Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Erik Dymbe

Decentralized Artificial Intelligence for Image Classification and Rating Prediction

Master's thesis in Informatics
Supervisor: Hai Thanh Nguyen
July 2021

Erik Dymbe

Decentralized Artificial Intelligence for Image Classification and Rating Prediction

Master's thesis in Informatics
Supervisor: Hai Thanh Nguyen
July 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Recommender systems are one of the biggest commercial success stories of artificial intelligence in practice. They are used to generate large amounts of wealth for the organizations that operate them. Netflix has estimated that their recommender system saves the company $1 billion every year, and has disclosed that about 75% of what the customers of Netflix watch is from some sort of recommendation. However, recommender systems are dependent on large datasets to be able to generate accurate recommendations for their users. This incentivizes organizations to collect massive amounts of data on their users. This is not necessarily in the best interest of the user. Data breaches are not uncommon occurrences, and some might not be comfortable with letting these organizations have large amounts of data on them and their behavior.

With the introduction of data protection laws, such as the GDPR in the EU, people's right to have control over their own data has been reaffirmed. This opens up for a new paradigm for Artificial Intelligence where a person's ownership of their own data is emphasized. This paradigm will be referred to as Decentralized Artificial Intelligence, and will describe a pattern where data collection, training, and inference is all performed locally on the device of the user. Using this paradigm, the user has complete control over how their data is used.

This thesis will explore the many challenges associated with Decentralized Artificial Intelligence. Two approaches were considered at first: decentralized ensemble methods, and collaborative learning methods like Federated Learning. Following experiments comparing these methods on the MNIST handwritten digit dataset, collaborative learning is selected as the most promising method.

Two collaborative learning methods, Federated Learning and Gossip Learning, are then compared for several movie-recommendation datasets. By performing multiple analyses of the training process of Gossip Learning, several areas of improvement in the previous work is identified. Multiple strategies to increase performance are developed, which are also applicable to Federated Learning.

Lastly a new paradigm which will be referred to as semi-decentralized learning is introduced. This paradigm has the advantage of offering better performance compared to a fully decentralized technique like Gossip Learning, but it comes at the cost of increasing centralization.

Sammendrag

Anbefalingssystemer er blant de største kommersielle suksesshistoriene innen kunstig intelligens i praksis. Anbefalingssystemer brukes til å skape store inntekter for organisasjonene som tar dem i bruk. Netflix har estimert at deres anbefalingssystem sparer selskapet for 1 milliard dollar hvert år, og har avslørt at omtrent 75% av det kundene deres ser på er anbefalt. Anbefalingssystemer er imidlertid avhengige av store datasett for å kunne gi nøyaktige anbefalinger til brukerne. Dette gir organisasjoner insentiv til å samle enorme mengder data om brukerne sine. Dette er ikke nødvendigvis i brukerens interesse. Databrudd er ikke en uvanlig hendelse, og noen personer er kanskje ikke komfortable med å la disse organisasjonene ha store mengder data om seg selv og sin oppførsel.

Med innføringen av databeskyttelseslover, som GDPR i EU, er folks rett til å ha kontroll over sin egen data blitt styrket. Dette åpner for et nytt paradigme innen kunstig intelligens der en persons eierskap til sin egen data blir vektlagt. Dette paradigmet vil bli referert til som desentralisert kunstig intelligens, og vil beskrive et mønster der datainnsamling og trening og bruk av maskinlæringsmodeller gjøres lokalt på brukerens enhet. Ved å bruke dette paradigmet har brukerene full kontroll over hvordan dataen deres brukes.

Denne oppgaven vil utforske de mange utfordringene knyttet til desentralisert kunstig intelligens. To tilnærminger ble først vurdert: desentraliserte ensemblemetoder og samarbeidende læringsmetoder, som føderert læring. Etter eksperimenter som sammenligner disse metodene på MNIST-datasettet for håndskrevne siffer, blir samarbeidslæring valgt som den mest lovende metoden.

To samarbeidende læringsmetoder, føderert læring og sladderlæring, sammenlignes deretter for flere datasett for filmanbefaling. Ved å utføre flere analyser av treningsprosessen til sladderlæring, identifiseres flere områder som kan forbedres hos tidligere arbeid innen dette området. Flere strategier utvikles for å øke ytelsen. Disse strategiene kan også brukes til å forbedre føderert læring.

Til slutt introduseres et nytt paradigme som vil bli referert til som semi-desentralisert læring. Dette paradigmet har fordelen av å tilby bedre ytelse sammenlignet med en helt desentralisert teknikk som sladderlæring, på kostnad av økt sentralisering.

Acknowledgements

I would like to thank my Master’s project advisor, Hai Thanh Nguyen, for his guidance and feedback. His input has been invaluable to this project.

I would also like to thank Dr. István Hegedűs, for the helpful insights he provided about his research on the topic of Gossip Learning.

Contents

Figures

Tables

Acronyms

AI Artificial Intelligence. 1, 2, 7, 10
ALS Alternating Least Squares. 14, 15
ANN Artificial Neural Network. 5, 6

DzAI Decentralized Artificial Intelligence. 2, 7, 10, 11

FL Federated Learning. xiii, xv, 2–4, 7–9, 15, 16, 23–25, 27, 28, 31, 32, 34, 35, 57, 58, 60, 61, 63

GL Gossip Learning. xiii, xiv, 2–4, 8, 9, 16, 19, 27, 28, 31–35, 37, 40–42, 46–48, 55–58, 60–63

IID Independent and Identically Distributed. 60

ML Machine Learning. 2–9, 17, 59
MLP Multilayer Perceptron. xiii, 5, 17–19
MV Majority Voting. 6, 19–22, 59

OLS Ordinary Least Squares. 14, 15

ReLU Rectified Linear Unit. 5
RMSE Root Mean Square Error. xiv, 15, 28, 32, 33, 36, 41, 55–57, 61

SGD Stochastic Gradient Descent. 14, 15, 17, 62, 63

WMA Weighted Majority Algorithm. xv, 6, 7, 20–23, 25, 59, 62
WMV Weighted Majority Voting. 6, 20

Chapter 1

Introduction

1.1 Motivation

Data has become one of the most valuable resources of the 21st century. It can be used to present solutions to problems that many face today, and can be used to provide valuable insights about topics that are important to society at large. Despite its significance to society, data processing is dominated by a handful of large tech-companies that managed to collect vast amounts of data on their users. Due to the potential that data can open up for businesses, there are growing concerns regarding large tech companies processing their users' data without their consent [1].

Recommender systems are one application of Artificial Intelligence (AI) that can be used to open up a lot of potential for a businesses. A recommender system is a system used to generate recommendations for a user. They are present on just about any e-commerce site and is a key feature for any distributor of digital content, whether that is video, film, or music. Netflix has estimated that their recommender system saves the company $1 billion every year [2], and has disclosed that about 75% of what their customers watch is from some sort of recommendation [3]. A challenge with recommender system is, however, that they require large amounts of data on users and their behavior for them to be able to generate accurate recommendations. This incentivizes organizations to collect massive amounts of data on their users. This is often not in the best interest of the user and it is the user themselves that may suffer from the negative impacts of large data collection. Data breaches are not uncommon occurrences [4], and some companies also share user data with third-parties without the consent of the users [5]. Sharing data with an organization therefore increases the probability that the users data is accessed by an unauthorized party. Some users may also simply be uncomfortable with letting these organizations have large amounts of data on them and their behavior.

With the introduction of data protection laws, such as the GDPR [6] in the EU, people's right to have control over their own data has been reaffirmed. This opens up for a new paradigm for AI, where a person's ownership of their own data is emphasized. This paradigm is referred to as Decentralized Artificial Intelligence (DzAI), and describes a pattern where data collection, training, and inference is all performed locally on the device of the user. This means that no raw user data will ever have to leave the device of the user. Using this paradigm, the user has complete control who has access to their data, and how it is processed. Using the DzAI-paradigm allows for the creation of a decentralized recommender system where the privacy of the users is maintained.

The DzAI-paradigm also offers solutions to other problems of centralized AI. When a company designs a centralized Machine Learning (ML) model, the interests of the user are often secondary. The user rarely has any influence over the design of the model, even when the users' data is used to train it. For example, when Amazon decides to promote their own products on their website [7], there is little the users can do to get recommendations that are less biased towards Amazon's products, even if this might be in the interest of the user. Likewise, video recommendations are not necessarily given on the basis of how much the user will benefit from watching the video, but rather to maintain user engagement on the site [8]. Videos that make the user spend as much time as possible on the platform increases the ad-revenue generated for the company. Decentralization can give the users more leverage when deciding what the ML-model should be trained to do. If the users disagree with the goals of the organization regarding what the ML-model should be trained to do, they can modify how their local model is trained.

1.2 Research Questions

1.2.1 RQ1 - What are the challenges associated with machine learning that do not violate privacy?

The goal of this thesis is to explore the possibilities of privacy-preserving Machine Learning. However, doing so is not trivial. How does one train a machine learning model if one do not have access to the data needed to train it, and how does this lack of access affect the final performance of the model? Chapter 3 will be used for exploring different ways to do machine learning while preserving privacy, which will be used to answer this question.

1.2.2 RQ2 - What are the advantages and disadvantages of a fully decentralized user-to-user recommender system?

Decentralized algorithms can be used to preserve the privacy of users. They could therefore be used with great utility for recommender systems. A decentralized technique like Federated Learning depends on having a central aggregation-server however, and can therefore not be considered fully decentralized. The users are dependent on an organization to maintain the recommender system. Another approach is a fully decentralized user-to-user technique like Gossip Learning. This technique allows for a recommendation system that can operate completely independent of an organization, which offers the user more leverage when deciding how the recommendation system should be designed. The advantages and disadvantages of fully decentralized algorithms should therefore be examined. Chapter 4 will compare Federated Learning to Gossip Learning, while Chapter 5 will explore new ways to improve the performance of Gossip Learning. This will then be used to answer this question.

1.2.3 RQ3 - What are the benefits of semi-decentralized learning?

Centralized approaches offers simpler and more efficient training. Decentralized methods offers privacy, and gives the user more leverage when deciding how a model is trained. Can a semi-decentralized approach offers some of the benefits of both approaches? A semi-decentralized approach can be regarded as a compromise between the centralized and the decentralized approach. It should be examined if there are any benefits to a semi-decentralized approach. Chapter 6 will investigate whether a semi-decentralized approach can achieve similar performance to the centralized approach while keeping some of the benefits of decentralization. This will be used to answer this question.

1.3 Contribution

This thesis explores the many challenges associated with Machine Learning that does not violate privacy. Machine Learning that does not violate privacy is, in this thesis, considered as Machine Learning were no raw user data ever has to leave the device of the user. Several techniques was tested, which can be divided into two larger categories: ensemble methods and collaborative learning methods. The results of these initial experiments highlights the several advantages and disadvantages associated with both decentralized approaches, and provides a starting point for further experiments.

