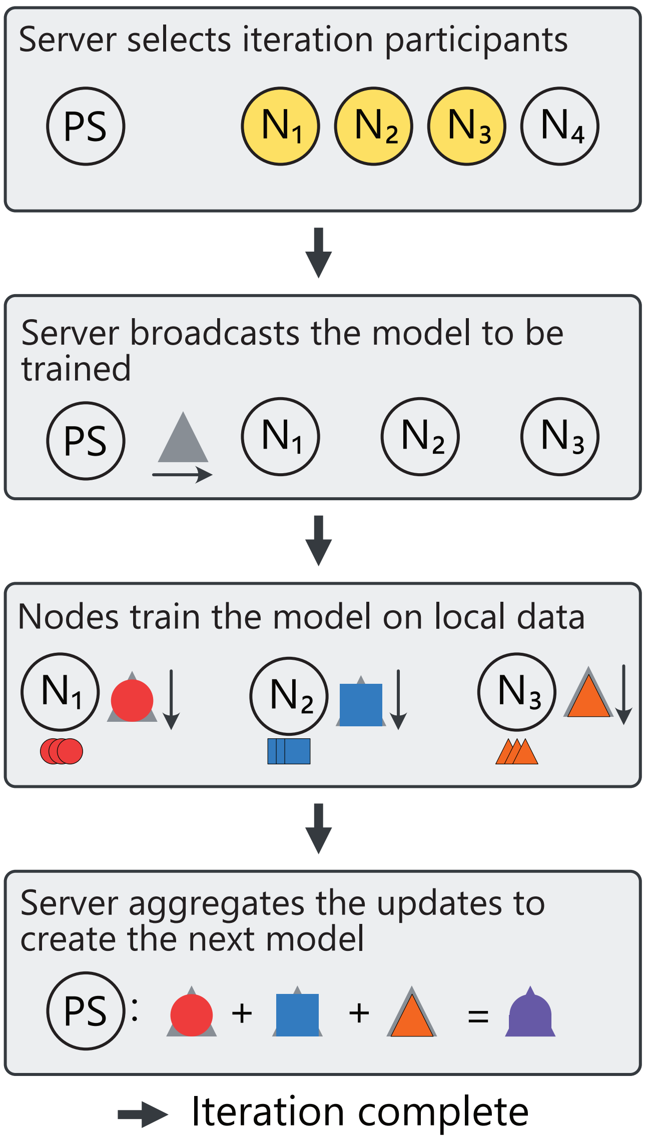

Server selects iteration participants
PS
N1
N2
N3
N4
Server broadcasts the model to be trained
PS
N1
N2
N3
Nodes train the model on local data
N1
N2
N3
Server aggregates the updates to create the next model
PS:
+
+
=
Iteration complete

Figure 3.2: Federated learning iteration. The central server broadcasts the chosen model to be trained after participant selection. The selected nodes train the received model on their local data and respond with their update. The server aggregates the model updates to build the global model for the next iteration.

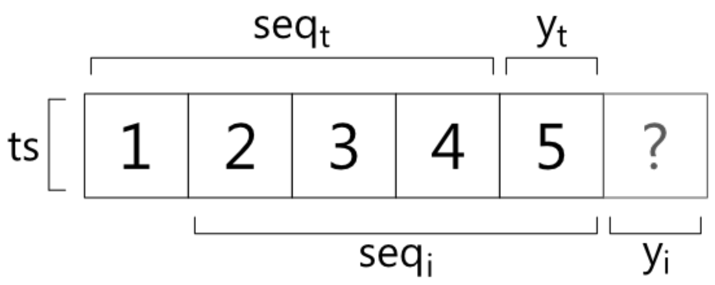


Figure 3.3: Online learning a time series forecasting task. $y_i$ is inferred for the input sequence $seq_i$, while training the model on the lagged input and target pair $(seq_t,\ y_i)$.

dataset shift occurs in most practical applications with causes ranging from a bias in the training data to a changing environment.

### 3.4.1 Online Learning

Online learning algorithms are an often effective way of dealing with the non-stationarity in data caused by dataset shift[2]. Whereas static ML models will often degrade in accuracy over time as the data changes, continuous incremental updates can not only keep the model fresh, exposure to new varieties of data might even improve it. Pagels[38] lists two common approaches for dealing with dataset shift, which are to train a model on the new data either manually or automatically on a schedule. Such a model will however always be lagging behind slightly in fast changing environments. A much more current can be produced by training in near-real time on individual samples or small batches as they are encountered. Care must be taken with the hyperparameters and the learning rate must be set appropriately, as online learning is naturally more prone to catastrophic inference, which occurs when a model forgets old information already learned.

Online learning relies on the truth being observable in the data, as the algorithms would otherwise not know how to adjust the parameters. Examples of applications that fit this description are time series such as sensor readings or the prediction of the next character or word typed on a smartphone. Figure 3.3 shows how a prediction can be made to serve a client while simultaneously training the model on the same request.

## 3.5 Non-IID Data in Decentralized ML

Underlying SGD is the assumption that data is IID, i.e. it is independent and identically distributed, which enables the estimation of the full gradient through a stochastic process[9]. However, in the geographically distributed edge setting and FL, it cannot be assumed that data is independent and identically distributed.

The remainder of this subsection will present a categorization of the types and sources of non-IID data as proposed by Hsieh et al.[10]. Their assumed setting of disjoint data partitions, each containing data collected from devices such as mobile phones or other sensors in a particular geographic location and time window closely resembles the setting the approach proposed by this thesis is designed for.

The independence assumption can be violated by *intra-partition correlation* and *inter-partition correlation*. The first is the case when the data from within a partition is processed in a specific order such as time of collection. Examples of this are when consecutive frames of a video or sensor readings in time series forecasting are processed. Inter-partition correlation can be present when devices across different partitions share common features. Neighboring geographic locations usually experience similar weather patterns and have the same diurnal effects such as daylight.

The identicalness property in turn is violated when the samples in each partition are not drawn from the same overall distribution. *Quantity skew* occurs when different partitions contain sufficiently different amounts of data, which can happen when some partitions collect data from fewer devices or less active devices. The distribution of labels can also vary across partitions since they are tied to specific locations, creating *label distribution skew*. For example, in image recognition tasks kangaroos only rarely show up outside of Australia and zoos. Another cause for non-identicalness is when the *same label* is given to *different features* in different partitions. The architecture of houses across the world varies greatly, as do clothing items. Time can also affect the features associated with a label, e.g. cars might be covered in snow during the winter and a house will look different at night. On the other hand, the *same features* can have *different labels* across partitions. Hsieh et al. give the example of next word prediction on mobile phones which is influenced by both personal and regional biases.

Kairouz et al.[8] further identify the problem of *feature distribution skew*, which is a form of covariate shift. In this type of identicalness violation, the distribution of samples skews with regards to independent features. Examples for this are handwriting recognition where users might write the same words with different stroke widths and slants or face recognition on photographs of users of different ages.

As noted by Hsieh et al.[10], dataset shift (concept drift) and geo-location are natural causes for non-IID data.

CHAPTER 4

# Model Branching

This thesis targets the commonly used edge ML architecture depicted in Figure 4.1 in which data from geographically distributed sources such as IoT sensors is processed on nearby edge nodes that host ML models. The nodes run inference tasks such as classification, detection or prediction of future events while simultaneously updating the model in near real-time to keep it current with regards to the possibly evolving data. Since the overall data distribution is partitioned by the catchment of data sources assigned to each node, each node can only train its model on a non-IID subset of the overall dataset. To improve generalization of the node local models, they are periodically synchronized using a centralized parameter server.

While training models on more diverse data will normally lead to better generalization and therefore improved accuracy, a model trained on highly heterogeneous data distributions might be so far removed from the typical data actually encountered at individual nodes that it performs worse than a model trained purely on the locally available subset of data present on that node[18, 26]. The danger of non-IID data partitions is that global accuracy has little predictive power for local accuracy at the node level, yet optimizing for this global accuracy has traditionally been the focus of research[18].

Since a single model that was trained while optimizing for global accuracy does not always provide the best results locally, model personalization approaches have been investigated as one possible remedy in FL in recent years. Fine-tuning of the global model through a small number of gradient descent steps using local data is a popular method to create a more personalized model which can then be used for inference. This is a sensible solution for regular, non-online machine learning where the model does not change, since the resulting personalized model will be used for a substantial amount of time following personalization. However, such a personalization step would have to be carried out frequently in online learning setups, making any personalized model short-lived and the process costly. While offline ML algorithms only have to produce an accurate model

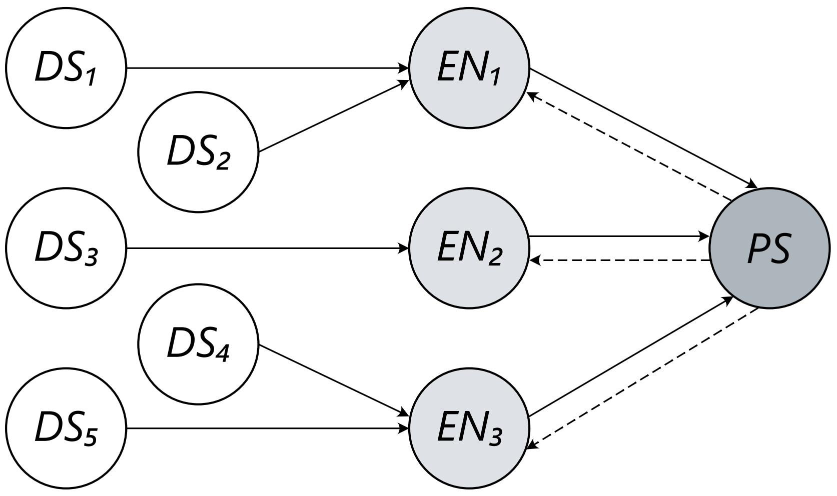


Figure 4.1: Architecture for distributed ML on the edge[2]. Edge nodes (ENs) receive requests from the data streams (DSs) assigned to them, which they act on and utilize to train their models. The parameter server (PS) aggregates model updates from the ENs and broadcasts the resulting global model back to the ENs.

as the end result, the goal of online learning is to achieve the best possible inference accuracy at all times throughout the entire processes.

The key insight that sparked this work is that many synchronization schemes already run a personalization step as part of the optimization process at the nodes, but do not actually utilize it since the schemes are designed for the offline learning setting. Algorithms such as *FedAvg*[4] iteratively create a global model by solving node-local optimization problems and then aggregating the resulting local models. In the next iteration, this new model replaces the local models resulting from tuning the previous global model to each nodes' data. Since only the nodes are privy to the data specific to their catchment, they are in the best position to estimate whether the new global model performs better on the live inference task they must also complete. Giving nodes the option to continue using – and training – the models that are only considered byproducts in the literature, while in parallel participating in the global optimization process as normal, is, to the best knowledge of the author, a novel model personalization technique with a natural fit for online machine learning on the edge. Temporarily decoupling from the synchronized global model for inference purposes can be understood as branching off the main model lineage, lending the name for this approach.

In settings where all data partitions across nodes are perfectly IID there is little need to utilize model branching as existing algorithms are well equipped to handle this case already. Alternatively, when the partitions at some or even all nodes are highly non-IID or subject to dataset shift, this personalization approach can improve inference performance. The default mode of operation of the model branching algorithm proposed in this thesis is to stay on the synchronized mainline branch that is the global model. Should a node however encounter data that will provide high utility to the inference quality for local data points in the near future, it is better served by branching to protect the resulting model parameter configuration. Such an effect can be observed for example when the data at only one node starts to drift from the remainder in a local phenomenon. In fact, it might even be preferable to hide the data from the global model in cases where it is such an extreme outlier that it would only reduce the accuracy of the model by pushing the parameters away from near-optimality. Such cases will however not be considered further in this work.

Deciding when to branch is a question critical to the success of this personalization approach, as each branch incurs cost in the form of the required additional computation and overly eager branching can lead to insular overfitting that leaves the node vulnerable to higher generalization errors. For this reason, the decision to branch must be constantly re-evaluated to ensure that new data from other nodes is considered as well. As the goal of this thesis is to demonstrate initial validity of the approach, the used heuristics are intentionally kept simple and do not require the nodes to communicate with other actors aside from the central server. Although inter-node communication might improve decision quality, it could also violate privacy concerns depending on what information is shared.

## 4.1 FedAvg-MB

Model branching is implemented in this thesis as an online learning variation of the popular federated learning algorithm *FedAvg*[4]. The original is shown in Algorithm 4.1 and the model branching enabled version for online learning in Algorithm 4.2.

*FedAvg* is split into two parts, the global aggregation step executed on a central server and the local part run on clients or, in this case, nodes. The parameter server (PS) must first initialize the parameters of the model to be trained (line 2). Iteratively, the server then disseminates the model to the nodes before aggregating and reincorporating their updates. At the start of each round a subset of the participating nodes is randomly selected(lines 4, 5). The current model is then sent to all nodes (line 7) which conduct an update using their local data partition. To do so they split their data partition into minibatches (line 12) and train the model (line 15) for the configured number of epochs.

While *FedAvg* deals with participating nodes that might not always be available, this work focuses on more stable nodes typically hosted in smaller data centers. As all nodes continually receive new data from the clients in their catchments that might reflect emerging global trends, *FedAvg-MB* modifies the node selection to always include all nodes, which is equivalent to setting the fraction parameter $C = 1$ in *FedAvg*. This first adjustment brings the algorithm more in line with the bulk-synchronous algorithms known from distributed learning, but leaves the parameter server part unchanged otherwise.

After each synchronization with the parameter server, the nodes receive the newly aggregated global model (line 13). Using data from previous iterations, each node now decides which model to use for inference during this iteration by running a heuristic comparing the performance of the global model, the model used for inference in the previous iteration and the previously locally trained global model (line 14). Depending on the branching decision in iteration $t-1$, only the current global model and the locally trained one from the last iteration might have to be considered at this step.

In this online version of the algorithm, the data streams have to contain both training and live data for inference, although it could trivially be adjusted to accept decoupled training and live data at different points in time. For time series data specifically, training data can also be extracted from the inference request by lagging it for the number of forecasting steps required as seen in Figure 3.3.

As a node receives data (line 16) it will serve the inference requests right away using the selected model (line 17). Once the node has acquired enough data to meet the local minibatch size $B$, it will train the synchronized global model as well as the model chosen by the heuristic, if it is not identical, $E$ times (lines 20 to 23). Finally, the trained global model received at the start of the iteration is sent back to the parameter server where it is incorporated into the new global model for the following iteration.