Based on this starting point, collaborative learning techniques was chosen as the preferred approach for a decentralized recommender system. Previous work was built upon to compare two collaborative learning techniques: Federated Learning and Gossip Learning. Federated Learning depends on having a centralized aggregation-server, while GL does not. This thesis examines and presents the advantages and disadvantages of both approaches.

Several analyses of the training process of Gossip Learning was performed to identify areas of improvement and create a deeper understanding of the Gossip Learning algorithm. This lead to techniques that improved upon the results achieved in the original paper. Applying these techniques to Federated Learning also improved the results of this method.

Finally, a new paradigm referred to as semi-decentralized learning was introduced. The advantage of this paradigm is that it can produce better results than a fully decentralized technique like Gossip Learning, although it comes at the cost of increasing centralization.

1.4 Outline of the thesis

Chapter 2 will present the background and the related work for the thesis.

Chapter 3 will be used for exploring different ways to do Machine Learning while preserving privacy. This will be used to answer RQ1.

Chapter 4 will compare Federated Learning to Gossip Learning. This will be used to help answer RQ2.

Chapter 5 will explore new ways to improve the performance of Gossip Learning. This will be used to help answer RQ2.

Chapter 6 will investigate whether a semi-decentralized approach can achieve similar performance to the centralized approach while keeping some of the benefits of decentralization. This will answer RQ3.

Chapter 7 will discuss the results and present ideas for future work.

Chapter 2

Background and related work

2.1 Centralized Machine Learning

Traditionally Machine Learning (ML) is done by a central entity collecting data and then training a model on either a single machine or a cluster of centrally coordinated machines. An example of this is a parameter server with multiple worker nodes [9]. An advantage of centralized ML is its simplicity and efficiency. Having all data available at one location enables a straightforward and efficient training process.

2.1.1 Deep Learning

Deep Learning refers [10] to a family of ML-methods based on Artificial Neural Networks (ANNs) that try to imitate some properties of the neural networks in biological brains.

An ANN consists of several interconnected artificial neurons. An artificial neuron is loosely analogous to the neurons in biological brains. An artificial neuron can have one or more inputs and calculates its output by first taking the weighted sum of its inputs and applies an activation function to this weighted sum.

The neurons are typically organized in layers, where the output signals of artificial neurons in one layer feeds into the input of neurons in the following layer.

Multilayer Perceptron

A Multilayer Perceptron (MLP) [10] is an ANN with an input layer, one or more hidden layers, and an output layer. MLPs also use a non-linear activation function.

ReLU

Rectified Linear Unit (ReLU) [10] is a non-linear activation function frequently used for ANNs. It is defined as:

$$f(x) = \max(0, x) \tag{2.1}$$

where x is the input to a neuron in the ANN.

Softmax

The Softmax-function [10] is an activation function that takes a vector z of K real numbers and outputs K real valued numbers in the interval $(0, 1)$, where all the components sum up to one. It is defined as:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.2}$$

The output vector of the Softmax-function sums up to 1. For classification tasks it is therefore often used as the activation function of the output layer of an ANN such that the outputs can be interpreted as probabilities for each class.

2.1.2 Ensemble Methods

Ensemble methods is the practice of combining the output of multiple ML-models in an attempt to produce better performance than any of the models would be able to produce on their own.

Majority Voting

Majority Voting can be used to combine the outputs of multiple models. For a classification problem one can use Majority Voting to combine the outputs of multiple classification models. Every model votes for the most likely class according to that model. With Majority Voting the class with the most votes is set as the output of the whole ensemble.

Weighted Majority Voting

Weighted Majority Voting (WMV) has similarities to regular Majority Voting, except that with WMV the vote of each model also has a weight. This means that the vote of some models can be more important for deciding the output of the ensemble. WMV is identical to Majority Voting if all weights are equal.

Weighted Majority Algorithm

The Weighted Majority Algorithm (WMA) [11] is a meta-learning algorithm that utilizes WMV to create a compound model out of a pool of models in an online learning context. WMA initializes the weights of all models to the same value. For every round of the algorithm the WMA will first use WMV on the outputs of the models in its pool to generate a compund prediction. If this prediction does not equal the ground truth, every model that did not vote for the ground truth is punished by having their weight reduced by a ratio β, where $0 < \beta < 1$.

It can be shown that the upper bound on the number of mistakes WMA will make for a given sequence of prediction is:

$$O(\log |A| + m) \tag{2.3}$$

where $|A|$ is the number of models in the model pool and m is the least amount of mistakes made by a single model in A [11].

2.2 Decentralized Artificial Intelligence

In this thesis Decentralized Artificial Intelligence (DzAI) will refer to a new paradigm in AI where the data collection, model training, and inference is done at the device where the data was collected. In this paradigm private data will remain private.

2.2.1 Decentralized Ensemble Methods

The ensemble methods described in Section 2.1.2 can be decentralized. Other users can share models that have been trained locally. This way, they do not share their own raw data directly. It is important to note that in some cases, it might be possible to extract information from a model if countermeasures are not taken. Differential privacy is therefore important to protect against this, more about this can be found in Section 2.2.5.

2.2.2 Federated Learning

Federated Learning (FL) [12] is a collaborative learning technique that can be used to train an ML-model without private data being directly shared to a central server. The technique bears some resemblance to the more common parameter server architecture. However, with FL the training is not performed on the server or a server cluster, but rather on the devices that collected the private data, for example a mobile device. This is done by the server sending the device a model to train which the device then tries to improve by using local data. The changes to the model, i.e. the calculated gradients to the model, are then sent back to the server and averaged with the gradients calculated by the other devices also participating in that iteration of the training process. Testing a model can be performed in a similar manner.

FL provides multiple benefits. First of all, not directly sharing raw private data during training is a big improvement to doing so using the traditional centralized approach. It should however be noted that sharing gradients does reveal some information about the data that was used to train it, although there are ways to mitigate this. More about this can be found in Section 2.2.4.

Not having to store private data centrally not only provides some benefit to the user, but it might also be beneficial for an organization as they do not have to worry as much about data breaches and complying with privacy regulations. Organizations can also save data storage space, since all of their users' data will be stored on their device. Lastly, since the models can be trained locally, they can also be used locally. If network speed and usage is a bigger constraint than computational power, one can expect a local model to offer better performance compared to having the model in the cloud. This also saves resources for the organization that would otherwise have to host that model in the cloud.

2.2.3 Gossip Learning

Gossip Learning (GL) [13] is a collaborative learning technique for training an ML-model without private data being shared directly. It shares some similarities to FL. For both collaborative learning techniques the training is performed on the device that collected the data. Private user data therefore does not need to be shared. However, contrary to FL, GL is not dependent on a central server aggregating the gradients. Instead of sharing gradients with a server, a network of devices using GL shares the locally trained models with other devices in the network. When a device receives a model, it merges the received model with its own locally trained model. This process is repeated until a satisfactory model has been trained.

Hegedűs *et al.* introduces two methods of merging models. The first simply discards the local model in favour of the received model. The other produces a new model by taking a weighted average of the parameters of the received model and the local model, where the model being updated the most times is weighted higher. The intuition behind model merging is that in certain linear hypothesis spaces, voting-based prediction is equivalent to a single prediction by the average of the models that participate in the voting [14]. While, generally, this does not hold true in a strict sense, it is often true that averaging models produces a better model that is better than any of the single models that was averaged [15]. For neural networks it is important that the networks have the same initialization of the weights and biases for this to hold true.

GL provides much of the same benefits as FL with the added benefit of requiring no central aggregation server. GL therefore provides much cheaper scalability, and since it does not have a single point of failure, it is arguably also more robust. Sharing a locally trained model might however reveal some information about the data used to train it, but there are methods that improves upon this. More on this in Section 2.2.4.

2.2.4 Secure Aggregation

A vulnerability of FL is the shared gradients. Gradients can reveal information about the private data that was used to calculate the gradients. Zhu *et al.* shows that a close reconstruction of the training data can often be extracted quite easily from gradients [16]. Increasing the amount of private data that was used to calculate the gradient did however make the extraction harder. Increasing the amount of data that went into a gradient therefore increases the difficulty of extracting data from that gradient.

Bonawitz *et al.* developed a protocol for FL that allows the server to securely calculate the average of all of the users gradients without ever seeing the gradients submitted by any individual user [17]. This average gradient can then be applied to the federated model. Since the server and any other malicious agents can only know about the average gradient, and not any individual gradient, it is a lot harder for them to learn any information about any individual user. Increasing the number of participants increases the difficulty of extracting information about any individual user. The protocol is built on top of Shamir's t-out-of-n Secret Sharing [18].

GL has a similar weakness. It is trivial to extract the gradients if one has access to both the original and updated model, but due to GL lacking any centralized coordination, it is harder for an attacker to have access to both, as long as the peer selection algorithm is secure. This still leaves GL vulnerable to multiple agents colluding. If agent A sends a model to agent B who then updates and sends the model to agent C, it is trivial for agent A and C to calculate the gradients calculated by agent B. This is why Danner *et al.* developed a fully distributed mini-batch algorithm that, in a completely decentralized manner, computes a sum gradient for a subset of agents in the network without it being possible to gain any information about the gradients of any individual. The algorithm is built upon the additively homomorphic Paillier cryptosystem [20]. None of these secure aggregation methods are used in the experiments in this paper.

2.2.5 Differential Privacy

For an ML-model to provide any useful information it has to reveal some information about the dataset it was trained on [21]. Differential privacy is a strong, mathematical definition of privacy in the context of statistical and machine learning analysis [22]. It provides a system for sharing information about a dataset while minimizing the information revealed about any individual in the dataset. Generally, an algorithm is differentially private if an observer would not be able to tell if an specific individuals data was used to create the output of the algorithm.

The goal when training ML-models is to create a model that generalizes, but it is not unheard of that a model will memorize some of the training data instead, especially if that training data is rare or unique. An ML-model with a high degree of memorization is not differentially private as it risks leaking information about the memorized training data. Carlini *et al.* shows that it is possible to extract highly sensitive information like social security numbers and credit card numbers from a generative text model trained on the Enron Email Dataset [23]. The authors also developed a methodology for measuring model memorization.

Although differential privacy is important for centralized AI, it is especially important for DzAI as privacy is a big part of its value proposition. Models are also distributed to multiple agents, some of which might be malicious. Since these malicious agents have direct access to the models, there should be limits to how much information they can extract from the models. A DzAI algorithm should therefore utilize strategies for increasing differential privacy. A simple way to mitigate risks related to model memorization is to remove rare or sensitive information from the training data. More sophisticated techniques are described in [24]. None of these will be implemented in this thesis.

2.3 Recommender systems

A recommender system is a system used for recommending items to a user, whether the items are movies, music, articles of clothing or something completely different. Recommender systems has a wide range of commercial applications. Netflix, Spotify, and YouTube uses recommender systems for recommending digital content, while e-commerce companies like Amazon uses them for recommending users items to buy.

These recommender systems are of the traditional centralized variant. These organizations therefore need to collect a massive amount of data about their users to make the recommender systems as accurate as possible. This puts the users at risk of having their data leaked through a data breach. The users might also be uncomfortable with these organizations having extensive knowledge about them and their behaviour.