*FedAvg-MB* is an online learning algorithm that continually receives new data to be processed and is thus necessarily stateful, unlike the offline *FedAvg*. This has advantages in terms of communication cost and privacy, since no data from any clients ever leaves the edge node. During synchronization, requests from data streams could either be blocked until synchronization is complete or served by the previously chosen model without training.

### 4.1.1 Complexity

*FedAvg-MB* extends *FedAvg* in several crucial ways that should be taken into consideration in edge deployments with limited processing power.

While the original algorithm is used for offline learning where the only goal is to train a model, the online variant *FedAvg-MB* must also serve inference requests. The costs of training and inference both depend on the used neural network architecture, i.e. the number of layers and cells and their accompanying parameters. Let $t$ and $i$ be the costs of training and inference for the used architecture. It can further be assumed that training costs dominate those of inference due to the additional backpropagation and parameter adjustment steps. The costs of both tasks increase linearly with the number of inference requests and available training samples each round.

A straightforward online learning port of *FedAvg* carries out inference and training on the available data samples once per iteration. The time complexity of the node part of

**Algorithm 4.1:** FedAvg[4]

```
Input: $K$ clients indexed by $k$; $C$ portion of clients selected each round; local
       minibatch size $B$; local epoch number $E$; learning rate $\eta$
Executed on parameter server:
    initialize $w_0$
    for each round $t = 1, 2, ...$ do
        $m \leftarrow max(C * K, 1)$
        $S_t \leftarrow$ (random set of $m$ clients);
        for each client $k \in S_t$ do
            $w_{t+1}^k \leftarrow ClientUpdate(k, w_t)$
        end
        $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
    end
// Executed on node $k$
ClientUpdate($k, w$):
    $\mathcal{B} \leftarrow$ (split $P_k$ into batches of size $B$)
    for each local epoch $i$ from $1$ to $E$ do
        for batch $b \in \mathcal{B}$ do
            $w \leftarrow w - \eta \nabla f(w, b)$
        end
    end
    return $w$
```

*FedAvg* can thus be given as $\mathcal{O}(t \cdot t_n + i \cdot i_n)$ where $t_n$ and $i_n$ are the counts of training and inference data.

The multi-model approach of *FedAvg-MB* means that either two or three models must be trained at every node during each round. Additionally, a branching decision must be made at the start of every iteration. This decision scales linearly with the number of models trained in the previous iteration. The extra training and the potentially expensive decision function $d$ lead to a worst case upper bound of $\mathcal{O}(3 \cdot t \cdot t_n + i \cdot i_n + 3 \cdot d)$ that is comparable to *FedAvg* once the constant factors are dropped.

The code for the central aggregating parameter server is nearly identical in the two algorithms and only differs in participant selection and number of updates. Model aggregation is done by averaging all received model parameters and has a time complexity of $\mathcal{O}(n \cdot w)$, where $n$ is the number of participating nodes in the iteration of question and $w$ the number of parameters the model for the neural network consists of. Operations of the parameter server can further be hosted on powerful instances in the cloud.

**Algorithm 4.2:** FedAvg-MB

```
    Input: $K$ nodes indexed by $k$; local minibatch size $B$; local epoch number $E$;
           learning rate $\eta$
 1  Executed on parameter server:
 2      initialize $w_0$
 3      $t \leftarrow 0$
 4      while $True$ do
 5          for each node $k \in K$ in parallel do
 6              Send $w_t$ to node $k$
 7              Receive $w_{t+1}$ from node $k$
 8              $t \leftarrow t + 1$
 9          end
10          $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
11      end
12  Executed on node $k$ at iteration $t$:
13      $w_t \leftarrow$ receive from server
14      $w_t^i \leftarrow bestForInference([w_t, w_{t-1}, w_{t-1}^i], [P_{t-1}^k, P_{t-2}^k, ...])$
15      while $length(P_t^k) < B$ do
16          $inference\_request, training\_data \leftarrow$ receive from data streams
17          Act on $inference\_request$ using $w_t^i$
18          $P_t^k \leftarrow P_t^k \cup training\_data$
19      end
20      for each local epoch $i$ from 1 to $E$ do
            // Training for the global model
21          $w_t \leftarrow w_t - \eta \nabla f(w_t, P_t^k)$
            // Training for the branched model
22          $w_t^i \leftarrow w_t^i - \eta \nabla f(w_t^i, P_t^k)$
23      end
24      Send $w_t$ to PS
```

CHAPTER 5

# Experimental Setup

To validate the proposed model branching algorithm, a simulation testbed is created to empirically measure and compare performance with baseline algorithms using several developed branching decision heuristics on three real-world and one synthetically generated dataset under varying parameters. This chapter introduces the experimental setup. All code is publicly available[1].

## 5.1 Simulation

To ease development and save on infrastructure cost, the implementation of the experiment architecture targets a single machine that simulates independent worker nodes by using operating system processes. PyTorch[39] is chosen as the underlying deep learning library because it was observed that an increasing portion of the research community is moving towards it. This library choice in turn prompted the use of the Python programming language[2] as, although APIs for other programming languages exist, Python enjoys favorable status which is reflected in the higher number of resources available. The decision to use Python extends to various helper tools and later evaluation of the results to reduce the number of dependencies. NumPy[40], the popular array programming library, and Pandas[41], a data analysis library, are used pervasively throughout the implementation.

Figure 5.1 visualizes the architecture using a deployment diagram. Parameter server and the global node are deployed together with the pre-trained model for the given task. The global node is peculiar to this simulation and is used only for evaluation of the proposed approach; there is no equivalent in real-world deployments. The edge nodes receive a new model from the parameter server at the start of each iteration and send back their

[1]https://github.com/schwarz/model_branching
[2]https://www.python.org/

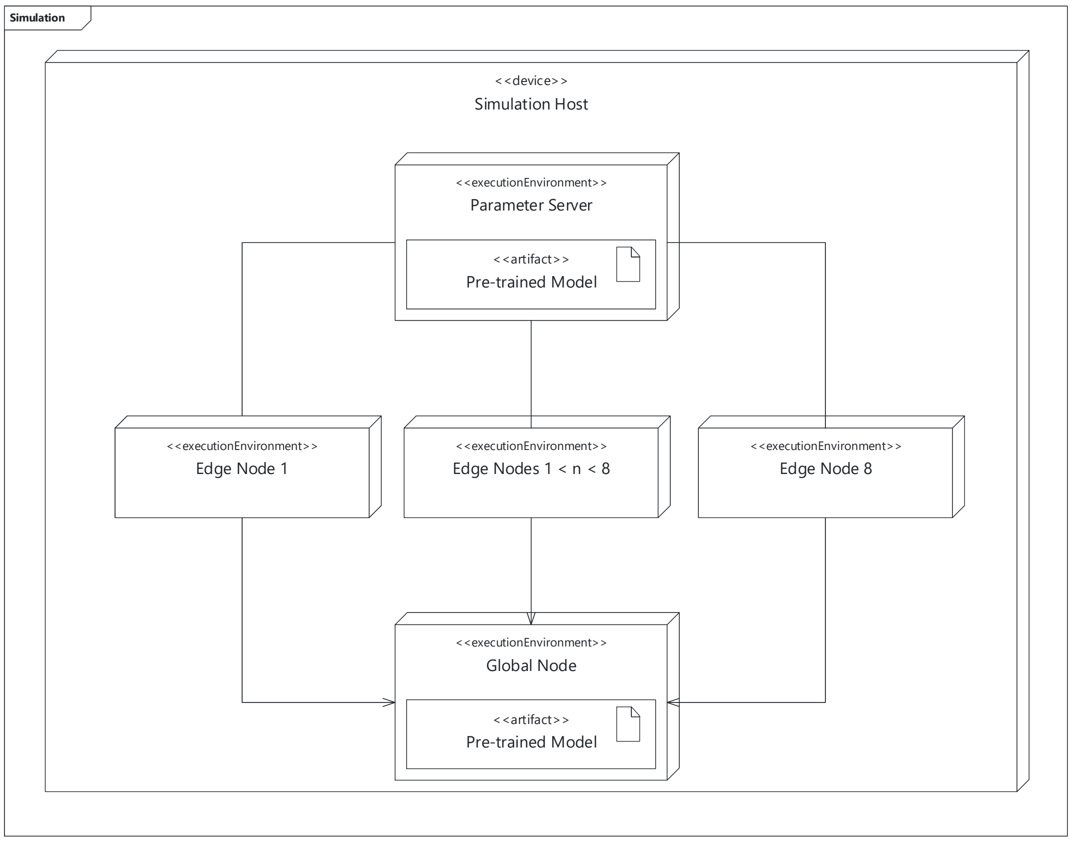


Figure 5.1: Deployment diagram for the simulation setup.

updated version. The edge nodes also pass on their minibatches of data to the global node each round. The global node then uses this data to evaluate and update its own model. The interactions between parameter server, edge nodes and the global node are shown in Figure 5.2 and explained in more detail in the following subsections.

All experiment runs are conducted on a machine with the following hardware:

- *CPU:* Intel i5 4570
- *GPU:* Palit Geforce GTX 1060 with 6075MiB memory
- *RAM:* 2x Ballistix 8GiB DDR3

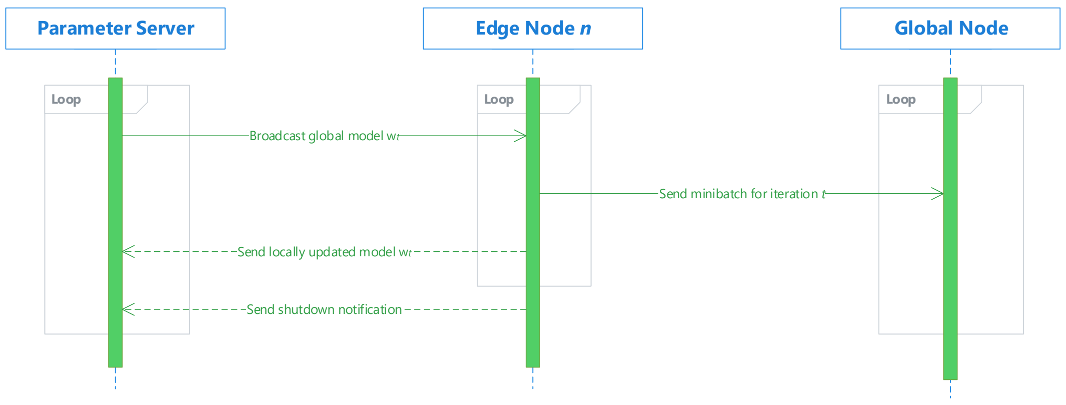

Parameter Server
Edge Node *n*
Global Node
Loop
Loop
Loop
Broadcast global model $w_t$
Send minibatch for iteration $t$
Send locally updated model $w_t$
Send shutdown notification

Figure 5.2: Sequence diagram for the simulation setup.

### 5.1.1 Parameter Server

The parameter server process is a straightforward implementation[3] of the server component as described by McMahan et al.[4], although there is no node selection processes since all nodes participate in all rounds. The parameter server first distributes the pre-trained model to all nodes, then continually runs loops of aggregating model updates from the nodes to create the model for the next round before broadcasting it again. In this simulation, nodes participate until they run out of test data, at which point they notify the parameter server before dropping out. This synchronization process is synchronous; the parameter server waits for updates from all nodes before moving on.

### 5.1.2 Edge Nodes

Each simulated edge node[4] is executed on a separate processes, implementing the node part of *FedAvg-MB*. The implementation differs in that the nodes run several branching decision algorithms simultaneously in order to measure their outcome performances.

The nodes run iterations in coordination with the parameter server. At the start of each iteration, the nodes receive the current global model. During initializing, nodes are passed a list of file paths which they process fully before moving to the next set of files. After loading a minibatch from the current data files, which they also send to the global node, the nodes run each branching decision heuristic and keep the models or model branches that were selected alive. The nodes then evaluate each model chosen by the heuristics against the actual data to measure the outcome. Finally, the global model and all models selected by heuristics are trained for a configurable number of times on the

[3]See `parameter_server.py`
[4]See `node.py`

minibatch. The locally updated global model is then sent back to the parameter server where it is incorporated into the global model for the next iteration. The nodes continue in this loop as long as they are able to load more minibatches.

The following branching heuristics are designed and implemented for the time series forecasting tasks used in this thesis:

1. *Best Last {1, 2, 3}:* Chooses the model with the lowest summed error for the $n$ previous iterations.
2. *Best Improving:* Similar to Best Last 3, but puts increasing weight on more current iterations.
3. *Greedy Oracle:* Looking ahead, it chooses the model that will produce the lowest error for the current iteration. This short-sighted heuristics only creates a branch when it is opportune and cannot perform worse than the global model.
4. *Random:* Selects a random model.

The hypothesis that the locality principle plays a critical role in this setting guides the two classes of heuristics *Best Last* and *Best Improving*. If the principle does apply, then a model that performs better on data from the recent past will continue to do so for the immediate future. The *Best Last* heuristics calculate the losses of the available models for the $n$ previous iterations, choosing the model with the lowest sum. By increasing $n$, the heuristics should become less susceptible to outliers and make more robust decisions, although this also makes it slower react to new trends. The *Best Improving* heuristic adjusts *Best Last 3* by placing additional weight on more recent iterations. These heuristics require costly model inference steps that increase with the number of considered iterations, which influenced the decision to use only a maximum of three most recent iterations as inputs. These heuristics also do not require communication with other nodes or the parameter server and can be implemented using information available only to the individual edge node.

The *Greedy Oracle* and *Random* heuristics are implemented as controls for the others. The *Greedy Oracle* heuristic cannot be replicated in the real world, as it cheats by using information a node does not have under normal circumstances. This allows the heuristic to greedily pick the better model for immediate inference results. However, this heuristic is also quick to give up on branches, ignoring the possibility that they could gain in the long run. Nonetheless, this heuristic presents a reference point for possible improvement. Since the *Random* heuristic makes an uninformed, random decision it cannot be expected to yield consistent results unless the purely local model is significantly better than the synchronized model. It should thus always be outperformed by a reasonable informed heuristic.

### 5.1.3 Global Node

The global node[5] simulates results for when data from all nodes is processed at a central location, training a single model along the way, and is used solely to establish an additional benchmark of performance in this thesis. This node receives the data all the edge nodes process during each iteration and evaluates the accuracy of prediction this global model has. It then updates its model before pooling the data for the next iteration from the edge nodes.

Since the goal of offline algorithms like *FedAvg* is to converge on models with similar accuracies a central node such as this would achieve, this sets an ideal target for all personalization algorithms.

## 5.2 Datasets

For the experiments, three real-world datasets containing time series that match the described use case of the algorithm are selected. Additionally, generators for a synthetic dataset are created in order to have more control over the data.

### 5.2.1 Demand

The `Demand` dataset[42][6] contains power consumption profiles for 200 households in the Midwest region of the United States. Each household is represented by a time series of electricity demand measured in watts at a point in time with a continuous ten minute resolution, spanning the whole year of 2010. The dataset contains no faulty data. Figure 5.3 shows a week of data for a random household.

For the experiments, the dataset is trimmed to only include data from February to August. Since the demand profile can be erratic, hourly averages are calculated and used instead to reduce unforeseeable jumps. The data for each household is chronologically split into 70 %, 10 % and 20 % for training, validation and online testing, respectively. This split is aimed to create a shift in the dataset as energy demands increase during the warmer summer months, which could potentially be attributed to increased air conditioning usage. Figure 5.4 shows the mean monthly energy demand across all households. Since the values can vary widely, they are scaled down through division by 10,600, which covers the highest recorded household demand, to help convergence[43].

### 5.2.2 FitRec

The `FitRec` dataset[44][7] introduced by Ni et al. is based on public fitness tracking data sourced from Endomondo[8]. The data is partitioned by workout bouts that fall under

[5]See `perfect_node.py`
[6]`https://data.nrel.gov/submissions/69`
[7]`https://sites.google.com/eng.ucsd.edu/fitrec-project/home`
[8]`https://www.endomondo.com/`

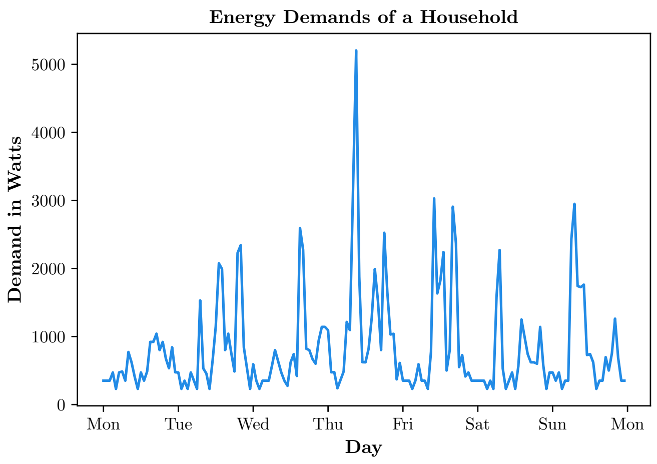


Figure 5.3: A week of energy demand created by a household in May.

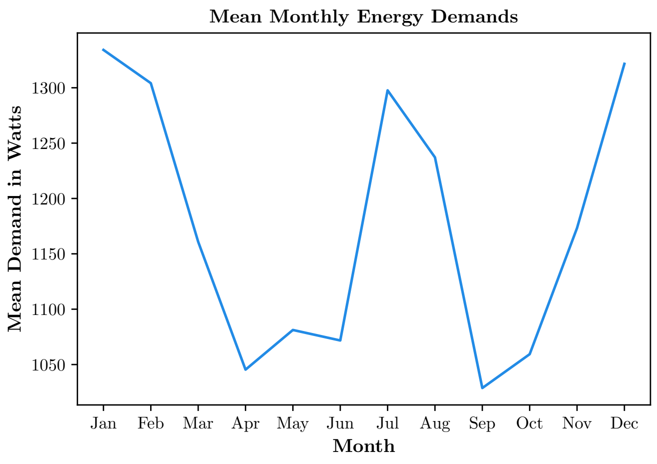


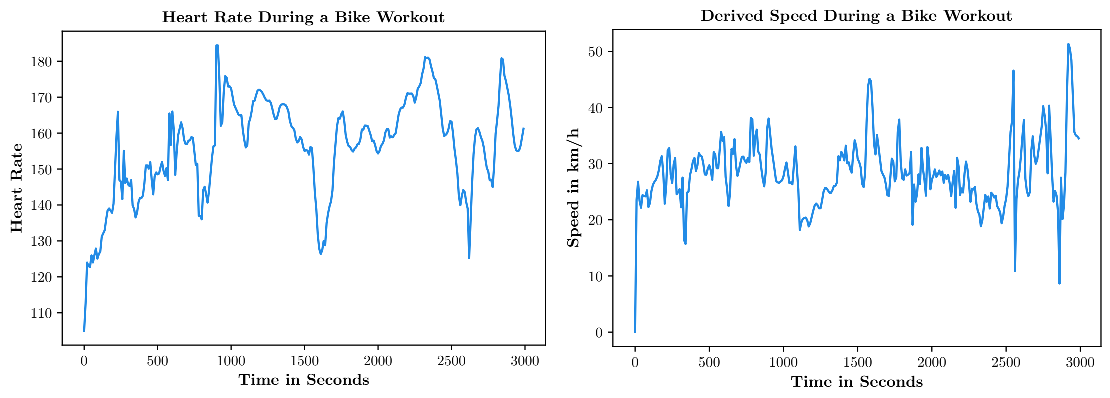
Figure 5.4: Mean monthly energy demands of all households.

a variety of activity types such as bicycling, running or snowshoeing. The re-sampled version with equidistant sampling includes 102,343 workouts across 887 users, each recording several metrics. In this thesis, only heart rate and derived speed are used for univariate time series forecasting.

In their work, Ni et al. normalized the measurements to gain the z-score, also called standard score, of the input sequence, while leaving the target sequence untouched. The measurements of the input sequences are de-normalized to their original values again, so that all datasets used in the simulation of this thesis can be handled in a uniform manner. The target sequences of the dataset are disregarded; only data from the input sequences are used for inference.

The dataset is split into 70 %, 10 % and 20 % for training, validation and online testing, respectively, by assigning entire workout sessions to one of the above sets. The recorded heart rates are further divided by 220 and the derived speeds by 50. The value of 220 beats per minutes is chosen for its common use in estimating the maximum heart rate of a person by subtracting their age from it, while the scale divisor of 50 km/h ensures a balance between sports with little movement and the ones where speeds exceeding this number are possible.

Figure 5.5 shows heart rate and derived speed for one randomly drawn bicycling workout session.



(a) Heart Rate (b) Derived Speed

Figure 5.5: Heart rate and derived speed recorded during one exemplary bike workout.

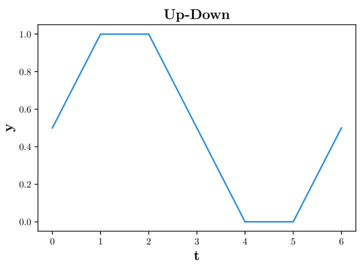
### 5.2.3 Synthetic

The `Synthetic` dataset is the only fully artificial dataset used in the experiments. A literature review yielded no suitable synthetic dataset or generator, prompting a custom design[9]. The goal of this dataset is to provide distinct classes of data for the creation of

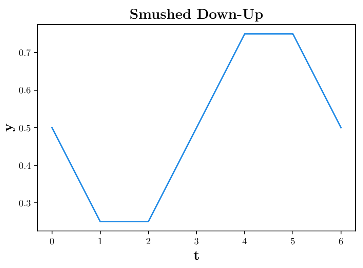
[9]See `data/synthetic/0_generate.py`

a non-IID univariate time series dataset. In general, only trend and seasonal patterns can be extrapolated in univariate time series forecasts[45]. The classes are thus designed to differ in their seasonal patterns, but the same overall trend is applied later on as a form of dataset shift.

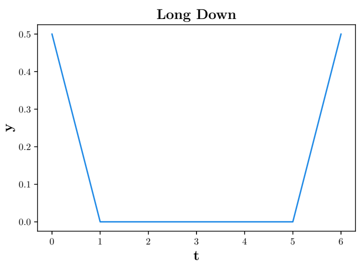
Figure 5.6 shows the nine classes of generated patterns used in this work.



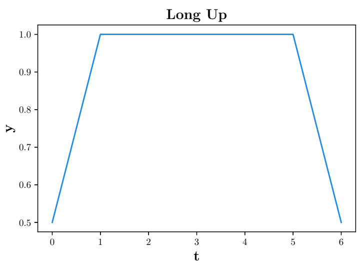
(a) Up-Down



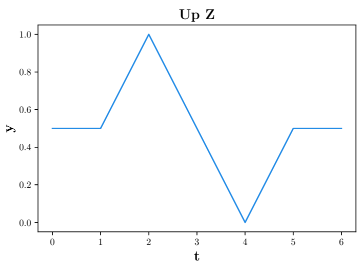
(b) Smushed Down-Up



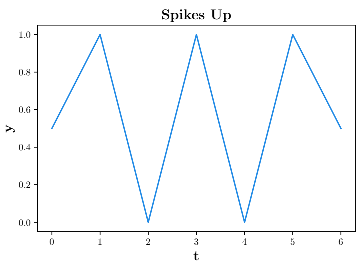
(c) Long Down



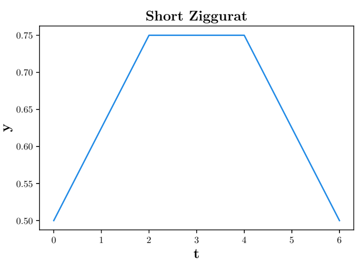
(d) Long Up



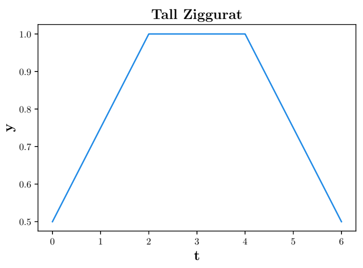
(e) Up Z



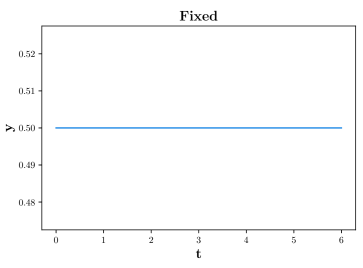
(f) Spikes Up



(g) Short Ziggurat



(h) Tall Ziggurat



(i) Fixed

Figure 5.6: Base patterns of the curves in the synthetic dataset.

The number of included patterns is based on the single digit number of planned simulated nodes. All nine classes are used for training and validation, but data of the `Spikes Up` class is dropped from testing due to hardware limitations. This class was randomly chosen for exclusion.

Each class is primarily made up of a seasonal pattern, which is defined on a floating point scale from zero to one and has a fixed length of seven, which is the cycle frequency in the final time series. The patterns are then repeated to reach the desired number of cycles and multiplied by normally distributed noise, before the values are clamped back into the zero to one range. The values are halved so that the time series can be moved to a uniformly distributed offset for additional variety before a 10 % increasing trend is applied. A final clamp is applied to make sure the values are contained in the zero to one range.

Additionally, three time series modifiers are developed and applied to a subset of the generated data:

1. *Upper Halved*: Subtract 50 % of the delta to the median of data points above the median.
2. *Lower Halved*: Subtract 50 % of the delta of data points below the median.
3. *Zeroed*: Sets all values to zero.

These modifiers are used to create local shifts in the data. While the `Upper Halved` and `Lower Halved` modifiers present mild but distinct mutations on top of the existing patterns, the `Zeroed` modifier creates a drastic pattern change for all but the `Fixed` class. This last modifier resembles a possible failure state of a broken sensor.

1,000 curves of uniformly distributed patterns are generated with 2,000 values each. The modifiers are applied to a randomly chosen 10 % of these curves. The first 70 % of data points for each curve are used for training, the following 10 % for validation of the model training process and the final 20 % are used for the online testing experiments.

### 5.2.4 Turnstile

The `Turnstile` dataset[10] contains entry counts for turnstiles of the New York City Subway with a regular recording interval of four hours from November 2014 to June 2020.

The data published by the MTA displays some peculiarities as identified by Whong[46]:

- The observations are not counts for a time frame, but compound like an odometer.
- When an unknown turnstile specific limit is reached, the count resets to zero.
- Measurements are regularly taken every four hours, but there might be additional readings and the times can differ between turnstiles and even change for a turnstile.

Additionally, some turnstiles count down instead of up and a turnstile may be out of order for periods of time during which would consequently have no recorded measurements. The dataset is pre-processed in this thesis to prevent these issues from overly affecting results. Irregular readings are removed to obtain a mostly uniform sampling interval with the exception of when the measurement times shift. The readings are re-shaped to

[10]`http://web.mta.info/developers/turnstile.html`

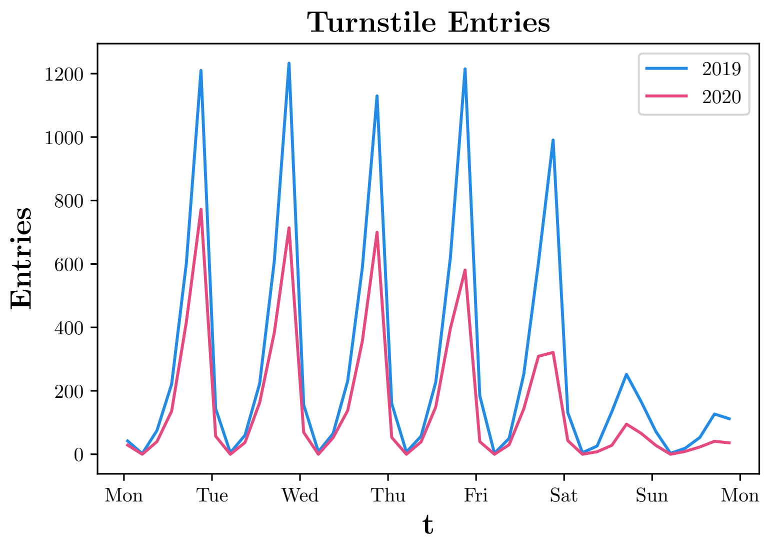


Figure 5.7: Entries for a turnstile in Grand Central-42nd Street in March 2019 and a year later in March 2020, influenced by the COVID-19 outbreak.

resemble the difference to the previous measurement instead of a total count and the zero rollover is handled by limiting the delta to a maximum of 10,000 entries, which could only be reached by an average of 41 people passing through a turnstile per minute for four continuous hours. To deal with rollover and downwards measuring turnstiles, the absolute value of the difference is used.