A centralized recommender system also does not let its users influence how the recommendations are made. When Amazon decides to promote their own products on their website [7], there is little the users can do to get less biased recommendations. If the recommender system is decentralized the users can decide to use a different model.

2.3.1 Content-based filtering

Content-based filtering is an approach for recommender systems where the system recommends items that are similar to items the user has previously liked [25]. Whether or not a user like an item can be derived from implicit activity like clicking on an item, or from explicit activity like rating the item highly.

Content-based filtering is especially useful when there is a full description available for each item, but little available information about each user. Content-based filtering therefore does not suffer from the *cold-start problem*. A recommender system that suffers from the cold-start problem is unable to produce accurate recommendation until it has collected enough information about its users activity.

A problem with content-based filtering is that only recommending items similar to items the user has previously interacted favorably with might in some cases not be ideal. A user that just bought a fridge is most likely not going to buy another fridge anytime soon. Similarly, a user might grow tired of recommendations for movies similar to movies the user just watched.

As content-based filtering is dependent on detailed descriptions for each item, it is not ideal for DzAI. The user has direct access to its own data, but users will typically not have access to detailed descriptions for each item without relying on a centralized source for that information.

2.3.2 Collaborative filtering

Collaborative filtering is another approach to recommender systems where the system recommends items that other similar users has liked [25]. This way the recommender system is able to recommend a more diverse selection of items that are not necessarily similar to other items the user has liked.

Collaborative filtering works well even if there is minimal information available for each item. Collaborative filtering is however more dependent on the user-item interaction history than content-based filtering is. Collaborative filtering therefore suffers more from the aforementioned cold-start problem than content-based filtering.

Since collaborative filtering can be effective when having nothing else than a history of user-item interactions, which can be easily collected by the user, it is more suited for decentralized learning than content-based filtering.

Memory-based

Memory-based collaborative filtering algorithms operate over the entire user database to make predictions [26]. A recommendation for an item could for example be based on what users with a similar rating history rated that item.

Memory-based collaborative filtering algorithms are unsuited for decentralized privacy-preserving recommender systems. This is because when using memory-based collaborative filtering algorithms a user has to have access to similar users' data to be able to make any predictions, which compromises the privacy of the participating users.

Model-based

Model-based collaborative filtering algorithms use the user database to estimate or learn a model which is then used for recommendation [26]. User data is needed to construct the model, but once the model is constructed one does not need user data to create recommendations. Model-based collaborative filtering algorithms are therefore better suited for privacy-preserving decentralized recommender systems.

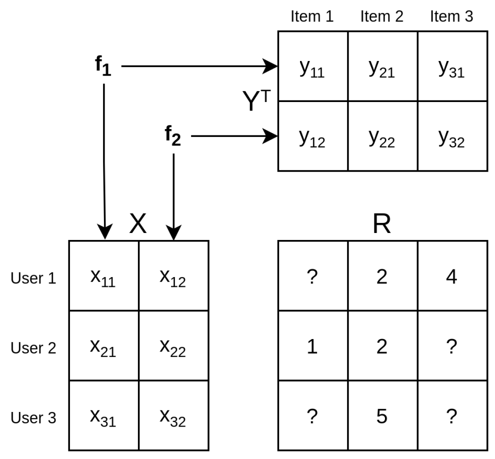


Figure 2.1: Illustration of Matrix Factorization. X and Y is a factorization of the rating matrix R using two latent factors. Notice that some values in R are unknown since the user has not rated that item. The model can estimate these value by using Equation 2.4.

2.3.3 Matrix factorization

An illustration of matrix factorization can be seen on Figure 2.1. The matrix factorization model is a model that tries to explain the ratings a user has given to a movie based on *k latent factors* [27]. The latent factors are factors that can be used to define items, inferred from the ratings patterns of the users. For movies, the factors might measure dimensions like comedy versus drama, amount of action, or possibly some uninterpretable dimension. The factors describing each item can be stored in a matrix $Y \in \mathcal{R}^{n \times k}$, where n is the number of items and k is the number of latent factors the model uses. The preferences each user has for each factor can be stored in a matrix $X \in \mathcal{R}^{m \times k}$, where m is the number of users. Given these matrices X and Y one can estimate what user i will rate item j:

$$\hat{r}_{ij} = x_i \cdot y_j \tag{2.4}$$

Let $R \in \mathcal{R}^{m \times n}$ be a sparse matrix of all ratings. To learn the user matrix X and item matrix Y the system minimizes the regularized squared error on the set of known ratings in R:

$$\min_{X,Y} \sum_{(i,j)\in K} (r_{ij} - x_i \cdot y_j)^2 + \lambda(||x_i||^2 + ||y_j||^2) \tag{2.5}$$

where K is the set of (i, j)-pairs for which $r_{ij} \in R$ is known, and λ is the regularization parameter. Regularization is necessary to prevent the model from overfitting to the known ratings in R.

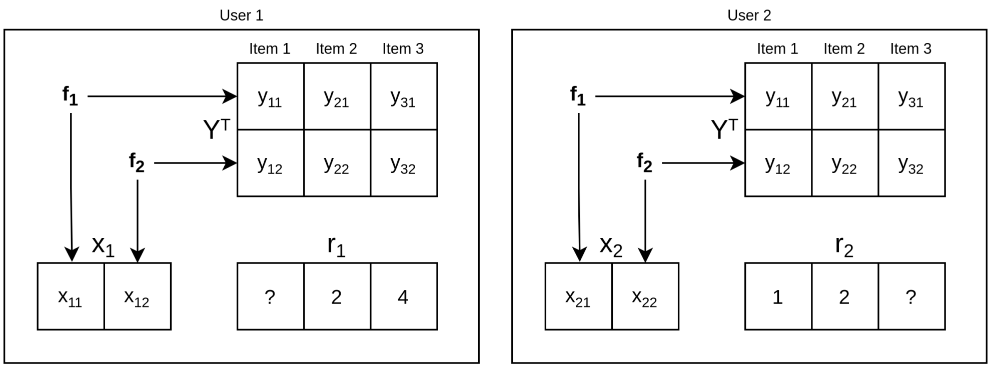


Figure 2.2: The matrix factorization model in Figure 2.1 distributed on the device of user 1 and user 2. Each user holds a copy of the full item matrix Y, while only holding their own vector in the user matrix X. They also only have access to the row in R with their own data.

Adding biases

Some users tends to rate items higher than others, and some items are generally rated higher than others. Some models therefore use biases to account for these variations which are independent of the latent factors. When using biases, $\hat{r}_{ij}$ is defined as:

$$\hat{r}_{ij} = x_i \cdot y_j + b_i + c_j \tag{2.6}$$

where b is the vector of user biases for all users and c is the vector of item biases for all items. To learn the user matrix X, item matrix Y, the user bias vector b and the item bias c the system minimizes the regularized squared error on the set of known ratings in R:

$$\min_{X,Y,b,c} \sum_{r_{ij} \in R} (r_{ij} - b_i - c_j - x_i \cdot y_j)^2 + \lambda(||x_i||^2 + ||y_j||^2) \tag{2.7}$$

2.3.4 Decentralized matrix factorization

The naive way to distribute a matrix factorization model across multiple devices is to distribute a copy of X and Y to every device. Distributing the full user matrix X to every device is however both unnecessary and a privacy violation. A user does not need the user vectors of other users to predict its own ratings. The user vector can also be considered private information as it can be used to learn about the preferences of a user. Instead of distributing the full user matrix, each user i should only have access its own user vector x_i.

An illustration of a decentralized matrix factorization model can be seen on Figure 2.2. The decentralized matrix factorization model consists of the full item matrix

Y and the personal user vector x_i of the user. If biases are used each user also needs a copy of the full item bias vector c, as well as their personal user bias b_i.

This decentralized matrix factorization architecture is used in several papers [28–30] with minor modifications. Read more about these papers in Section 2.4.

2.3.5 Alternating Least Squares

Alternating Least Squares (ALS) is a method used for optimizing X and Y [27]. ALS alternates between calculating the optimal value for X using Ordinary Least Squares (OLS) when holding Y fixed, and calculating the optimal value for Y using OLS while holding X fixed. This is repeated until convergence.

For decentralized matrix factorization it is possible for the users to independently calculate their optimal user vector x_i [28], since this calculation is only dependent on r_i and Y, both of which are locally available. Calculating the optimal value of Y is however impossible to do locally for decentralized matrix factorization. This is because the optimal calculation of Y is dependent on all values in X and R. The optimal solution of Y per iteration can therefore only be found if all devices participates in this calculation for each step. This would make the calculation vulnerable to participants dropping out due to failures, or simply because the user turned off their device.

2.3.6 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) [31] is an optimization algorithm that can be used for matrix factorization [27]. The algorithm loops through each algorithm in the training dataset and calculates the error:

$$e_{ij} = r_{ij} - \hat{r}_{ij} \tag{2.8}$$

where $\hat{r}_{ij}$ is calculated using either Equation 2.4 or Equation 2.6 depending on if biases are used or not. It then updates the appropriate user and item vectors in X and Y respectively:

$$x_i \leftarrow x_i + \eta\,(e_{ij}\,y_j - \lambda\,x_i) \tag{2.9}$$

$$y_i \leftarrow y_i + \eta\,(e_{ij}\,x_j - \lambda\,y_i) \tag{2.10}$$

where η is the step size, or learning rate used. If biases are used they are updated using:

$$b_i \leftarrow b_i + \eta\,e_{ij} \tag{2.11}$$

$$c_i \leftarrow c_i + \eta\,e_{ij} \tag{2.12}$$

SGD is simpler and more robust to participants dropping out than ALS. It is therefore more ideal when optimizing Y in a decentralized fashion.

2.3.7 RMSE

Root Mean Square Error (RMSE) is a frequently used statistical metric to measure model performance for various domains [32], including recommender systems. The RMSE of a matrix factorization model for a testset T is defined as:

$$\sqrt{\frac{1}{|T|}\sum_{i,j\in T}(r_{ij}-\hat{r}_{ij})^2} \tag{2.13}$$

where $\hat{r}_{ij}$ is calculated using either Equation 2.4 or Equation 2.6 depending on if a matrix factorization with or without biases is used.

2.4 Related work

Pouwelse *et al.* introduces a social peer-to-peer file-sharing paradigm in [33], which includes a fully decentralized recommender system. This recommender system uses memory-based collaborative filtering, where the users select which users to share their preferences with. The recommendations are then made based on these exchanged lists of preferences. A problem with this approach is that the users have to compromise their privacy for the recommender system to work, as is explained in Section 2.3.2.

There are numerous preceding approaches to distributed model-based collaborative filtering, typically using matrix factorization [34–37]. Most of these approaches are however meant for a server cluster architecture, and do not take privacy into account. However, recently several approaches which do have been proposed.