The dataset is chronologically split into training, validation and test data. The entire month of December in 2019 is used for validation, while all available measurements before that are used for training. Entries from January 2020 to June 2020 are used for testing. Due to the outbreak of the COVID-19 pandemic, the test dataset is subject to dataset shift, manifested through drastically reduced ridership numbers. Finally, the turnstile entries are divided by 5,000 to aid training. This number is chosen as the maximum entry number of 10,000 is extraordinary.

Figure 5.7 compares a week of entries for a turnstile in one of the busiest stations in New York City (Grand Central-42nd Street) in March 2019 to the week of March 9th, 2020 which is the day state of emergency was declared in the city. The comparison shows the gradual but swift drop in ridership caused by this event.

## 5.3 Models

For all datasets, a network architecture inspired by Chen at al.[16] consisting of one Long Short-Term Memory (LSTM)[47] layer followed by a linear output layer is used. LSTMs, a type of recurrent neural network (RNN), are chosen because they are known to perform well on time series forecasting tasks[48][49]. For simplicity, the models predict only a single value into the future.

Pre-trained models are used in the experiments to circumvent the cold-start problem and mimic real-world deployments. Since centrally trained models appear to have lower potential for personalization than models trained using *FedAvg*[6], the models for each dataset are trained through a centralized, single process implementation of *FedAvg* using the Adam optimizer[50], which can give good results even with sub-optimal hyperparameters[51][11]. The algorithm is faithfully implemented, requiring a trade-off to be made between inclusion of all data producers and number of rounds. Fraction of clients per round and number of rounds are arbitrarily chosen with the number of data producers in mind and tested to ensure that training losses stabilize.

For each dataset, a model is trained for every learning rate in {0.05, 0.01, 0.005, 0.00, 0.0005, 0.0003, 0.0001, 0.00005, 0.00001}. The resulting models are then evaluated against the validation set and the model producing the lowest Mean Absolute Error (MAE), see Equation 5.1, is carried over into the online experiments.

$$MAE = \frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|}{n} \tag{5.1}$$

All models are trained using Google Colaboratory[12] Pro GPU enabled instances to speed up the training process.

## 5.3.1 Demand

The smallest seasonal cycle in this dataset with a frequency of 24 observations is a day. Additionally, there is a strong weekly component, but this is not paid any attention since the window size to capture just a day is already large. Training parameters are listed in Table 5.1.

Figure 5.8 compares actual values for two random days from the validation set and a forecast in which previously predicted values are used as inputs for subsequent ones. The model is able to capture a weak general seasonality.

## 5.3.2 FitRec

Two models are pre-trained for the FitRec dataset, one to predict heart rate and one for the derived speed. Tables 5.2a and 5.2b contain the training parameters for the heart rate and derived speed model, respectively.

The window size of nine is adopted from the original paper by Ni et al.[44].

Figures 5.9a and 5.9b compare the actual heart rate and derived speed for a workout from the validation set with forecasts by the models. As more predicted values serve as

[11]See the `generic_model_training.ipynb` notebook

[12]`https://colab.research.google.com/`

| **Parameter** | **Value** |
|---|---|
| Window size | 24 |
| Loss function | MAE |
| Learning rate | 0.0003 |
| Fraction C | 0.05 |
| Epochs E | 1 |
| Local minibatch size B | 10 |
| Rounds | 25 |

Table 5.1: Demand model training parameters.

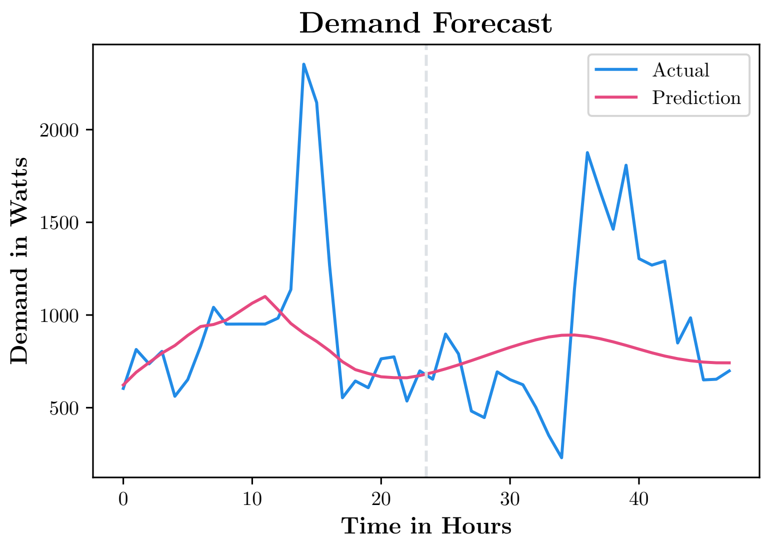


Figure 5.8: Demand: Example of actual versus predicted values. Predictions past the dashed line are made purely on previously predicted values.

| Parameter | Value |
|---|---|
| Window size | 9 |
| Loss function | MAE |
| Learning rate | 0.0003 |
| Fraction C | 0.0005 |
| Epochs E | 1 |
| Local minibatch size B | 10 |
| Rounds | 1000 |

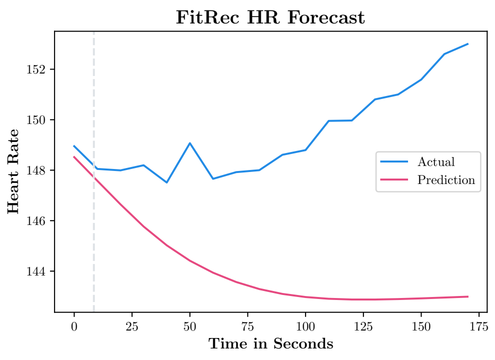
(a) Heart Rate

| Parameter | Value |
|---|---|
| Window size | 9 |
| Loss function | MAE |
| Learning rate | 0.0005 |
| Fraction C | 0.0005 |
| Epochs E | 1 |
| Local minibatch size B | 10 |
| Rounds | 1000 |

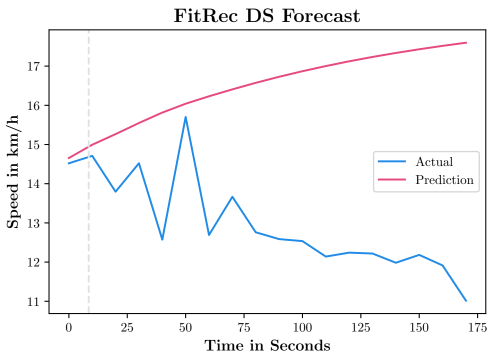
(b) Derived Speed

Table 5.2: FitRec model training parameters.

the input, the model can only continue on the trend observed in the original workout values.



(a) Heart Rate



(b) Derived Speed

Figure 5.9: FitRec HR & DS: Example of actual versus predicted values. Predictions past the dashed line are made solely on previously predicted values.

### 5.3.3 Synthetic

The training parameters for the `Synthetic` dataset can be found in Table 5.3. To capture a full cycle of the inherent pattern, a window size of seven is chosen.

Figure 5.10 shows the forecast quality of the model.The original data has a very strong seasonal pattern, which the model is able to reproduce.

### 5.3.4 Turnstile

Turnstiles undergo two prominent seasonal cycles: days and weeks. A window size of 12 is chosen to capture the primary daily cycle. Depending on the station, ridership

| **Parameter** | **Value** |
|---|---|
| Window size | 7 |
| Loss function | MAE |
| Learning rate | 0.0001 |
| Fraction C | 0.01 |
| Epochs E | 1 |
| Local minibatch size B | 10 |
| Rounds | 250 |

Table 5.3: Synthetic model training parameters.

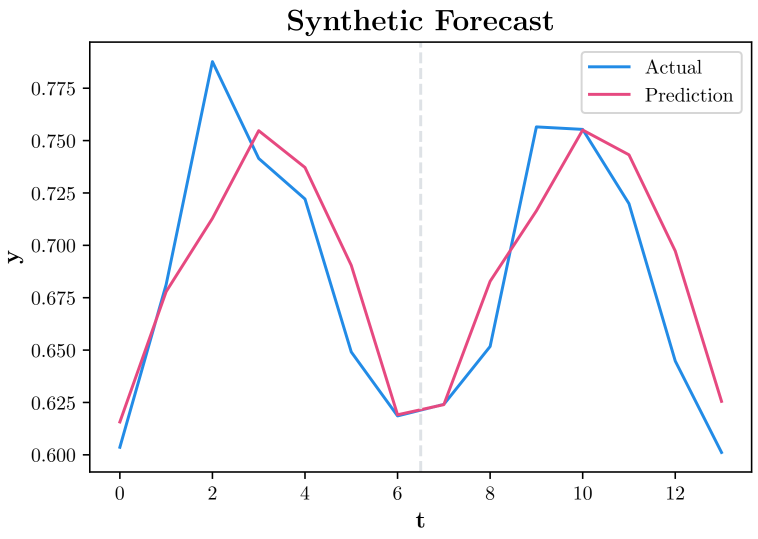


Figure 5.10: Synthetic: Example of actual versus predicted values. Predictions past the dashed line are made purely on previously predicted values.

| Parameter | Value |
|---|---|
| Window size | 12 |
| Loss function | MAE |
| Learning rate | 0.0001 |
| Fraction C | 0.002 |
| Epochs E | 1 |
| Local minibatch size B | 10 |
| Rounds | 500 |

Table 5.4: Turnstile entry model training parameters.

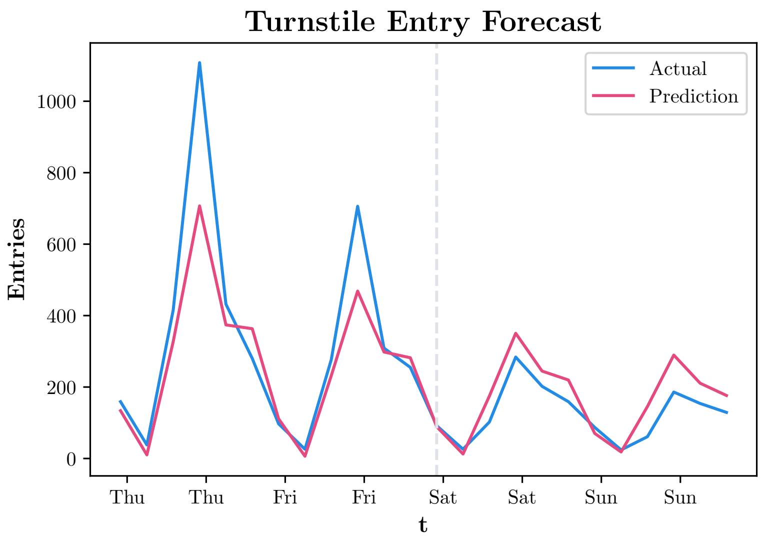


Figure 5.11: Turnstile: Example of actual versus predicted values. Predictions past the dashed line are made purely on previously predicted values.

numbers may be higher or lower on the weekends; this two-day window size is chosen with the hope that the model extracts the transition as a feature.

In Figure 5.11, the entries for day in the validation set from a random turnstile are compared to an inductive forecast by the model.

## 5.4 Node Assignment

For all datasets, two assignments that map data streams to nodes are created. One of the configurations is designed to expose the nodes to a non-IID data distribution where prominent features are split such that each node only sees a certain type of data. The other configuration takes the previously created assignment and shuffles it for a more IID distribution where the branching approach is expected to be less effective.

The simulated nodes can read minibatches from several files at once and turn over to a new set of files once all data has been processed from the previous set. While the assignments for the `Demand` and `Turnstile` datasets only have one such set per node, the `FitRec` and `Synthetic` datasets have many sets of files that the nodes processes throughout the simulation. Dataset shift in the latter two is induced through the transition of sets of files, while the earlier two datasets have a continuous trend in the case of the `Demand` dataset or an abrupt shift for the `Turnstile` dataset.

All configurations make use of eight nodes, which is the maximum number of nodes the used hardware is able to simulate concurrently with GPU memory being the limiting factor.

CHAPTER 6

# Experimental Results

This chapter presents and analyzes the outcomes of the experiments. Results for the following branching heuristics introduced in the previous chapter using the *FedAvg-MB* algorithm are reported:

- *Best Last 3*
- *Best Improving*
- *Greedy Oracle*
- *Random*

The results for *Best Last 1* and *Best Last 2* are omitted for readability as they show considerably weaker performance than both *Best Last 3* and *Best Improving* on average. To rate the performance of the developed branching algorithms, four baseline strategies are further used and presented:

- *Synced:* Online variant of *FedAvg* without branching.
- *Global:* The combined iteration minibatches of the nodes are processed on a single node, no branching.
- *Local:* The nodes train the initial model purely on local data and never synchronize it, resulting in a single local branch.
- *Static:* The unchanged initial model.

The local and synchronized baselines are implemented as branching decision heuristics in the nodes, while the static baseline is a special heuristic that skips training. The global baseline is described in the previous chapter.

In addition to testing the algorithms on both IID and more non-IID node assignments, other parameters are varied as well to prevent single parameters from skewing the results. To this end, multiple configurations for the learning rate, the number of local training epochs on an iteration's minibatch as well as the minibatch size are chosen. Two online learning rates (LR) are picked for each model for the runs depending on the learning rate $lr$ the model was original trained with: $lr \cdot 0.5$ and $lr \cdot 0.1$. Local epoch (LE) number and minibatch size (BS) are varied and chosen from $\{1, 2\}$ and $\{2, 4\}$ respectively.

## 6.1 Result Reporting

For all of these regression tasks, the mean absolute error (MAE) between actual and predicted value is used as the primary performance measure. As part of the simulation, nodes log the error resulting from the different branching heuristics on their local data for each iteration. Reported errors are averaged for individual nodes and averaged again across nodes for the result summaries presented in the following. The branching decisions for each node and iteration are classified as being either better, equal or worse than simply following the synchronized baseline based on the MAEs for that iteration. The relative improvement of the mean MAE for each heuristic is calculated in relation to the synchronized baseline as well. In all tables, lower values for MAE are better. Numbers in bold represent the best performance, underlined numbers the second best.

Results are aggregated for this chapter. The full experimental results can be found in the source code repository.

## 6.2 Results

### 6.2.1 Demand

Compared to the other datasets in these experiments, the window size to capture a cycle is fairly large in the case of the `Demand` dataset. Due to this, the information learned in one iteration is not immediately relevant in the following ones. The results, summarized in Tables 6.1 and 6.2, show that the used heuristics, which primarily rely on the locality principle, are not able to yield an improvement over baseline in these scenarios.