Ammad-ud-din *et al.* introduces a federated collaborative filtering method based on matrix factorization and FL [28]. The authors describe a method inspired by ALS and SGD that is used to update the recommendation model. The server hosts the item matrix Y, which each user participating in the training process downloads at the start of every epoch. Each user i then use OLS to calculate the optimal values for their own x_i based on their local data while holding Y fixed. Each user then calculates the gradients for Y using their local data and the newly calculated x_i. The gradients are then sent to the server where they are aggregated and applied to Y.

This algorithm is improved upon in a subsequent paper [29]. This paper improves upon the previous paper by integrating additional data sources into the recommendation model other than just user-item interactions. The authors demonstrate this method for movie recommendation where the additional data sources is information about a users (age, gender, location) and information about the item (genre, actors, director). This leads to increased performance, especially in the cold-start scenario where limited user-item interaction data is available.

These are examples of promising privacy-by-design solutions to recommender systems, but both examples use a centralized FL-architecture. They can therefore not be considered fully decentralized. Users are still dependent on an organization maintaining this aggregation server.

Hegedűs *et al.* presents a fully decentralized user-to-user approach in [30]. This paper might help answer RQ2, i.e. "What are the trade-offs for a fully decentralized user-to-user recommender system". In the paper they use both FL and GL to train a rating prediction model using matrix factorization with biases. They then compare the performance of FL and GL when taking into account network constraints. They also simulated users dropping out and coming online during the training process by using a real cellphone availability trace dataset.

Both the FL-implementation and the GL-implementation used the decentralized matrix factorization model described in Section 2.3.4. Each user holds a private user vector, a private user bias, a public item matrix and a public item bias vector. For each iteration of both FL and GL the user updates the local model using its local data. The FL-variant then sends the gradients of the public model to the aggregation-server where an average of all the participating agents gradients are applied to the public model. The public model is then distributed to the users again, and the network is then ready for the next iteration of the collaborative training process. GL does not send their gradients to an aggregation server. Instead, it chooses a random user and sends the public model to that user. That receiving user then merges the received public model with its own public model. This is then repeated in parallel for all users.

Hegedűs *et al.* also explores ways to increase the model transfer speed. They do this for FL by compressing the model gradients, and for GL by compressing the transmitted models. This way more iterations of the algorithm can be performed given the same network constraints.

The main focus of the paper is to compare FL and GL. They conclude that FL does not have a clear performance advantage to GL, which their data also suggests. This is despite GL not requiring any central server. However, the focus of the paper was to compare the results of both approaches. The performance of both the FL- and GL-algorithm was not very competitive to centralized methods. It could therefore be interesting to see if it is possible to improve their model.

Chapter 3

Comparing centralized and decentralized learning for handwritten digit recognition

To be able to give an answer to RQ1, i.e. "What are the challenges associated with Machine Learning that does not violate privacy?", this chapter will be used for exploring different ways to do machine learning while preserving privacy. This will give some insights into the challenges associated with privacy-preserving ML. The MNIST dataset [38] will be used as a simple benchmark dataset to test out different ideas on. Comparisons to a fully centralized model that does not preserve privacy will be done.

Hypothesis: It is expected that methods that preserve privacy will perform worse, as not having direct access to the data makes creating a useful model harder.

3.1 Baseline for comparison

To be able to compare decentralized methods to a centralized method a centralized baseline is needed. For simplicity, a fairly uncomplicated Multilayer Perceptron (MLP) is chosen for this. See Figure 3.1 for the structure of the network. The strong model is trained for 5 epochs using SGD on 12 000 images sampled from the MNIST training dataset. Retraining the model ten times with randomly initialized weights each time achieves an average accuracy of 94.01% (0.12 standard deviation) on the whole MNIST testset consisting of 10 000 images.

3.2 Majority Voting

An alternative to having one strong centralized model that produce a single prediction, is to use Majority Voting to aggregate the output from all the weak models into a single prediction.

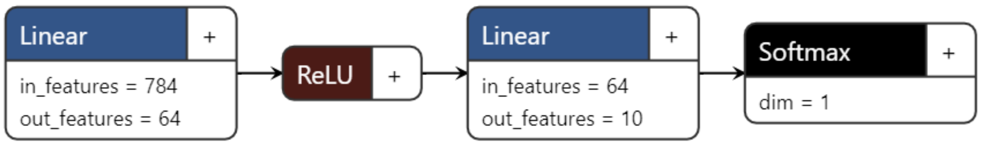
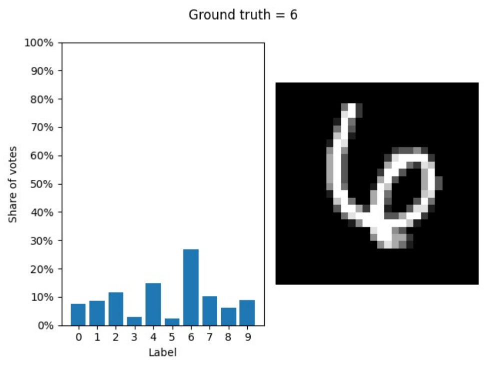
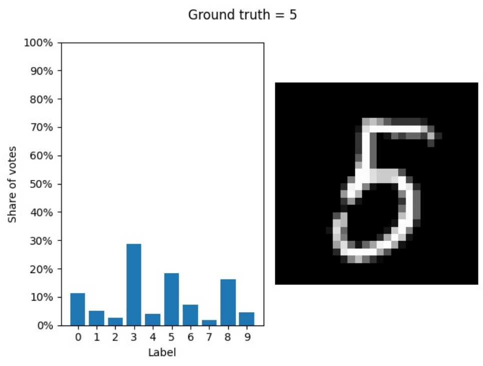


Figure 3.1: The architecture of a simple MLP



(a)



(b)

Figure 3.2: Two example voting results when the 375 models vote for which label to assign each image. On Figure 3.2a the image was correctly classified as a six due to that label receiving the highest share of the votes, while on Figure 3.2b the image was incorrectly classified as a three due to that label receiving the highest share of the votes.

To compare Majority Voting on multiple weak models to the baseline, a collection of models must be trained. Both the dataset, the MLP-architecture, and the number of training epochs (see Figure 3.1) must be the same as in the baseline experiment to make the comparison fair. With 12 000 data points in the training dataset, and if each weak model got 32 data points each, a total of 375 models are trained.

Training each model for 5 epochs on a training dataset of 32 data points achieves an average accuracy of 25.23% (0.27 standard deviation) when repeating this 10 times on the whole MNIST test dataset. This is indeed a very weak model, especially considering an accuracy of 10% can be achieved by blindly guessing. This is however to be expected with such a small training dataset. Considering the MNIST classification task has 10 different output classes, each individual model will on average only have approximately 3 data points for each class.

This accuracy is dramatically improved upon by using Majority Voting. Using Majority Voting to aggregate the 375 outputs into a single prediction for each data point in the test dataset yields an accuracy of 72.41% (1.22 standard deviation) when running 10 times with randomly initialized models. See Figure 3.2 for the result of two example votes. This is a dramatic improvement in accuracy compared to the accuracy of each individual model. The accuracy is however still

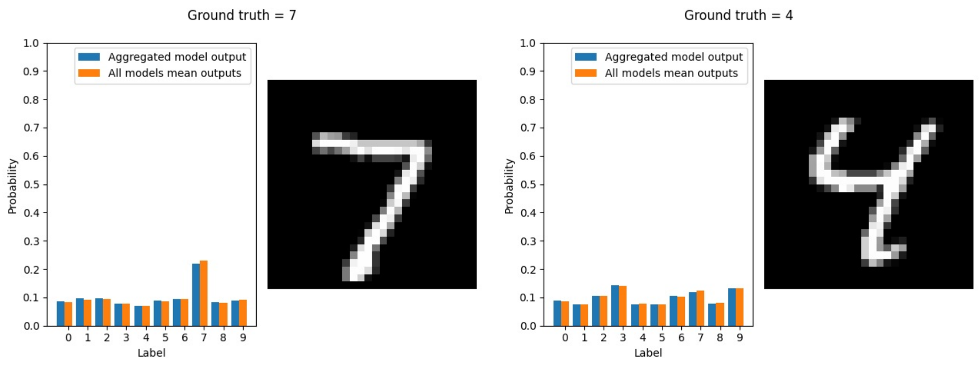


Figure 3.3: Two examples of the output from a model aggregated from 375 models (in blue), together with the mean output of those 375 models (in orange). Notice that the aggregated model has a very similar output to the mean output of the models it is aggregated from. "Probability" represents the output of the softmax-layer (see Section 2.1.1) of the models, and can be interpreted as how confident a model is about a classification.

significantly lower than the baseline model. It is also worth noting that the Majority Voting method has to calculate the output of all the 375 models and then aggregate it, while the baseline method only has to calculate the output of a single model. The strong centralized model has the same architecture as any of the weak models, the computational complexity of calculating the output of the centralized model is therefore identical to the computational complexity of calculating the output of a weak model. Majority Voting does not scale as well as the centralized approach, and might therefore be unsuited for applications where speed or resource use is critical.

3.3 Model aggregation

Calculating the output of every model to do Majority Voting can be very inefficient. A possible solution could be instead of calculating the output of every single weak model, one could aggregate the models into a single model before calculating the output of the aggregate model. One way of producing an aggregated MLP is to average the parameters of all of the models. Model aggregation is a technique often used with Gossip Learning, and is further described in Section 2.2.3. To test this method 375 models were trained in the same way as in Section 3.2, only this time they were all given the same initialization, as recommended in Section 2.2.3. An aggregated model is then produced by averaging their parameters.

This experiment is run 10 times. Looking at the outputs of the aggregated model one can immediately see that it is very similar to the mean output of the individual models that the aggregated model was made from. See Figure 3.3 for two examples of this. The average accuracy for the aggregated model is 32.98% (2.94 standard deviation) for all of the 10 runs. This is an improvement to the average

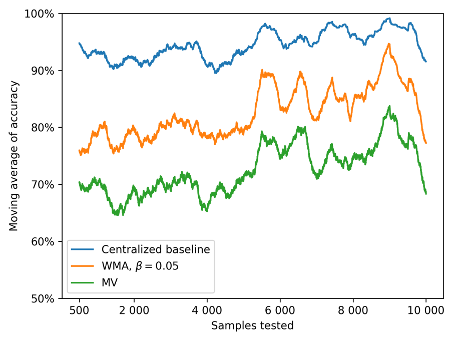


Figure 3.4: The moving average of the accuracy of the centralized baseline, WMA, and Majority Voting, plotted in blue, orange and green respectively. The moving average has a width of 500 samples, which is why the graph starts at sample 500.

accuracy of the individual models, which was 26.44% (1.44 standard deviation). This is far less than the accuracy achieved in Section 3.2. Interestingly enough, when using Majority Voting on the models trained with the same initialization one only achieves an accuracy of 36.51% (4.17 standard deviation). It therefore seems like a diverse initialization is beneficial when using MV, however this is not beneficial when using model aggregation. When differently initialized models were used, then only an accuracy of 16.89% (4.80% standard deviation) was achieved.