While the regular branching strategies have no notable impact on inference accuracy in comparison to baseline in the non-IID setting, the oracle heuristic consistently extracts an improvement. Notably, the greedy oracle branching heuristic outperforms the synchronized model even when both local and global models are worse (Table 6.2g). The synchronized model resulting from the federation is very close to the globally trained and the purely local model in terms of error rates in both configurations. This suggests there

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 468.013 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 467.224 | 57 / 5 / 38 | 0.17 |
| Best Improving | 467.207 | 57 / 4 / 39 | 0.17 |
| Greedy Oracle | 467.172 | 65 / 35 / 0 | 0.18 |
| Random | 467.913 | 34 / 42 / 24 | 0.02 |
| Local | 463.485 | 62 / 0 / 38 | 0.97 |
| Global | **460.755** | 69 / 0 / 31 | 1.55 |
| Static | 487.878 | 26 / 0 / 74 | -4.24 |

(a) Learning Rate=0.00003, Local Epochs=2, Batch Size=2

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 430.750 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 433.393 | 38 / 19 / 42 | -0.61 |
| Best Improving | 433.663 | 39 / 17 / 44 | -0.68 |
| Greedy Oracle | **421.559** | 52 / 48 / 0 | 2.13 |
| Random | 432.219 | 29 / 40 / 31 | -0.34 |
| Local | 434.474 | 46 / 0 / 54 | -0.86 |
| Global | 437.190 | 40 / 0 / 60 | -1.50 |
| Static | 489.628 | 32 / 0 / 68 | -13.67 |

(b) Learning Rate=0.00015, Local Epochs=2, Batch Size=2

Table 6.1: Summary of Demand-IID. Results for the experimental configurations with the (a) worst and (b) best synchronized model by mean error across nodes are shown.

is not much room for improvement of the model for this network architecture beyond what is achieved by the oracle branching strategy.

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 463.288 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 462.490 | 54 / 8 / 38 | 0.17 |
| Best Improving | 462.494 | 54 / 7 / 39 | 0.17 |
| Greedy Oracle | 461.845 | 62 / 38 / 0 | 0.31 |
| Random | 463.128 | 35 / 40 / 25 | 0.03 |
| Local | 455.539 | 57 / 0 / 43 | 1.67 |
| Global | **452.902** | 61 / 0 / 39 | 2.24 |
| Static | 489.628 | 31 / 0 / 69 | -5.69 |

(a) LR=0.0003, LE=1, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 468.696 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 467.069 | 58 / 5 / 36 | 0.35 |
| Best Improving | 467.075 | 59 / 5 / 37 | 0.35 |
| Greedy Oracle | 467.803 | 64 / 36 / 0 | 0.19 |
| Random | 468.556 | 34 / 43 / 23 | 0.03 |
| Local | 462.088 | 61 / 0 / 39 | 1.41 |
| Global | **460.755** | 68 / 0 / 32 | 1.69 |
| Static | 487.878 | 26 / 0 / 74 | -4.09 |

(b) LR=0.0003, LE=1, BS=4

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 451.121 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 449.980 | 52 / 8 / 40 | 0.25 |
| Best Improving | 449.977 | 52 / 7 / 41 | 0.25 |
| Greedy Oracle | 448.428 | 60 / 40 / 0 | 0.60 |
| Random | 450.937 | 33 / 41 / 26 | 0.04 |
| Local | **444.399** | 53 / 0 / 47 | 1.49 |
| Global | 452.902 | 40 / 0 / 60 | -0.39 |
| Static | 489.628 | 31 / 0 / 69 | -8.54 |

(c) LR=0.0003, LE=2, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 457.523 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 455.566 | 55 / 5 / 40 | 0.43 |
| Best Improving | 455.604 | 55 / 5 / 40 | 0.42 |
| Greedy Oracle | 455.852 | 61 / 39 / 0 | 0.37 |
| Random | 457.311 | 36 / 38 / 26 | 0.05 |
| Local | **449.937** | 56 / 0 / 44 | 1.66 |
| Global | 460.755 | 29 / 0 / 71 | -0.71 |
| Static | 487.878 | 26 / 0 / 74 | -6.63 |

(d) LR=0.0003, LE=2, BS=4

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 438.134 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 437.776 | 45 / 12 / 43 | 0.08 |
| Best Improving | 437.782 | 46 / 11 / 43 | 0.08 |
| Greedy Oracle | **432.011** | 56 / 44 / 0 | 1.40 |
| Random | 438.284 | 31 / 41 / 29 | -0.03 |
| Local | 435.206 | 51 / 0 / 49 | 0.67 |
| Global | 437.190 | 48 / 0 / 52 | 0.22 |
| Static | 489.628 | 32 / 0 / 68 | -11.75 |

(e) LR=0.00015, LE=1, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 443.280 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 441.421 | 48 / 10 / 42 | 0.42 |
| Best Improving | 441.403 | 49 / 8 / 43 | 0.42 |
| Greedy Oracle | 439.539 | 56 / 44 / 0 | 0.84 |
| Random | 443.115 | 33 / 38 / 29 | 0.04 |
| Local | **438.404** | 50 / 0 / 50 | 1.10 |
| Global | 438.993 | 52 / 0 / 48 | 0.97 |
| Static | 487.878 | 27 / 0 / 73 | -10.06 |

(f) LR=0.00015, LE=1, BS=4

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 430.572 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 432.643 | 37 / 21 / 42 | -0.48 |
| Best Improving | 432.546 | 38 / 19 / 43 | -0.46 |
| Greedy Oracle | **420.305** | 52 / 48 / 0 | 2.38 |
| Random | 431.707 | 30 / 40 / 31 | -0.26 |
| Local | 433.825 | 45 / 0 / 55 | -0.76 |
| Global | 437.190 | 40 / 0 / 60 | -1.54 |
| Static | 489.628 | 33 / 0 / 67 | -13.72 |

(g) LR=0.00015, LE=2, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 435.164 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 435.906 | 41 / 17 / 42 | -0.17 |
| Best Improving | 435.879 | 42 / 15 / 43 | -0.16 |
| Greedy Oracle | **429.607** | 50 / 50 / 0 | 1.28 |
| Random | 435.827 | 30 / 38 / 32 | -0.15 |
| Local | 434.451 | 47 / 0 / 53 | 0.16 |
| Global | 438.993 | 39 / 0 / 61 | -0.88 |
| Static | 487.878 | 28 / 0 / 72 | -12.11 |

(h) LR=0.00015, LE=2, BS=4

Table 6.2: Summary for Demand-nonIID. Individual experiments can vary by learning rate (LR), local epochs (LE) and minibatch size (BS).

### 6.2.2 FitRec

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 2.519 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.506 | 47 / 14 / 40 | 0.53 |
| Best Improving | 2.506 | 47 / 12 / 41 | 0.51 |
| Greedy Oracle | **2.486** | 56 / 44 / 0 | 1.32 |
| Random | 2.517 | 30 / 43 / 27 | 0.08 |
| Local | 2.508 | 52 / 0 / 48 | 0.45 |
| Global | <u>2.488</u> | 60 / 0 / 40 | 1.25 |
| Static | 3.034 | 16 / 0 / 84 | -20.44 |

(a) Learning Rate=0.00003, Local Epochs=1, Batch Size=4

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | <u>2.472</u> | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.501 | 23 / 49 / 28 | -1.16 |
| Best Improving | 2.502 | 24 / 47 / 29 | -1.22 |
| Greedy Oracle | **2.393** | 41 / 59 / 0 | 3.21 |
| Random | 2.530 | 23 / 40 / 37 | -2.35 |
| Local | 2.557 | 42 / 0 / 58 | -3.43 |
| Global | 2.484 | 48 / 0 / 52 | -0.48 |
| Static | 3.035 | 21 / 0 / 79 | -22.78 |

(b) Learning Rate=<u>0.00015</u>, Local Epochs=1, Batch Size=<u>2</u>

Table 6.3: Summary of FitRec-HR-IID. Results for the experimental configurations with the (a) worst and (b) best synchronized model by mean error across nodes are shown.

Similarly to the `Demand` forecast, the model for heart rate prediction of the `FitRec` dataset is not able to improve upon the synchronized model using the model branching approach. Aside from the oracle heuristic, most heuristics produce worse results in the majority of conditions. While the global model does again not differ significantly from the synchronized model, the local model performs worse in many of the IID as well as non-IID configurations. Summaries for `FitRec HR` are shown in Tables 6.3 and 6.4.

The forecast of derived speeds does however profit from branching to various degrees in all cases. Both global and local models outperform the synchronized. Although the local models beat the synchronized models on more iterations, the global models often come out ahead in terms of total error. The results for these runs can be found in Tables 6.5 and 6.6.

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 2.488 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.470 | 46 / 12 / 43 | 0.72 |
| Best Improving | 2.471 | 47 / 10 / 43 | 0.71 |
| Greedy Oracle | **2.439** | 56 / 44 / 0 | 2.00 |
| Random | 2.486 | 32 / 40 / 28 | 0.09 |
| Local | 2.471 | 51 / 0 / 49 | 0.72 |
| Global | 2.455 | 55 / 0 / 45 | 1.35 |
| Static | 2.987 | 21 / 0 / 79 | -20.02 |

(a) LR=0.00003, LE=1, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 2.524 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.484 | 47 / 13 / 40 | 1.57 |
| Best Improving | 2.485 | 47 / 12 / 41 | 1.56 |
| Greedy Oracle | 2.472 | 55 / 45 / 0 | 2.08 |
| Random | 2.521 | 31 / 42 / 27 | 0.13 |
| Local | 2.489 | 53 / 0 / 47 | 1.40 |
| Global | **2.468** | 60 / 0 / 40 | 2.21 |
| Static | 2.985 | 16 / 0 / 84 | -18.27 |

(b) LR=0.00003, LE=1, BS=4

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 2.463 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.465 | 40 / 17 / 43 | -0.08 |
| Best Improving | 2.466 | 41 / 15 / 44 | -0.14 |
| Greedy Oracle | **2.404** | 51 / 49 / 0 | 2.40 |
| Random | 2.466 | 30 / 39 / 31 | -0.12 |
| Local | 2.468 | 48 / 0 / 52 | -0.20 |
| Global | 2.455 | 52 / 0 / 48 | 0.31 |
| Static | 2.987 | 21 / 0 / 79 | -21.27 |

(c) LR=0.00003, LE=2, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 2.482 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.468 | 39 / 22 / 39 | 0.59 |
| Best Improving | 2.468 | 40 / 19 / 41 | 0.56 |
| Greedy Oracle | **2.431** | 50 / 50 / 0 | 2.05 |
| Random | 2.484 | 29 / 39 / 31 | -0.05 |
| Local | 2.471 | 49 / 0 / 51 | 0.45 |
| Global | 2.468 | 55 / 0 / 45 | 0.57 |
| Static | 2.985 | 17 / 0 / 83 | -20.26 |

(d) LR=0.00003, LE=2, BS=4

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 2.462 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.489 | 23 / 50 / 27 | -1.12 |
| Best Improving | 2.491 | 24 / 48 / 28 | -1.18 |
| Greedy Oracle | **2.379** | 41 / 59 / 0 | 3.37 |
| Random | 2.518 | 23 / 40 / 37 | -2.28 |
| Local | 2.542 | 42 / 0 / 58 | -3.25 |
| Global | 2.468 | 49 / 0 / 51 | -0.24 |
| Static | 2.987 | 22 / 0 / 78 | -21.32 |

(e) LR=0.00015, LE=1, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 2.470 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.480 | 18 / 62 / 20 | -0.44 |
| Best Improving | 2.481 | 18 / 60 / 21 | -0.47 |
| Greedy Oracle | **2.414** | 36 / 64 / 0 | 2.25 |
| Random | 2.525 | 20 / 41 / 39 | -2.24 |
| Local | 2.537 | 41 / 0 / 59 | -2.74 |
| Global | 2.451 | 54 / 0 / 46 | 0.75 |
| Static | 2.985 | 17 / 0 / 83 | -20.88 |

(f) LR=0.00015, LE=1, BS=4

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 2.472 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.494 | 30 / 40 / 31 | -0.90 |
| Best Improving | 2.496 | 30 / 38 / 31 | -0.98 |
| Greedy Oracle | **2.375** | 48 / 52 / 0 | 3.92 |
| Random | 2.543 | 25 / 39 / 36 | -2.89 |
| Local | 2.592 | 41 / 0 / 59 | -4.87 |
| Global | 2.468 | 51 / 0 / 49 | 0.16 |
| Static | 2.987 | 22 / 0 / 78 | -20.83 |

(g) LR=0.00015, LE=2, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 2.464 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.466 | 32 / 40 / 29 | -0.08 |
| Best Improving | 2.467 | 32 / 38 / 30 | -0.12 |
| Greedy Oracle | **2.405** | 49 / 51 / 0 | 2.40 |
| Random | 2.512 | 26 / 37 / 37 | -1.93 |
| Local | 2.552 | 39 / 0 / 61 | -3.56 |
| Global | 2.451 | 52 / 0 / 48 | 0.54 |
| Static | 2.985 | 17 / 0 / 83 | -21.14 |

(h) LR=0.00015, LE=2, BS=4

Table 6.4: Summary for FitRec-HR-nonIID. Individual experiments can vary by learning rate (LR), local epochs (LE) and minibatch size (BS).

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 3.977 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 3.934 | 57 / 4 / 39 | 1.09 |
| Best Improving | 3.933 | 57 / 3 / 40 | 1.11 |
| Greedy Oracle | <u>3.929</u> | 60 / 40 / 0 | 1.22 |
| Random | 3.973 | 34 / 39 / 27 | 0.10 |
| Local | 3.942 | 55 / 0 / 45 | 0.89 |
| Global | **3.876** | 51 / 0 / 49 | 2.56 |
| Static | 4.295 | 15 / 0 / 85 | -7.99 |

(a) Learning Rate=0.00005, Local Epochs=1, Batch Size=4

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 3.943 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 3.828 | 44 / 15 / 41 | 2.91 |
| Best Improving | 3.828 | 44 / 14 / 42 | 2.92 |
| Greedy Oracle | <u>3.774</u> | 51 / 49 / 0 | 4.29 |
| Random | 3.931 | 30 / 40 / 31 | 0.31 |
| Local | 3.855 | 50 / 0 / 50 | 2.24 |
| Global | **3.725** | 48 / 0 / 52 | 5.53 |
| Static | 4.291 | 20 / 0 / 80 | -8.81 |

(b) Learning Rate=<u>0.00025</u>, Local Epochs=1, Batch Size=<u>2</u>

Table 6.5: Summary of FitRec-DS-IID. Results for the experimental configurations with the (a) worst and (b) best synchronized model by mean error across nodes are shown.

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 3.104 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 3.031 | 57 / 3 / 39 | 2.38 |
| Best Improving | 3.030 | 58 / 3 / 39 | 2.39 |
| Greedy Oracle | 3.030 | 62 / 38 / 0 | 2.40 |
| Random | 3.099 | 34 / 40 / 25 | 0.18 |
| Local | 3.021 | 57 / 0 / 43 | 2.67 |
| Global | **3.014** | 49 / 0 / 51 | 2.91 |
| Static | 3.442 | 21 / 0 / 79 | -10.89 |

(a) LR=0.00005, LE=1, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 3.119 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 3.065 | 60 / 4 / 36 | 1.73 |
| Best Improving | 3.065 | 61 / 4 / 36 | 1.74 |
| Greedy Oracle | 3.069 | 63 / 37 / 0 | 1.63 |
| Random | 3.115 | 34 / 41 / 26 | 0.14 |
| Local | 3.063 | 58 / 0 / 42 | 1.80 |
| Global | **3.054** | 50 / 0 / 50 | 2.10 |
| Static | 3.445 | 16 / 0 / 84 | -10.45 |

(b) LR=0.00005, LE=1, BS=4

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 3.093 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.995 | 56 / 3 / 41 | 3.20 |
| Best Improving | 2.995 | 56 / 3 / 41 | 3.18 |
| Greedy Oracle | **2.983** | 61 / 39 / 0 | 3.57 |
| Random | 3.083 | 33 / 40 / 26 | 0.35 |
| Local | 2.985 | 57 / 0 / 43 | 3.49 |
| Global | 3.014 | 47 / 0 / 53 | 2.57 |
| Static | 3.442 | 22 / 0 / 78 | -11.28 |

(c) LR=0.00005, LE=2, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 3.107 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 3.021 | 59 / 3 / 38 | 2.77 |
| Best Improving | 3.021 | 59 / 3 / 39 | 2.77 |
| Greedy Oracle | **3.019** | 62 / 38 / 0 | 2.83 |
| Random | 3.098 | 34 / 41 / 25 | 0.28 |
| Local | 3.026 | 59 / 0 / 41 | 2.61 |
| Global | 3.054 | 45 / 0 / 55 | 1.71 |
| Static | 3.445 | 17 / 0 / 83 | -10.89 |

(d) LR=0.00005, LE=2, BS=4

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 3.077 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.973 | 45 / 18 / 38 | 3.37 |
| Best Improving | 2.974 | 45 / 16 / 39 | 3.35 |
| Greedy Oracle | **2.922** | 52 / 48 / 0 | 5.03 |
| Random | 3.073 | 29 / 41 / 31 | 0.14 |
| Local | 2.964 | 54 / 0 / 46 | 3.66 |
| Global | 2.945 | 48 / 0 / 52 | 4.29 |
| Static | 3.442 | 23 / 0 / 77 | -11.87 |

(e) LR=0.00025, LE=1, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 3.096 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.994 | 46 / 18 / 35 | 3.32 |
| Best Improving | 2.993 | 46 / 18 / 36 | 3.32 |
| Greedy Oracle | **2.965** | 52 / 48 / 0 | 4.23 |
| Random | 3.093 | 29 / 40 / 32 | 0.11 |
| Local | 2.998 | 55 / 0 / 45 | 3.17 |
| Global | 2.976 | 46 / 0 / 54 | 3.89 |
| Static | 3.445 | 18 / 0 / 82 | -11.27 |

(f) LR=0.00025, LE=1, BS=4

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 3.085 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.975 | 45 / 16 / 39 | 3.57 |
| Best Improving | 2.973 | 46 / 14 / 40 | 3.62 |
| Greedy Oracle | **2.869** | 55 / 45 / 0 | 7.00 |
| Random | 3.073 | 29 / 41 / 30 | 0.38 |
| Local | 2.966 | 54 / 0 / 46 | 3.86 |
| Global | 2.945 | 50 / 0 / 50 | 4.53 |
| Static | 3.442 | 24 / 0 / 76 | -11.60 |

(g) LR=0.00025, LE=2, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 3.105 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 2.994 | 48 / 12 / 39 | 3.57 |
| Best Improving | 2.994 | 49 / 11 / 40 | 3.55 |
| Greedy Oracle | **2.931** | 55 / 45 / 0 | 5.61 |
| Random | 3.094 | 29 / 40 / 30 | 0.33 |
| Local | 2.994 | 54 / 0 / 46 | 3.55 |
| Global | 2.976 | 48 / 0 / 52 | 4.14 |
| Static | 3.445 | 19 / 0 / 81 | -10.98 |

(h) LR=0.00025, LE=2, BS=4

Table 6.6: Summary for FitRec-DS-nonIID. Individual experiments can vary by learning rate (LR), local epochs (LE) and minibatch size (BS).

### 6.2.3 Synthetic

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 0.02370 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 0.02350 | 20 / 74 / 6 | 0.85 |
| Best Improving | 0.02350 | 20 / 74 / 6 | 0.83 |
| Greedy Oracle | <u>0.02339</u> | 27 / 73 / 0 | 1.32 |
| Random | 0.02479 | 19 / 40 / 41 | -4.61 |
| Local | 0.02465 | 43 / 0 / 57 | -4.00 |
| Global | **0.01875** | 91 / 0 / 9 | 20.87 |
| Static | 0.02652 | 37 / 0 / 63 | -11.92 |

(a) Learning Rate=0.00005, Local Epochs=1, Batch Size=4

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 0.01982 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 0.01909 | 51 / 31 / 18 | 3.67 |
| Best Improving | 0.01909 | 51 / 31 / 18 | 3.66 |
| Greedy Oracle | <u>0.01895</u> | 62 / 38 / 0 | 4.39 |
| Random | 0.02040 | 29 / 39 / 32 | -2.93 |
| Local | 0.02372 | 29 / 0 / 71 | -19.68 |
| Global | **0.01824** | 72 / 0 / 28 | 7.96 |
| Static | 0.02651 | 20 / 0 / 80 | -33.73 |

(b) Learning Rate=0.00005, Local Epochs=<u>2</u>, Batch Size=<u>2</u>

Table 6.7: Summary of Synthetic-IID. Results for the experimental configurations with the (a) worst and (b) best synchronized model by mean error across nodes are shown.

Experiments on the `Synthetic` dataset yielded the most promising results for the proposed model branching approach.

In the IID scenario, as summarized in Table 6.7, the more stable branching heuristics *Best Last 3* and *Best Improving* show good performance and outperform the purely local models in all cases. The global models show significant improvement over the synchronized baseline as well as the branched models, although there are configurations in which the branched models produce a lower total error despite making fewer decisions that outperform the synchronized model. This suggests that the branching strategies are able to make significant gains when they do decide to branch. The synchronized model improves, when compared to the static model, with more local training epochs but loses some of this gain with a bigger batch size. The branching strategies in turn improve on

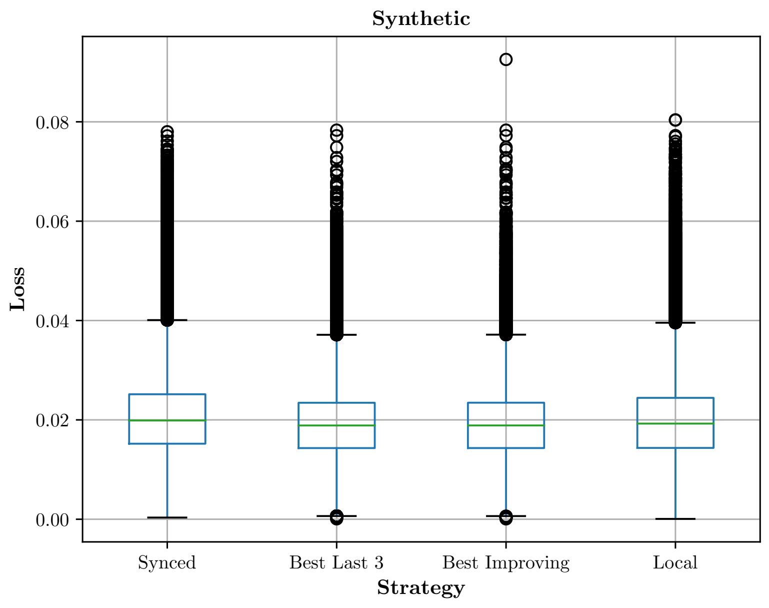


Figure 6.1: Synthetic: Box plot comparison of all the iteration errors for the non-IID scenarios.

the performance of the synchronized model more greatly when using the bigger batch size.

The summary of results for the non-IID scenario can be found in Table 6.8. By utilizing the branching strategies, the nodes are able to markedly improve their local inference accuracies when compared to the synchronized model, achieving the biggest improvement found in this thesis. The models produced by branching show strong performance regardless of how the local models fare against the synchronized models. Their performance is comparable to the models resulting from the oracle strategy. The global models also display considerable improvements over baseline, sometimes being matched or even outperformed by the branched models.

Configurations with a well performing local model also show an improvement over the synchronized baseline when using the *Random* branching heuristic. When the local model is not significantly better than the baseline, the models from this control group are worse than baseline unlike the models from the more stable strategies.

Figure 6.1 shows that the branching heuristic *Best Last 3* improves on both median error as well as variability.

To give an example of a direct, iteration-based comparison between the baseline and *FedAvg-MB* using the *Best Last 3* strategy, Figure 6.2 plots the relative improvement when using this strategy for the two median nodes in terms of average error for the experiment configuration with the most accurate synchronized model (Table 6.8g). The spikes in improvement are caused by the added modifiers creating temporary dataset shift.

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 0.02144 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 0.01872 | 57 / 17 / 26 | 12.71 |
| Best Improving | 0.01868 | 58 / 16 / 27 | 12.85 |
| Greedy Oracle | 0.01904 | 60 / 40 / 0 | 11.20 |
| Random | 0.02111 | 35 / 40 / 25 | 1.53 |
| Local | **0.01785** | 70 / 0 / 30 | 16.73 |
| Global | 0.01867 | 75 / 0 / 25 | 12.90 |
| Static | 0.02601 | 28 / 0 / 72 | -21.33 |

(a) LR=0.00001, LE=1, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 0.02223 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 0.01861 | 66 / 16 / 17 | 16.27 |
| Best Improving | 0.01863 | 67 / 16 / 18 | 16.20 |
| Greedy Oracle | 0.01934 | 67 / 33 / 0 | 13.02 |
| Random | 0.02166 | 38 / 40 / 21 | 2.59 |
| Local | **0.01796** | 78 / 0 / 22 | 19.22 |
| Global | 0.01934 | 78 / 0 / 22 | 13.00 |
| Static | 0.02604 | 25 / 0 / 75 | -17.13 |

(b) LR=0.00001, LE=1, BS=4

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 0.02045 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 0.01830 | 48 / 28 / 24 | 10.51 |
| Best Improving | 0.01834 | 48 / 26 / 25 | 10.34 |
| Greedy Oracle | 0.01804 | 56 / 44 / 0 | 11.78 |
| Random | 0.02035 | 32 / 40 / 29 | 0.48 |
| Local | **0.01758** | 65 / 0 / 35 | 14.01 |
| Global | 0.01867 | 68 / 0 / 32 | 8.69 |
| Static | 0.02602 | 27 / 0 / 73 | -27.23 |

(c) LR=0.00001, LE=2, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 0.02112 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 0.01785 | 64 / 18 / 18 | 15.49 |
| Best Improving | 0.01783 | 64 / 17 / 19 | 15.58 |
| Greedy Oracle | 0.01820 | 68 / 32 / 0 | 13.82 |
| Random | 0.02051 | 37 / 40 / 22 | 2.88 |
| Local | **0.01719** | 75 / 0 / 25 | 18.60 |
| Global | 0.01934 | 74 / 0 / 26 | 8.42 |
| Static | 0.02604 | 23 / 0 / 77 | -23.31 |

(d) LR=0.00001, LE=2, BS=4

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 0.02035 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 0.01993 | 13 / 80 / 7 | 2.05 |
| Best Improving | 0.01996 | 13 / 80 / 8 | 1.94 |
| Greedy Oracle | 0.01894 | 26 / 74 / 0 | 6.96 |
| Random | 0.02355 | 17 / 40 / 43 | -15.73 |
| Local | 0.02248 | 43 / 0 / 57 | -10.47 |
| Global | **0.01642** | 75 / 0 / 25 | 19.31 |
| Static | 0.02601 | 32 / 0 / 68 | -27.82 |

(e) LR=0.00005, LE=1, BS=2

| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 0.02186 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 0.02081 | 18 / 77 / 5 | 4.82 |
| Best Improving | 0.02081 | 18 / 77 / 5 | 4.80 |
| Greedy Oracle | 0.02058 | 26 / 74 / 0 | 5.86 |
| Random | 0.02357 | 18 / 40 / 42 | -7.79 |
| Local | 0.02193 | 47 / 0 / 53 | -0.32 |
| Global | **0.01701** | 84 / 0 / 16 | 22.20 |
| Static | 0.02604 | 34 / 0 / 66 | -19.10 |