Model aggregation managed to lift the performance slightly above the average performance of the individual models. Although it is much cheaper computationally to use this model compared to Majority Voting, it failed to give accurate results. It is therefore not a viable solution on its own.

3.4 Weighted Majority Voting

Some models might be more accurate than others, Majority Voting might therefore not be the best choice as it weighs all the models predictions equally. Using Weighted Majority Voting could help better reflect this. The Weighted Majority Algorithm (WMA) was therefore tested using the same models as in the MV experiment in Section 3.2. WMA is an online algorithm, the ground truth is therefore revealed after each compound prediction the algorithm makes. This information is used to adjust the weights of the models (see Section 2.1.2).

Running WMA 10 times using the same models as in Section 3.2 produced positive results. When using the dicount factor $\beta = 0.05$, an average accuracy of 82.24% (0.78 standard deviation) was achieved, which is a decent improvement to the accuracy achieved in Section 3.2.

See Figure 3.4 for a comparison of the centralized model, WMA, and Majority Voting. Although WMA did outperform Majority Voting, it did not manage to achieve a similar performance to the centralized model, while also being a lot more computationally expensive.

3.5 Non-static test distribution

In previous experiments the test data was drawn from the same distribution throughout the whole experiment. It could be of interest to see if WMA outperforms the centralized baseline if the test distribution changes over time. Since WMA is an online algorithm it has the ability to adapt to changes over time by relying more on models that has performed well recently. The centralized strong model lacks this kind of flexibility.

This hypothesis was tested by testing the centralized strong model, WMA, and Majority Voting on a segmented test set. Each segment of the test set was set up to be biased towards a specific digit in the MNIST-dataset. WMA will be used with both $\beta = 0.05$ and $\beta = 0.2$ as the discount factor in two separate experiments.

In the first version of this experiment each segment will be 50% biased towards a specific digit. The results of this experiment can be seen on Figure 3.5. Majority Voting was pretty sensitive to shifts in test-bias, most likely due to some classes being harder than other for the weak models, but interestingly WMA was pretty robust towards these shifts, although the centralized strong model performed better throughout the whole experiment. The performance did not change much between the two tested discount factors β.

In a second experiment each segment was 100% biased, meaning every sample inside each segment will be of the same class. The results can be seen on Figure 3.6. Majority Voting was even more sensitive to the shifts in test-bias in this experiment. The centralized strong model was slightly more sensitive in this experiment. WMA performed very differently in this experiment. The accuracy of WMA plummeted as soon as the test-bias changed, but it quickly managed to adapt to the new bias. Higher value for the discount factor β made it adapt faster, which proved very beneficial in this scenario.

Note that the performance of Majority Voting and the centralized strong model was the same as in previous experiments, as only the order of the test data changed in this experiment. The overall performance of WMA for the different scenarios and discount factors can be seen on Table 3.1. As can be seen on the table, WMA managed to outperform the centralized baseline for the 100% bias scenario. This is not a fair comparison, as the centralized model is not an online algorithm and the setup of the experiment did not allow the model to learn anything while testing was going on. 100% bias scenarios are arguably also very unlikely. But it is still interesting to find that the predictions of several very weak models can be combined to produce comparable results to a centralized strong model, and in some scenarios even outperform it.

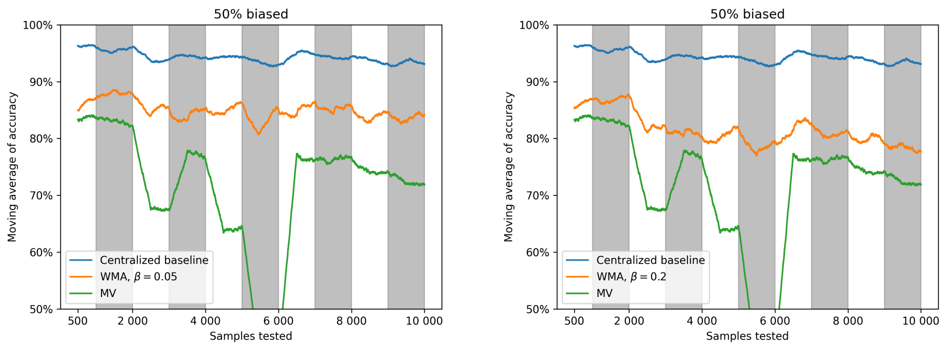


Figure 3.5: The moving average of the accuracy of the centralized baseline, WMA, and Majority Voting for a changing test-distribution. For each colored segment of the graph, the test distribution is 50% biased toward a specific digit, meaning every sample inside each segment will be of the same class. The moving average has a width of 500 samples, which is why the graph starts at sample 500.

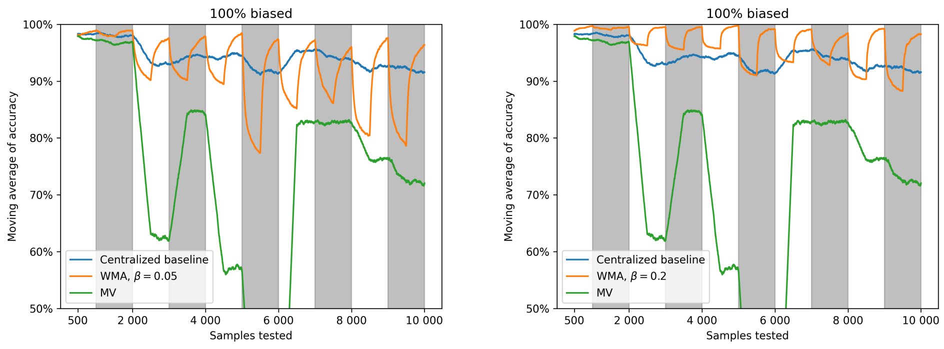


Figure 3.6: The moving average of the accuracy of the centralized baseline, WMA, and Majority Voting for a changing test-distribution. For each colored segment of the graph, the test distribution is 100% biased toward a specific digit, meaning every sample inside each segment will be of the same class. The moving average has a width of 500 samples, which is why the graph starts at sample 500.

	$\beta = 0.05$	$\beta = 0.20$
50% bias	84.68% (0.57)	81.32% (0.65)
100% bias	92.50% (0.30)	96.57% (0.25)

Table 3.1: The accuracy of WMA for non-static test distribution where each segment of the test distribution has a bias towards a specific label for two discount factors β.

C	Accuracy (standard deviation)
$\frac{1}{375}$	93.90% (0.19)
$\frac{1}{125}$	92.75% (0.20)
$\frac{1}{75}$	91.43% (0.33)
$\frac{1}{25}$	89.35% (0.16)
$\frac{1}{15}$	87.28% (0.41)
$\frac{1}{5}$	68.80% (4.01)
$\frac{1}{3}$	56.05% (6.23)
1	32.63% (2.33)

Table 3.2: The final accuracy of FL for different values of C

3.6 Federated Learning

Although WMA was able to compete with the centralized strong model in some scenarios, it did not manage to do this in general. When using WMA one also has to compute the output of every single model, which is not very computationally efficient, and would not scale well with more users.

A new approach could therefore be tested out. Instead of finding better ways to combine the outputs of several decentralized and weak models one could instead try to find a way to train a centralized strong model in a decentralized fashion. Federated Learning (FL) offers a way of doing this (see Section 2.2.2)

An experiment using Federated Learning was made by dividing the training data set consisting of 12 000 images between 375 agents like in earlier experiments. (see Section 3.2) For each iteration of the algorithm a fraction C of the agents was randomly selected to receive the federated model. Each agent then calculated the gradients for the model by training the model for one epoch using their local training data. All gradients calculated by every randomly selected agent was then averaged and applied to the federated model. For each epoch, this process was repeated until all agents have been selected exactly once. Five epoch was used to make it more comparable to earlier experiments.

The performance of the federated model was tested against the whole test set each time the federated model was updated. Different values for C was also tested out. See Table 3.2 for the results of the experiment. A graph of the accuracy of the federated model when using different values for C can be seen on Figure 3.7.

From the results it is clear the the lower the value of C is, the better the performance of the final federated model. When $C = \frac{1}{375}$ the accuracy is almost identical to the performance of the completely centralized method.

When $C = \frac{1}{375}$ only one agent is asked to calculate the gradients for each iteration. This makes the training process of the centralized model almost identical to the

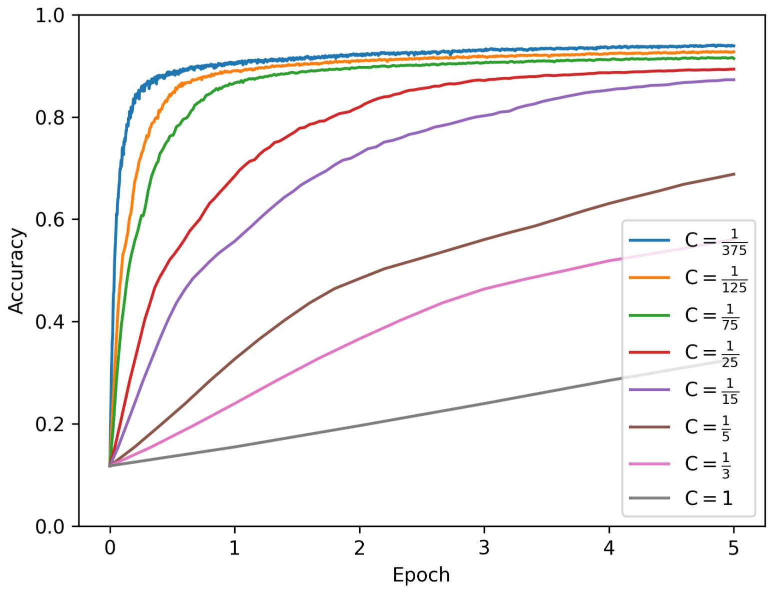


Figure 3.7: Accuracy of FL for different values of C

federated training process. In both processes the model is updated by calculating the gradients using 32 training images. The main difference is only the fact that in the centralized training process each data point in the training batch is randomly selected for each update, but when using federated learning all of the training batches are essentially segmented at the start when the data is divided among the agents.

Increasing the fraction of agents calculating gradients is analogous to increasing the size of the batch. Making the batch size significantly higher than 32 might not be beneficial in this scenario. Increasing the amount of agents calculating gradients at the same time therefore decreases the rate of which the model improves per epoch.

Computational efficiency is however not the only metric to consider in a real-world scenario. If only one agent calculates the gradients for the model for each iteration, the training process is going to take a lot more time as that agent has to first receive the model, finish calculate the gradients for the model, and finally transmit them to the server before the next agent can continue the process. The time required to finish an epoch of training is halved if the amount of agents calculate gradients at the same time is doubled. This makes the time to finish an epoch inversely proportional to C.