(f) LR=0.00005, LE=1, BS=4

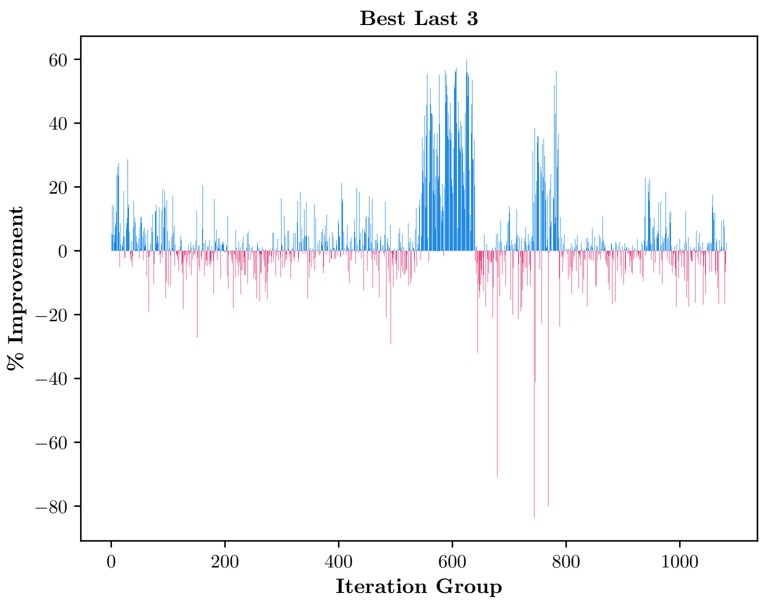
| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 0.01957 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 0.01782 | 50 / 27 / 23 | 8.94 |
| Best Improving | 0.01783 | 50 / 27 / 24 | 8.89 |
| Greedy Oracle | 0.01730 | 61 / 39 / 0 | 11.61 |
| Random | 0.02050 | 28 / 40 / 32 | -4.77 |
| Local | 0.02211 | 43 / 0 / 57 | -12.99 |
| Global | **0.01643** | 74 / 0 / 26 | 16.06 |
| Static | 0.02602 | 33 / 0 / 67 | -32.96 |

(g) LR=0.00005, LE=2, BS=2

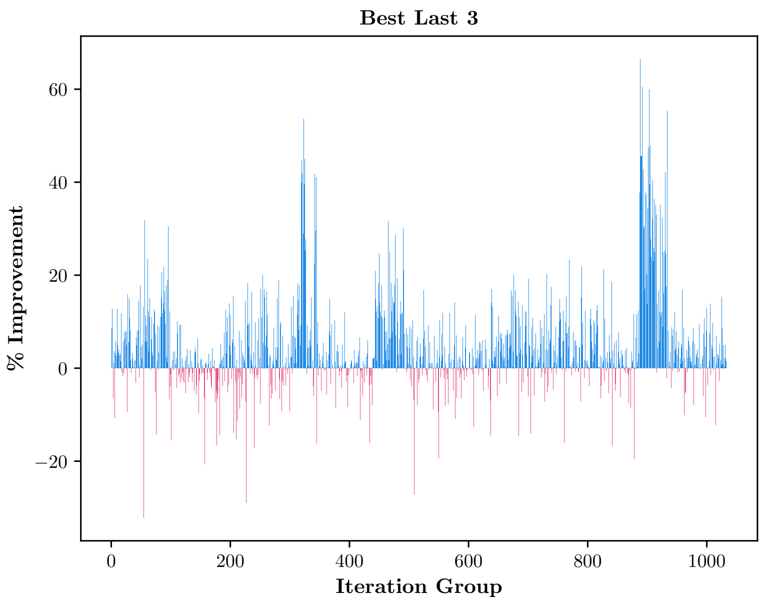
| Heuristic | MAE ↓ | % Be/Eq/Wo | % Improv. |
|---|---|---|---|
| Synced | 0.02005 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 0.01776 | 64 / 22 / 14 | 11.41 |
| Best Improving | 0.01777 | 64 / 22 / 14 | 11.36 |
| Greedy Oracle | 0.01774 | 71 / 29 / 0 | 11.49 |
| Random | 0.02037 | 30 / 40 / 30 | -1.61 |
| Local | 0.02116 | 42 / 0 / 58 | -5.56 |
| Global | **0.01702** | 81 / 0 / 19 | 15.11 |
| Static | 0.02604 | 27 / 0 / 73 | -29.89 |

(h) LR=0.00005, LE=2, BS=4

Table 6.8: Summary for Synthetic-nonIID. Individual experiments can vary by learning rate (LR), local epochs (LE) and minibatch size (BS).



(a) Node 5 - Node with lower median improvement



(b) Node 2 - Node with higher median improvement

Figure 6.2: Synthetic: Iteration-wise relative performances of Best Last 3 in the configuration that produces the best performing synchronized model (LR=0.00005, LE=2, BS=2). Improvements are averaged in groups of four consecutive iterations for better readability.

### 6.2.4 Turnstile

With the exception of the oracle heuristic, the branching strategies result in slightly decreased performance, on average, for both the IID and non-IID setting in the `Turnstile` dataset, as shown in Tables 6.9 and 6.10. This is despite the local models faring noticeably better than baseline in the non-IID scenario. The global models also track very closely to the synchronized models' accuracy.

Further runs with the learning rates $lr \cdot 0.05$ and $lr \cdot 0.01$ yield no improvement in any of the mean errors, suggesting this to be the limit for the network architecture and model.

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 44.388 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 44.193 | 50 / 15 / 35 | 0.44 |
| Best Improving | 44.176 | 51 / 14 / 36 | 0.48 |
| Greedy Oracle | 44.150 | 58 / 42 / 0 | 0.54 |
| Random | 44.366 | 33 / 39 / 28 | 0.05 |
| Local | **43.169** | 60 / 0 / 40 | 2.75 |
| Global | <u>43.909</u> | 57 / 0 / 43 | 1.08 |
| Static | 49.487 | 20 / 0 / 80 | -11.49 |

(a) Learning Rate=0.00001, Local Epochs=2, Batch Size=2

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 42.753 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 43.049 | 41 / 18 / 41 | -0.69 |
| Best Improving | 43.054 | 42 / 16 / 41 | -0.70 |
| Greedy Oracle | **41.789** | 50 / 50 / 0 | 2.25 |
| Random | 43.065 | 26 / 41 / 34 | -0.73 |
| Local | <u>42.675</u> | 46 / 0 / 54 | 0.18 |
| Global | 43.074 | 44 / 0 / 56 | -0.75 |
| Static | 49.484 | 22 / 0 / 78 | -15.74 |

(b) Learning Rate=<u>0.00005</u>, Local Epochs=2, Batch Size=2

Table 6.9: Summary of Turnstile-IID. Results for the experimental configurations with the (a) worst and (b) best synchronized model by mean error across nodes are shown.

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 44.292 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 44.144 | 51 / 11 / 38 | 0.34 |
| Best Improving | 44.146 | 51 / 10 / 39 | 0.33 |
| Greedy Oracle | 43.971 | 56 / 44 / 0 | 0.72 |
| Random | 44.280 | 31 / 40 / 29 | 0.03 |
| Local | **42.130** | 59 / 0 / 41 | 4.88 |
| Global | 43.666 | 52 / 0 / 48 | 1.41 |
| Static | 49.503 | 25 / 0 / 75 | -11.77 |

(a) LR=0.00001, LE=1, BS=2

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 44.468 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 44.218 | 52 / 12 / 36 | 0.56 |
| Best Improving | 44.222 | 53 / 11 / 36 | 0.55 |
| Greedy Oracle | 44.157 | 61 / 39 / 0 | 0.70 |
| Random | 44.437 | 33 / 41 / 26 | 0.07 |
| Local | **42.383** | 63 / 0 / 37 | 4.69 |
| Global | 43.940 | 55 / 0 / 45 | 1.19 |
| Static | 49.512 | 20 / 0 / 80 | -11.34 |

(b) LR=0.00001, LE=1, BS=4

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 43.942 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 43.813 | 50 / 10 / 41 | 0.29 |
| Best Improving | 43.815 | 50 / 8 / 41 | 0.29 |
| Greedy Oracle | 43.434 | 56 / 44 / 0 | 1.16 |
| Random | 43.935 | 30 / 41 / 29 | 0.02 |
| Local | **41.599** | 58 / 0 / 42 | 5.33 |
| Global | 43.666 | 45 / 0 / 54 | 0.63 |
| Static | 49.503 | 24 / 0 / 76 | -12.65 |

(c) LR=0.00001, LE=2, BS=2

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 44.122 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 43.940 | 50 / 10 / 39 | 0.41 |
| Best Improving | 43.931 | 51 / 9 / 40 | 0.43 |
| Greedy Oracle | 43.664 | 60 / 40 / 0 | 1.04 |
| Random | 44.079 | 33 / 39 / 28 | 0.10 |
| Local | **41.774** | 63 / 0 / 37 | 5.32 |
| Global | 43.940 | 43 / 0 / 57 | 0.41 |
| Static | 49.512 | 22 / 0 / 78 | -12.22 |

(d) LR=0.00001, LE=2, BS=4

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 43.641 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 43.708 | 39 / 24 / 37 | -0.15 |
| Best Improving | 43.711 | 39 / 22 / 38 | -0.16 |
| Greedy Oracle | 42.706 | 47 / 53 / 0 | 2.14 |
| Random | 43.806 | 27 / 40 / 33 | -0.38 |
| Local | **41.610** | 54 / 0 / 46 | 4.65 |
| Global | 43.100 | 52 / 0 / 48 | 1.24 |
| Static | 49.503 | 24 / 0 / 76 | -13.43 |

(e) LR=0.00005, LE=1, BS=2

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 43.884 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 43.920 | 37 / 27 / 36 | -0.08 |
| Best Improving | 43.936 | 37 / 26 / 37 | -0.12 |
| Greedy Oracle | 43.115 | 47 / 53 / 0 | 1.75 |
| Random | 43.987 | 27 / 40 / 34 | -0.24 |
| Local | **41.696** | 58 / 0 / 42 | 4.99 |
| Global | 42.891 | 61 / 0 / 39 | 2.26 |
| Static | 49.512 | 22 / 0 / 78 | -12.83 |

(f) LR=0.00005, LE=1, BS=4

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 43.186 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 43.350 | 43 / 16 / 40 | -0.38 |
| Best Improving | 43.367 | 44 / 15 / 42 | -0.42 |
| Greedy Oracle | 41.802 | 52 / 48 / 0 | 3.20 |
| Random | 43.456 | 28 / 39 / 33 | -0.63 |
| Local | **41.796** | 49 / 0 / 51 | 3.22 |
| Global | 43.100 | 47 / 0 / 53 | 0.20 |
| Static | 49.503 | 23 / 0 / 77 | -14.63 |

(g) LR=0.00005, LE=2, BS=2

| **Heuristic** | **MAE ↓** | **% Be/Eq/Wo** | **% Improv.** |
|---|---|---|---|
| Synced | 43.517 | 0 / 100 / 0 | 0.00 |
| Best Last 3 | 43.619 | 43 / 18 / 39 | -0.23 |
| Best Improving | 43.640 | 44 / 16 / 40 | -0.28 |
| Greedy Oracle | 42.373 | 53 / 47 / 0 | 2.63 |
| Random | 43.679 | 27 / 40 / 33 | -0.37 |
| Local | **41.626** | 53 / 0 / 47 | 4.34 |
| Global | 42.891 | 53 / 0 / 47 | 1.44 |
| Static | 49.512 | 22 / 0 / 78 | -13.78 |

(h) LR=0.00005, LE=2, BS=4

Table 6.10: Summary for Turnstile-nonIID. Individual experiments can vary by learning rate (LR), local epochs (LE) and minibatch size (BS).

## 6.3 Discussion

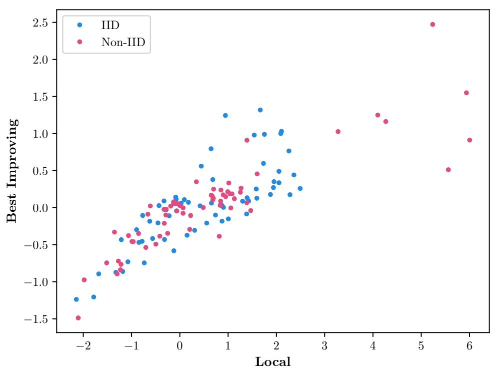
While the model branching approach has only shown a small negative impact on the forecast accuracy in those cases where it performs worse than the baseline, the question of what criteria must be fulfilled for the method to work well is important. To this end, the correlation of local and branched as well as global and branched accuracy is explored. Intuitively, the potential of branching should be limited when the purely local model shows no improvement over the synchronized model.

Figure 6.3 plots the improvement over the synchronized baseline resulting from using the *Best Improving* branching strategy against the respective local models for all datasets and configurations. Figure 6.4 shows the same for the global models. While there exists a weak correlation in some cases between the improvement rate of the local model and the branched model, no such correlation exists between the global and branched models. As noted by Yu et al.[18], nodes where a purely local model outperforms the synchronized model require an incentive to join the federation. Model branching can provide the needed incentive for these nodes by presenting a middle ground.

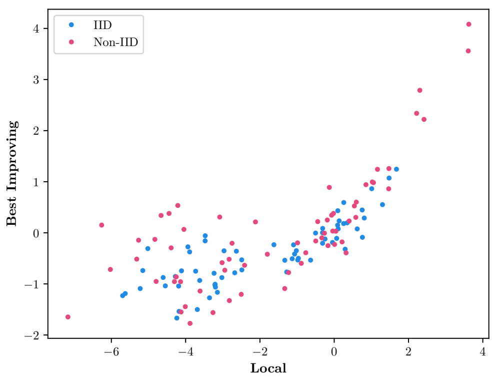
As previously mentioned, forecasts on univariate time series are limited in their capabilities and are generally only able to extract a trend component or seasonal pattern[45]. In the experiments, model accuracy for the real-world datasets does not improve or worsen much in comparison to the synchronized model, which can be explained by the relative complexity of these datasets that pushes the used model network architecture to its limits. The `Synthetic` dataset on the other hand has clear, defining seasonal patterns that create easily distinguishable classes of data. The modifiers included in this dataset further create drastic, immediate dataset shift that favor branching. The approach is thus likely to yield benefits in real-world deployments and settings where the data can quickly evolve in a way that does not resemble training data. Possible causes for this include newly emerging technologies or changing user behavior. On the technical side, the used model must be capable of overfitting to local data partitions in order to enable the branching method, as this method improves performance by managing degrees of overfitting.

While the approach is shown to yield significant improvements on the `Synthetic` dataset as well as the real-world FitRec speed prediction task, more experiments are required on other real-world datasets to test applicability on different types of tasks and validate the approach. Image recognition and classification tasks such as digit recognition on the popular MNIST dataset are common reference tasks in the related research and should allow for higher control over the distribution of classes of data.