Having a lower value for C also makes it easier for the server to recreate each agents data. The server will have an easier time recreating the data of a user if less data went into the aggregated gradient. More can be found about this in

Section 2.2.4 and 2.2.5

Federated Learning is however more communication efficient than the previously tested ensemble methods in most scenarios. When using the model described in Figure 3.1 a transfer of a set of gradients is roughly equivalent to a transfer of a full model since one gradient has to be calculated for every parameter in the model. In the previous experiment with Federated Learning each agent has to transfer gradients one time per epoch, and the central server has to send the model one time for each agent per epoch. In the previous experiments using ensemble methods each agent had to transfer its model to every other agent in the network. The amount of data transferred is therefore roughly linear with regards to the amount of agents for federated learning, but quadratic when using ensemble methods.

3.7 Conclusion

This chapter explored different ways of doing machine learning while also preserving privacy. Two main approaches were tested: ensemble methods and Federated Learning. In the end FL performed better overall. The hypothesis was correct in general, as the centralized model performed better for all cases except when compared to WMA on a non-static test-distribution with a high bias.

Chapter 4

Federated Learning and Gossip Learning for rating prediction

In the previous chapter, it was concluded that FL provided better performance in general when compared to ensemble methods. This chapter will be used to compare FL to GL to answer RQ2, i.e. "What are the advantages and disadvantages of a fully decentralized user-to-user recommender system?"

GL is not dependent on a central aggregation-server, and is therefore fully decentralized, in contrast to FL. This chapter will therefore examine the advantages and disadvantages of using a fully decentralized user-to-user learning algorithm like GL.

Hegedűs *et al.* has already performed an experiment that compares the performance of GL to FL. They concluded that FL does not seem to have a clear performance advantage, GL might even have an advantage in certain scenarios. To see how well this generalizes, this experiment will be reproduced for other datasets as well in this chapter.

Hypothesis: It is expected that FL performs better than GL, as the aggregation-server makes it easier for the users to coordinate a useful factorization of the rating matrix.

4.1 Experiment setup

Both GL and FL will be tested in the experiment. Two variants of GL will be tested. In the first variant, any received model overwrites the local model, while in the second variant the received and local model is merged by a weighted average as described in Section 2.2.3.

The implementation of both the FL-algorithm and the GL-algorithm will be based on the pseudocode described in [30]. In the simulation used to simulate a network of agents using either FL or GL, the agents communicates synchronously in rounds. In [30] it was assumed that the transfer time of a model is 1 728 second, regardless of model size. The same assumption was therefore used in this experiment. For each iteration of the GL-algorithm, every agent sends a model to a randomly selected agent in the network. One iteration of the GL-algorithm therefore takes the time of one model transfer, i.e. 1 728 seconds. For each iteration of the FL-algorithm, the aggregation-server first distributes the federated model to every agent in the network. Every agent then calulates the gradients for that model using local data, which they then send to the aggregation-server. One iteration of the FL-algorithm therefore takes the time of two model transfers, i.e. 3 456 seconds.

The network structure used in the experiment in this thesis was a complete graph, meaning that every agent could communicate with any other agent in the network. This is different from the experiment by Hegedűs *et al.*, where a fixed random 20-out graph was used. It was also assumed that all agents are online throughout the entire experiment, as described in the churn-free scenario in [30]. The number of agents will equal the number of users in the given dataset. The algorithms will be evaluated by their RMSE compared to the time passed in the simulation.

4.1.1 Test set

As described in [30], the test set consists of 10 randomly picked ratings from each user. The exception to this is the Yahoo R3 dataset, where only 5 ratings per user was used due to some users only having 10 ratings in total. These ratings are not used when training. Each experiment is performed on 10 such train/test splits and then averaged.

4.1.2 Hyperparameters

These experiment uses the same hyperparameters as in [30], i.e. the regularization parameter $\lambda = 0.1$, the learning rate $\eta = 0.01$ and the number of latent factors $k = 5$.

4.2 Datasets

4.2.1 MovieLens 100K

The MovieLens 100K dataset[1] consists of 100 000 movie ratings that 943 users gave to 1 682 movies. Each rating is an integer from 1 to 5. Every user has rated at least 20 movies. The distribution of ratings can be seen on Figure 4.1. The density of the dataset is 6.24%.

[1]The MovieLens 100K dataset can be found at https://grouplens.org/datasets/movielens/100k/

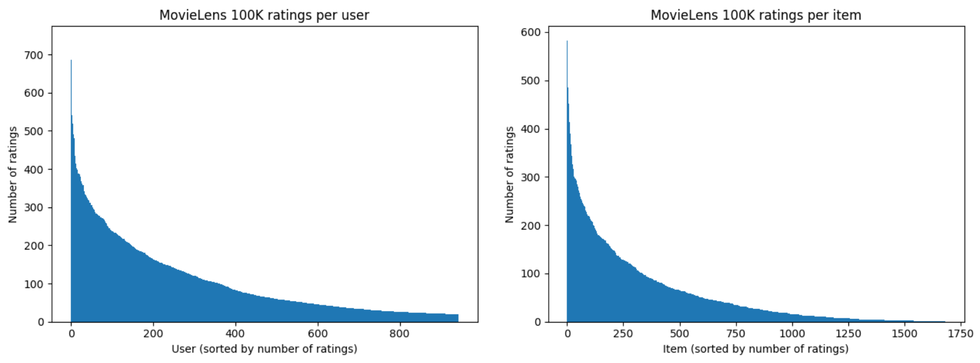


Figure 4.1: The distribution of ratings for the MovieLens 100K dataset

4.2.2 MovieLens 1M

The MovieLens 1M dataset[2] consists of 1 000 209 movie ratings that 6 040 users gave to 3 706 movies. Each rating is an integer from 1 to 5. Every user has rated at least 20 movies. The distribution of ratings can be seen on Figure 4.2. The density of the dataset is 4.47%.

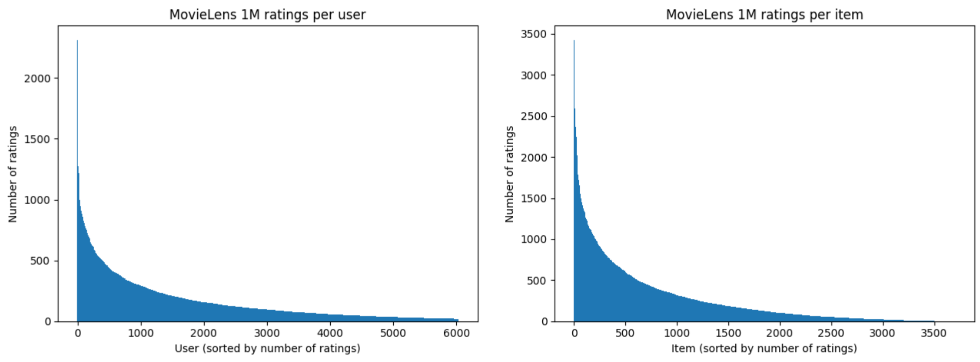


Figure 4.2: The distribution of ratings for the MovieLens 1M dataset

4.2.3 Netflix prize dataset

The full Netflix prize dataset[3] consists of 100 480 507 movie ratings that 480 189 users gave to 17 770 movies. Each rating is an integer from 1 to 5. Working with a dataset of this size is unpractical. A subset of this dataset is therefore used.

The subset is made by first selecting ratings belonging to the first 4 499 movies. This subset is further reduced by removing all ratings belonging to users that has rated less than 20 of these movies. Finally, random users are selected until the total amount of their ratings are above 1 000 000

[2]The MovieLens 1M dataset can be found at https://grouplens.org/datasets/movielens/1m/
[3]Netflix prize dataset can be found at https://www.kaggle.com/netflix-inc/netflix-prize-data

The final subset consists of 1 000 043 movie ratings that 11 656 users gave to 4 478 movies. The distribution of ratings can be seen on Figure 4.3. The density of the dataset is 1.92%.

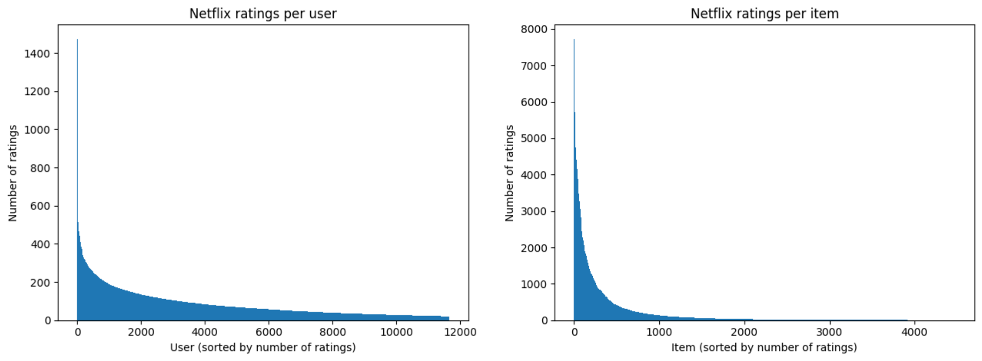


Figure 4.3: The distribution of ratings for the Netflix dataset subset

4.2.4 Yahoo R3

"R3 - Yahoo! Music ratings for User Selected and Randomly Selected songs"[4] is a dataset of music ratings collected from two sources. The first source is ratings collected when users interact with the Yahoo! Music services, while the second source is ratings for randomly selected songs collected from an online survey. The first source is used. This source contains 311 704 ratings that 15 400 users gave to 1 000 songs. Each rating is an integer from 1 to 5. The distribution of ratings can be seen on Figure 4.4

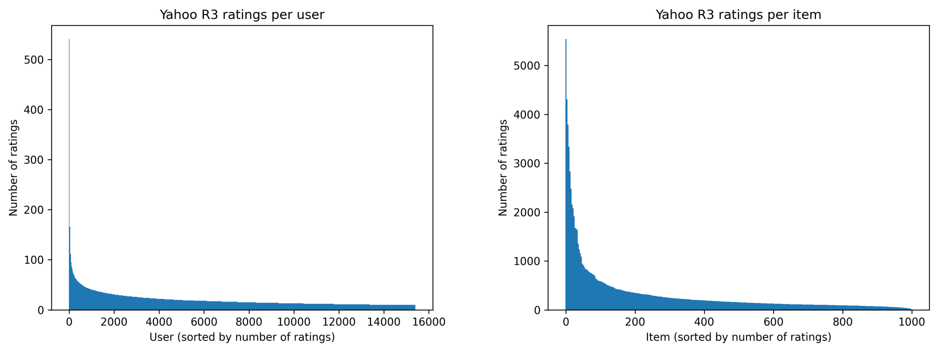


Figure 4.4: The distribution of ratings for the Yahoo R3 dataset

[4]The Yahoo R3 dataset can be found at https://webscope.sandbox.yahoo.com/

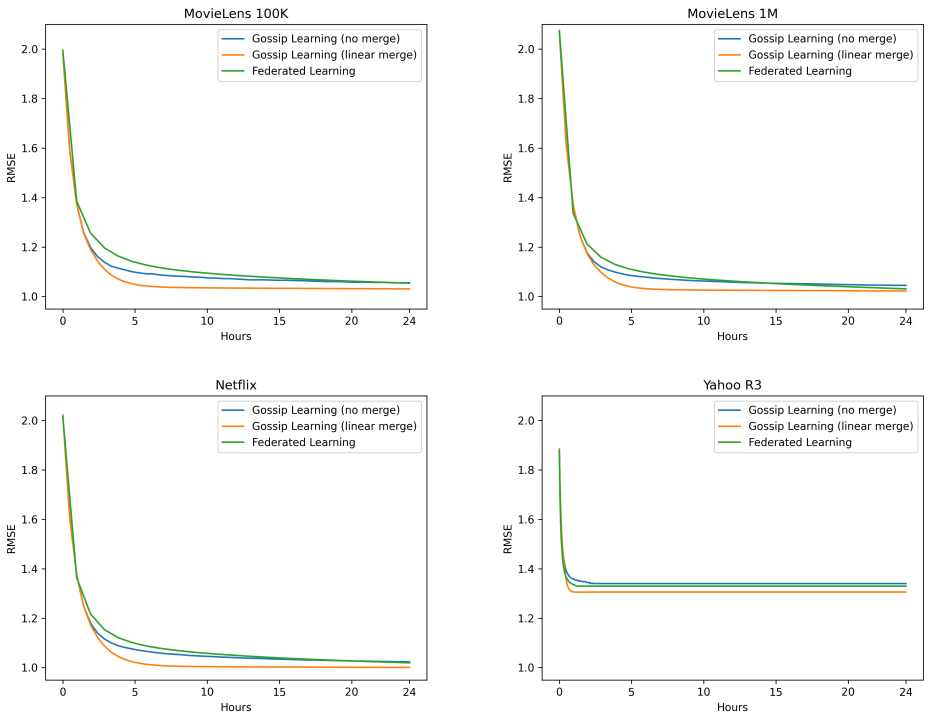


Figure 4.5: The results when comparing GL without merge, GL with linear merge, and FL, in blue, orange, and green respectively. The comparison was performed on four different datasets.

4.3 Results

The results of the experiment can be found on Figure 4.5. As can be seen on the Figures, all of the algorithms did converge to more or less the same value in all the experiments, but GL with merge seemed to perform the best in general. There was some inconsistencies in the results of this experiment compared to the original. The Federated Learning-algorithm was slightly worse than reported in [30], while both variations of GL performed slightly better in this experiment. GL did also not have any spike in the beginning.

4.4 Explanation of results

4.4.1 Difference in network topology

At first, it was assumed that the difference in results was due to the difference in network topology. In the original experiment a fixed random 20-out graph was used as the overlay network [30], while in this experiment a complete graph was used. The reasoning was that since each agent could only directly communicate with its 20 immediate neighbours in the original experiment, local updates would

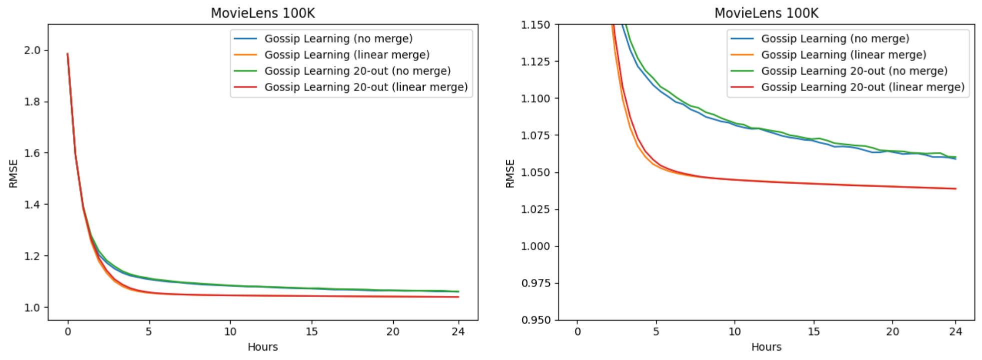


Figure 4.6: A comparison of a complete graph and a random 20-out graph as network topology. Tested on GL without merge, GL with linear merge, and FL for the MovieLens 100K dataset. Notice that the difference in performance is insignificant.

need more time to propagate throughout the network, compared to this experiment where every agent can communicate directly with every other agent.

To test this theory the previous experiment was repeated with a fixed random 20-out graph as the network structure instead of a complete graph. As can be seen on Figure 4.6, the difference in performance when using a fixed random 20-out graph was negligible. This is consistent with the findings of Giaretta and Girdzijauskas in [39]. They found that topologies based on graphs with a low diameter and a high link redundancy match the performance of a complete graph topology. A fixed random 20-out graph has both a high redundancy and a low diameter, which explains why the difference in performance was negligible.

4.4.2 Difference in implementation

After correspondence with one of authors of the paper, Dr. I. Hegedűs, it became apparent that the main reason for the inconsistencies was due to a difference in the implementation of the evaluation and initialization of the network.

In the Gossip Learning pseudocode in [30], the local models are described as being initialized by all agents in the beginning of the experiment. However, after correspondence with Dr. Hegedűs, it became clear that this was not used in their implementation. In their implementation, a row in X or Y was only initialized when that specific row was updated locally by an agent. Agents therefore do not have a model that can be used for inference at the start of the experiment. The authors solved this by making an agent predict the middle value if the agent has not received or initialized a model. The middle value is 3 if ratings are between 1 and 5. The mean rating of the datasets Hegedűs *et al.* tested on, among them MovieLens 100K and MovieLens 1M, lies between 3.5 and 3.6. Predicting 3 is therefore not a bad guess, which explains why the RMSE-score of Hegedűs *et al.* starts off low at the start of their experiment.

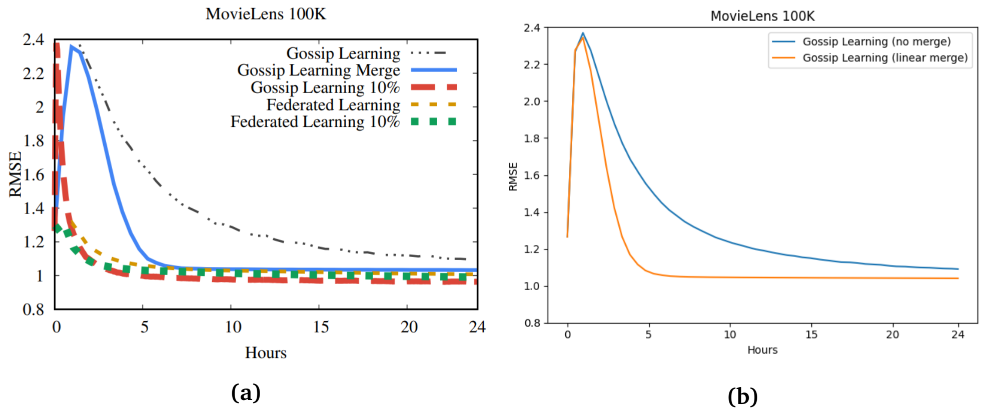


Figure 4.7: A comparison of the results from the original experiment by Hegedűs *et al.* and the results achieved in the experiment in this thesis. Figure 4.7a is from [30], while Figure 4.7b was created for this thesis. "Gossip Learning" and "Gossip Learning Merge" on Figure 4.7a is "Gossip Learning (no merge)" and "Gossip Learning (linear merge)" on Figure 4.7b respectively.

Another problem is when an agent has initialized its own latent user vector and user bias, but has not yet received the latent vector and bias of an item it would like to predict a rating for. Hegedűs *et al.* solved this by predicting the user's bias for these situations. The user bias is initialized to 0.5, which is a really bad guess for ratings between 1 and 5. This explains why the RMSE of GL had a big spike in the beginning of the experiment of Hegedűs *et al.* Every agent that initialized their own latent user vector and user bias will start guessing 0.5, which is a lot worse than guessing 3.

Since no rating in the training set is present in the test set, each agent will only stop guessing these bad predictions when it has received the latent vectors and biases for the items in its testset from other agents in the network. This explains why the implementation of Hegedűs *et al.* seems to converge slower than the implementation made for this thesis. This holds especially true for the Gossip Learning implementation without merge, as item vectors and biases are not propagated throughout the network as quickly with this method.

An experiment was performed to see if these changes in implementation was the main reason for the differences in results. In this experiment the previous experiment was modified to make agents predict the middle value when no vectors or biases were available. However, if the agent had initialized the user bias, but not the latent vector and bias of the item it wants to predict a rating for, it will predict the user bias. As can be seen on Figure 4.7, these changes made the results almost identical.

According to Dr. Hegedűs, the reason for implementing Gossip Learning this way was due to framework restrictions, as well as due to the fact that they did not want an agent to perform well only because of a lucky initialization. This design choice also made it easier for them to follow the spread of information throughout the network.

This implementation will not be used for the the rest of this paper. The latent vectors for the items are initialized randomly at the start of the experiment when using FL. The comparison is therefore more fair if the latent vectors for the items are initialized the same way at the start of the experiment for GL. It is especially unfair to GL to make the agents predict the user bias when they have not received the latent vector and bias of an item yet, as this guess will in the majority of cases be a really bad guess, even worse than just guessing the middle value, as explained in earlier paragraphs.

4.5 Evaluation of results

The results achieved in both this experiment and in the experiment performed by Hegedűs *et al.* are not very competitive when compared to centralized solutions. For example, using the *Surprise* library for Python [40], one can easily train a baseline predictor that produces better results[5]. This baseline method produces prediction using only the mean rating, a user bias, and an item bias. This is without any hyperparameter tuning.

A recommender system based on this baseline predictor does not offer any personalization, it will simply recommend items that are generally rated highly, i.e. the items with the highest item bias. A recommender system should therefore preferably be better than this baseline.

4.6 Conclusion

In this chapter FL and GL was compared for several datasets. GL performed better than FL when using the algorithms described by Hegedűs *et al.* in [30]. The result from [30] was recreated for GL if a different evaluation and initialization was used, but the result for FL were worse than the results achieved by Hegedűs *et al.* in [30]. From these results one can conclude that the hypothesis was wrong. GL achieved a better performance than FL.

[5]http://surpriselib.com#benchmarks

Chapter 5

Improving Gossip Learning Matrix Factorization

GL performed better than FL in the experiments performed in Chapter 4. GL is also a fully decentralized user-to-user algorithm, as opposed to FL which is dependent on a central aggregation-server. This lack of dependency can be considered an advantage in itself.

As mentioned in Chapter 4, the results of both GL and FL were not particularly competitive when compared to centralized methods. Methods to improve GL even further for recommender systems is therefore explored in this chapter.

Hypothesis: Several aspects of the GL-algorithm described by Hegedűs *et al.* can be improved. Different strategies for merging models can improve the quality of the merged models, and different initialization of the latent vectors and biases can improve convergence speed.

5.1 Strategies for model merging

5.1.1 Keep oldest

A problem with the no merge variant is that the incoming model overwrites the entire local model regardless of if the parameters of the local model, i.e. the latent factors and biases, have been updated more times. An "older" parameter will in this thesis refer to the parameter that has been updated the most times when compared to another less updated, i.e. "younger", parameter.