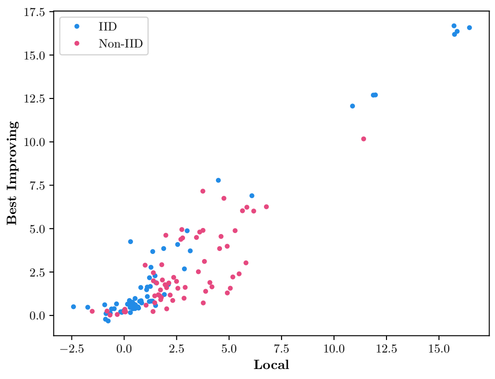
This thesis does not consider the computational cost that is incurred on all nodes by continuing a local branch in addition to participating in the federated optimization process. The higher performing branching heuristics also rely on extraneous computation to make decisions. These costs should be evaluated against both offline and online model personalization methods, whose performance impacts are also not compared against the proposed model branching approach in this work since the goal was to answer whether this approach is capable of improving on a synchronized model.



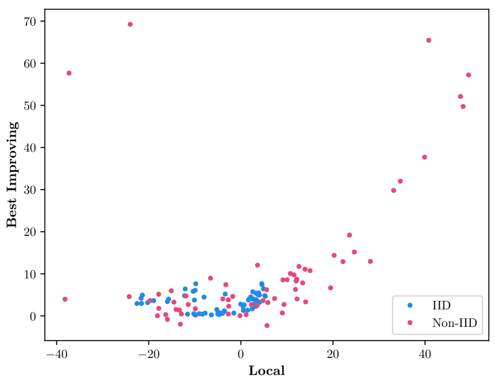
(a) Demand



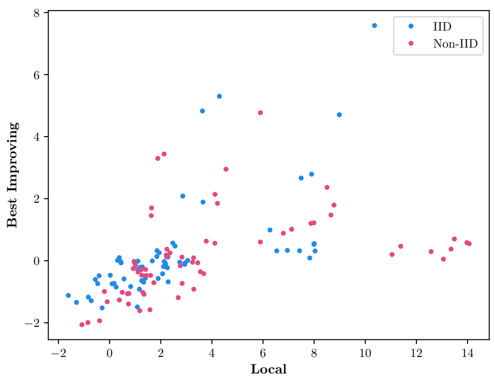
(b) FitRec HR



(c) FitRec DS

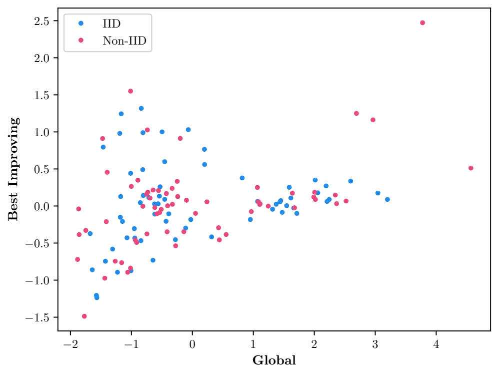


(d) Synthetic

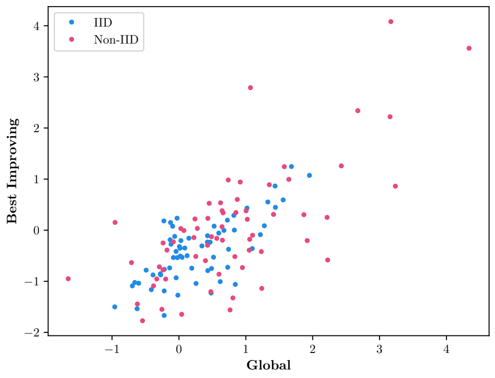


(e) Turnstile

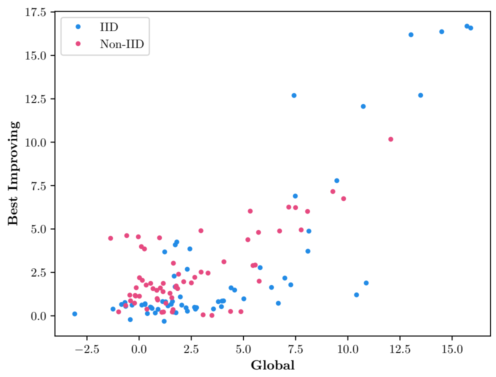
Figure 6.3: Scatter plots comparing the improvements of Best Improving and Local.



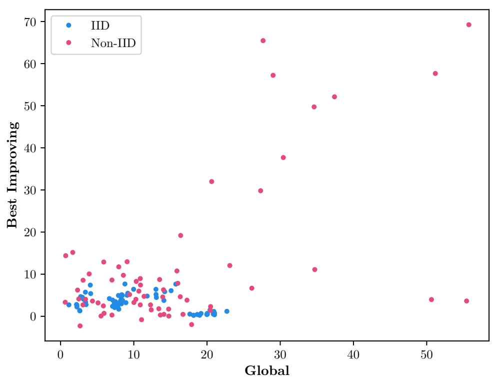
(a) Demand



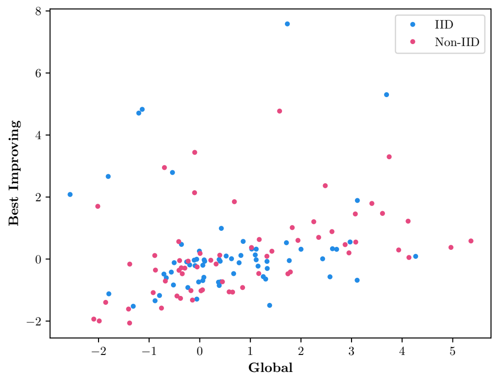
(b) FitRec HR



(c) FitRec DS



(d) Synthetic



(e) Turnstile

Figure 6.4: Scatter plots comparing the improvements of Best Improving and Global.

CHAPTER 7

# Conclusion and Future Work

This thesis proposes a novel method for model personalization in online distributed and federated machine learning. In the addressed scenario, high statistical heterogeneity and dataset shift are pervasive, which not only calls for some form of personalization, but also enables this approach. During operation, local phenomena can push the model of a node that is participating in the global optimization process in a direction that improves the model inference quality in the following synchronization iterations. Decoupling from the global model for inference and continuing to use this model while it promises better localized performance is termed model branching in this thesis.

Experimental results on several datasets using a simulation setup show that this approach is indeed capable of improving model performance when compared to an adapted version of a popular synchronization scheme used in federated learning. However, significant positive results could only be achieved on the synthetic and one of the real-world datasets. The results further confirm the need for continued online learning in all experimental scenarios. Without incremental updates to the model, the forecast accuracy suffers considerably when facing dataset shift.

In future work, the findings should be further validated using additional machine learning tasks such as image classification before extending the approach to include pooling of model branches. Nodes with similar reasons to branch out from the main line could then work together on a new shared model to reduce overfitting and improve generalization. This would allow nodes to autonomously establish regional or feature-based models without supervision. Additional mechanics for heuristic design should also be investigated, such as increasing pressure or reluctance to merge back into the main line.

# List of Figures

# List of Tables

# List of Algorithms

# Bibliography

[1] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset Shift in Machine Learning.* The MIT Press, 2009.

[2] A. Aral, M. Erol-Kantarci, and I. Brandić, "Staleness Control for Edge Data Analytics," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, pp. 38:1–38:24, June 2020.

[3] A. Aral and I. Brandic, "Consistency of the Fittest: Towards Dynamic Staleness Control for Edge Data Analytics," in *Euro-Par 2018: Parallel Processing Workshops*, pp. 40–52, Springer International Publishing, 2019.

[4] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *arXiv:1602.05629 [cs]*, Feb. 2017.

[5] Y. Deng, M. M. Kamani, and M. Mahdavi, *Adaptive Personalized Federated Learning.* Mar. 2020.

[6] Y. Jiang, J. Konečný, K. Rush, and S. Kannan, "Improving Federated Learning Personalization via Model Agnostic Meta Learning," *arXiv:1909.12488 [cs, stat]*, Sept. 2019.

[7] D. Li and J. Wang, "FedMD: Heterogenous Federated Learning via Model Distillation," *arXiv:1910.03581 [cs, stat]*, Oct. 2019.

[8] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, *Advances and Open Problems in Federated Learning.* 2019.

[9] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated Learning with Non-IID Data," *arXiv:1806.00582 [cs, stat]*, June 2018.

[10] K. Hsieh, A. Phanishayee, O. Mutlu, and P. B. Gibbons, "The Non-IID Data Quagmire of Decentralized Machine Learning," *arXiv:1910.00189 [cs, stat]*, Sept. 2019.

[11] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the Convergence of FedAvg on Non-IID Data," *arXiv:1907.02189 [cs, math, stat]*, June 2020.

[12] S. U. Stich, "Local SGD Converges Fast and Communicates Little," *arXiv:1805.09767 [cs, math]*, May 2019.

[13] H. Yu, S. Yang, and S. Zhu, "Parallel Restarted SGD with Faster Convergence and Less Communication: Demystifying Why Model Averaging Works for Deep Learning," July 2018.

[14] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," *arXiv:1812.06127 [cs, stat]*, Apr. 2020.

[15] C. Xie, O. Koyejo, and I. Gupta, "SLSGD: Secure and Efficient Distributed On-device Machine Learning," in *Machine Learning and Knowledge Discovery in Databases* (U. Brefeld, E. Fromont, A. Hotho, A. Knobbe, M. Maathuis, and C. Robardet, eds.), Lecture Notes in Computer Science, (Cham), pp. 213–228, Springer International Publishing, 2020.

[16] Y. Chen, Y. Nin, M. Slawski, and H. Rangwala, "Asynchronous Online Federated Learning for Edge Devices," *arXiv:1911.02134 [cs]*, May 2020.

[17] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Communication-Efficient On-Device Machine Learning: Federated Distillation and Augmentation under Non-IID Private Data," *arXiv:1811.11479 [cs, stat]*, Nov. 2018.

[18] T. Yu, E. Bagdasaryan, and V. Shmatikov, "Salvaging Federated Learning by Local Adaptation," *arXiv:2002.04758 [cs, stat]*, Feb. 2020.

[19] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," *arXiv:1703.03400 [cs]*, July 2017.

[20] M. Khodak, M.-F. Balcan, and A. Talwalkar, "Adaptive Gradient-Based Meta-Learning Methods," *arXiv:1906.02717 [cs, stat]*, Dec. 2019.

[21] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized Federated Learning: A Meta-Learning Approach," *arXiv:2002.07948 [cs, math, stat]*, June 2020.

[22] P. P. Liang, T. Liu, L. Ziyin, N. B. Allen, R. P. Auerbach, D. Brent, R. Salakhutdinov, and L.-P. Morency, "Think Locally, Act Globally: Federated Learning with Local and Global Representations," *arXiv:2001.01523 [cs, stat]*, July 2020.

[23] J. Wang, M. Kolar, and N. Srerbo, "Distributed Multi-Task Learning," in *Artificial Intelligence and Statistics*, pp. 751–760, May 2016.

[24] I. M. Baytas, M. Yan, A. K. Jain, and J. Zhou, "Asynchronous Multi-task Learning," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 11–20, Dec. 2016.

[25] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated Multi-Task Learning," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 4424–4434, Curran Associates, Inc., 2017.

[26] F. Hanzely and P. Richtárik, "Federated Learning of a Mixture of Global and Local Models," *arXiv:2002.05516 [cs, math, stat]*, Feb. 2020.

[27] Y. Mansour, M. Mohri, J. Ro, and A. T. Suresh, "Three Approaches for Personalization with Applications to Federated Learning," *arXiv:2002.10619 [cs, stat]*, July 2020.

[28] P. Vanhaesebrouck, A. Bellet, and M. Tommasi, "Decentralized Collaborative Learning of Personalized Models over Networks," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (A. Singh and J. Zhu, eds.), vol. 54 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 509–517, PMLR, Apr. 2017.

[29] A. Bellet, R. Guerraoui, M. Taziki, and M. Tommasi, "Personalized and Private Peer-to-Peer Machine Learning," in *International Conference on Artificial Intelligence and Statistics*, pp. 473–481, Mar. 2018.

[30] V. Zantedeschi, A. Bellet, and M. Tommasi, "Fully Decentralized Joint Learning of Personalized Models and Collaboration Graphs," *arXiv:1901.08460 [cs, stat]*, Mar. 2020.

[31] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive Federated Learning in Resource Constrained Edge Computing Systems," *arXiv:1804.05271 [cs, math, stat]*, Feb. 2019.

[32] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015.

[33] M. A. Nielsen, *Neural Networks and Deep Learning*, vol. 2018. Determination Press San Francisco, CA, 2015.

[34] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling Distributed Machine Learning with the Parameter Server," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, (USA), pp. 583–598, USENIX Association, 2014.

[35] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-Aware Distributed Parameter Servers," in *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, (New York, NY, USA), pp. 463–478, Association for Computing Machinery, 2017.

[36] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated Machine Learning: Concept and Applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, Jan. 2019.

[37] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards Federated Learning at Scale: System Design," *arXiv:1902.01046 [cs, stat]*, Mar. 2019.

[38] M. Pagels, "What is Online Machine Learning?." `https://medium.com/value-stream-design/online-machine-learning-515556ff72c5`, Feb. 2020. (accessed 2020-07-13).

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[40] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[41] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference* (S. van der Walt and Jarrod Millman, eds.), pp. 56–61, 2010.

[42] M. Muratori, "Impact of uncoordinated plug-in electric vehicle charging on residential power demand - supplementary data," 2017.

[43] J. Brownlee, *Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery, 2018.

[44] J. Ni, L. Muhlstein, and J. McAuley, "Modeling Heart Rate and Activity Data for Personalized Fitness Recommendation," in *The World Wide Web Conference*, WWW '19, (San Francisco, CA, USA), pp. 1343–1353, Association for Computing Machinery, May 2019.

[45] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. Australia: OTexts, second ed., 2018.

[46] C. Whong, "Taming the MTA's Unruly Turnstile Data." `https://medium.com/qri-io/taming-the-mtas-unruly-turnstile-data-c945f5f96ba0`, May 2020. (accessed 2020-09-07).

[47] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[48] Z. Zhao, W. Chen, X. Wu, P. C. Y. Chen, and J. Liu, "LSTM network: A deep learning approach for short-term traffic forecast," *IET Intelligent Transport Systems*, vol. 11, no. 2, pp. 68–75, 2017.

[49] S. Siami-Namini, N. Tavakoli, and A. S. Namin, "A Comparison of ARIMA and LSTM in Forecasting Time Series," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1394–1401, 2018.

[50] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017.

[51] A. Karpathy, "A Recipe for Training Neural Networks." `https://karpathy.github.io/2019/04/25/recipe/`, Apr. 2019. (accessed 2020-09-09).