A possible improvement to the no merge strategy could be to only overwrite a local parameter if the corresponding parameter in the incoming model is older. This way a younger parameter will never overwrite an older parameter. For the pseudocode of the algorithm see Algorithm 1.

An experiment was ran on the MovieLens 100K dataset comparing this merge strategy to the previously tested merge strategies. The result of this experiment

Algorithm 1 Keep Oldest Merge

```
1: procedure KeepOldestMerge($Y, c, t, \tilde{Y}, \tilde{c}, \tilde{t}$)
2:     for $j \leftarrow 1 \ldots n$ do                          ▷ $n$: number of items
3:         if $\tilde{t}_j > t_j$ then            ▷ $t_j, \tilde{t}_j$: age of the parameters of item $j$
4:             $t_j \leftarrow \tilde{t}_j$
5:             $Y_j \leftarrow \tilde{Y}_j$
6:             $c_j \leftarrow \tilde{c}_j$
7:         end if
8:     end for
9: end procedure
```

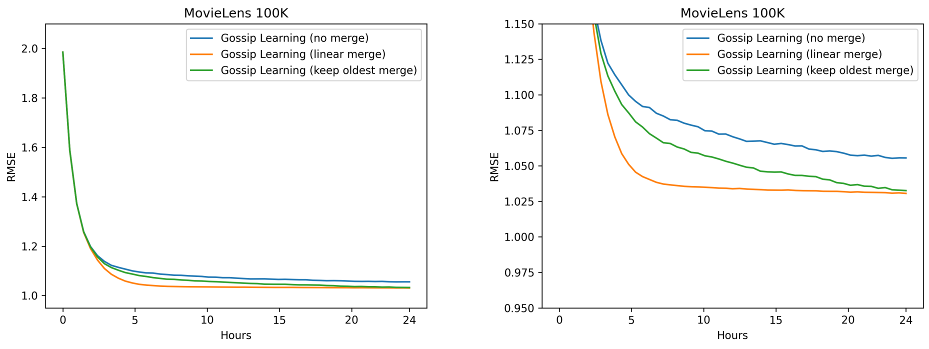


Figure 5.1: A comparison of the no merge, linear merge, and keep oldest merge variants, in blue, orange, and green respectively. Notice that keep oldest merge converges slower than linear merge, but keep oldest merge achieves the same RMSE at the end of the experiment.

can be seen on Figure 5.1. As can be seen on the figure, the *keep oldest* variant converged faster than the no merge variant, but slower than the linear merge variant. It is however also worth noticing that the keep oldest merge variant achieved the same RMSE at the end of the experiment. This could suggest that averaging model parameters speeds up convergence, but keeping the oldest parameter can be beneficial if given enough time.

5.1.2 Polynomially weighted average merge

The weighted average used in earlier experiments is a linearly weighted merge. This means that a parameter that has been updated twice the amount of times compared to another parameter, is weighed twice as heavily as the younger parameter in the weighted average performed when merging. It is not necessarily the case that this is beneficial. It is possible that a non-linear merge might prove more effective.

In the previous experiment it was shown that keeping the oldest parameter produced a good RMSE if given enough time. Keeping the oldest parameter is essen-

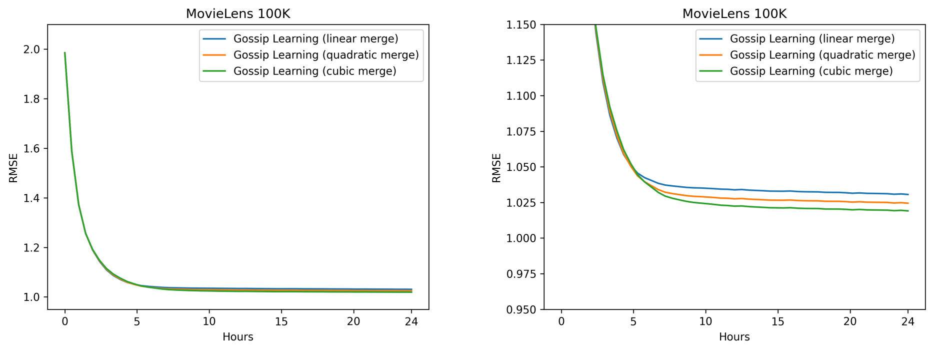


Figure 5.2: A comparison of polynomial merge types of different degrees. Linear, quadratic, and cubic merge was compared, shown on the graph in blue, orange, and green respectively. Notice that the performance of each merge was almost identical, but the graph suggests that a higher degree polynomial merge performs slightly better than a lower degree polynomial merge.

tially the same as the oldest parameter having an infinitely heavier weight than the younger parameter in a weighted average. It would be of interest to explore if a less extreme weighting would prove beneficial for long term performance, while still converging more or less as fast as when using linear merge. The next experiment should therefore see if a polynomially weighted merge improves the performance of GL. For the pseudocode of the algorithm see Algorithm 2. Note that the polynomially weighted average merge of degree one is identical to the linear merge variant used in the experiment of Hegedűs *et al*.

Algorithm 2 Polynomially Weighted Average Merge

```
 1: procedure PolyAvgMerge($Y, c, t, \tilde{Y}, \tilde{c}, \tilde{t}, d$)    ▷ $d$: degree of the polynomial
 2:     for $j \leftarrow 1 \ldots n$ do                                    ▷ $n$: number of items
 3:         if $\tilde{t}_j \neq 0$ then                        ▷ $t_j, \tilde{t}_j$: age of the parameters of item $j$
 4:             $w \leftarrow \frac{\tilde{t}_j^d}{t_j^d + \tilde{t}_j^d}$        ▷ $w$: weight of the incoming parameter
 5:             $t_j \leftarrow \max(t_j, \tilde{t}_j)$
 6:             $Y_j \leftarrow (1-w)Y_j + w\tilde{Y}_j$
 7:             $c_j \leftarrow (1-w)c_j + w\tilde{c}_j$
 8:         end if
 9:     end for
10: end procedure
```

Two new variants of polynomial merge is considered: quadratic merge and cubic merge. When using these merge strategies, a parameter that has been updated twice the number of times compared to another parameter will be weighted 4 (2^2) and 8 (2^3) times heavier than the other parameter respectively. The performance increase of this merge strategy was only slight, as can be seen on Figure 5.2.

5.1.3 Exponentially weighted average merge

When using a polynomially weighted merge, small differences in parameter age will matter less as the age of this parameter increases for all agents in the network. Consider a parameter of age x. Consider an older version of this parameter with age $x + k$ where k is a constant. If the degree d of the polynomial merge is also constant, then the weight of the older parameter will go towards the weight of the younger parameter as x increases:

$$\lim_{x\to\infty} \frac{(x+k)^d}{x^d} = 1$$

If the weight of the older parameter is equal to the weight of the younger parameter, the older and the younger parameter will be weighed equally in the weighted average when merging. This might not be beneficial in some cases.

Consider an exponentially weighted average merge instead:

$$\lim_{x\to\infty} \frac{e^{x+k}}{e^x} = e^k$$

When using a exponentially weighted merge, even as x goes towards infinity, the oldest parameter will always be weighed significantly higher than the youngest parameter. An experiment was ran when using exponentially weighted merge on the MovieLens 100K dataset to find out if this property is useful. See Algorithm 3 for the pseudocode for this merge strategy.

Algorithm 3 Exponentially Weighted Average Merge

```
1: procedure ExpAvgMerge($Y, c, t, \tilde{Y}, \tilde{c}, \tilde{t}$)
2:     for $j \leftarrow 1 \dots n$ do                          ▷ $n$: number of items
3:         if $\tilde{t}_j \neq 0$ then
4:             $w \leftarrow \frac{e^{\tilde{t}_j}}{e^{t_j} + e^{\tilde{t}_j}}$     ▷ $w$: weight of the incoming parameter
5:             $t_j \leftarrow \max(t_j, \tilde{t}_j)$
6:             $Y_j \leftarrow (1-w)\,Y_j + w\,\tilde{Y}_j$
7:             $c_j \leftarrow (1-w)\,c_j + w\,\tilde{c}_j$
8:         end if
9:     end for
10: end procedure
```

As can be seen on Figure 5.3, the exponential merge improved slightly slower than linear merge in the beginning, but as linear merge converged exponential merge kept on improving, much like keep oldest merge. In this scenario it does seem like the properties of exponential merge proves useful.

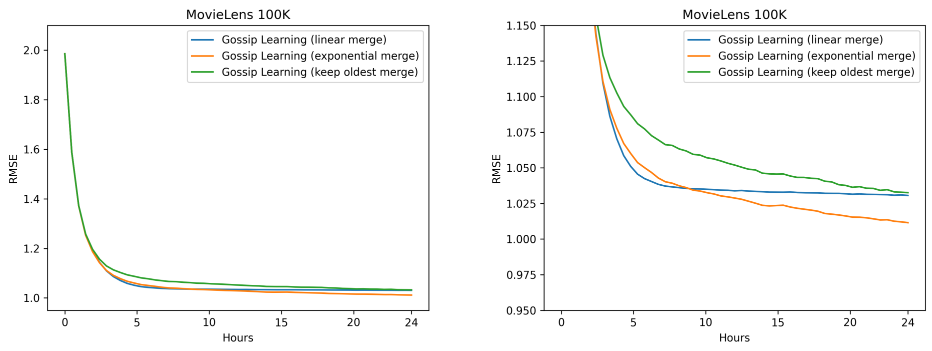


Figure 5.3: A comparison of linear merge, exponential merge, and keep oldest merge, shown on the graph in blue, orange, and green respectively.

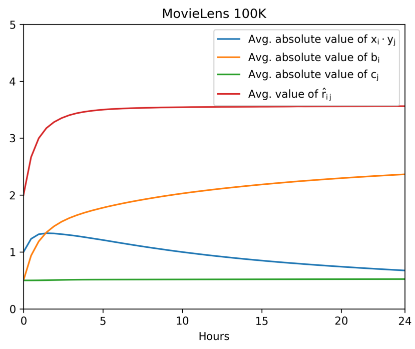


Figure 5.4: The agent-average absolute value for each of the components of $\hat{r}_{ij}$ plotted over the whole training process. Gossip Learning with linear merge on the MovieLens 100K dataset.

5.2 Analysis of prediction components

A prediction for what user i would rate item j, referred to as $\hat{r}_{ij}$, is made using Equation 2.6. The prediction $\hat{r}_{ij}$ can be divided into three components:

1. $x_i \cdot y_j$ - The dot product of the user feature vector for user i and item feature vector for item j.
2. b_i - The user bias for user i.
3. c_j - The item bias for user j.

To make further improvements to the algorithm it could be useful to try to better understand the importance of the different components of the prediction model. This is done by recording the agent-average absolute value of each of these components on the test set for each training epoch.

Figure 5.4 shows how the the components of the prediction $\hat{r}_{ij}$ evolves during the training process for Gossip Learning with linear merge on the MovieLens 100K

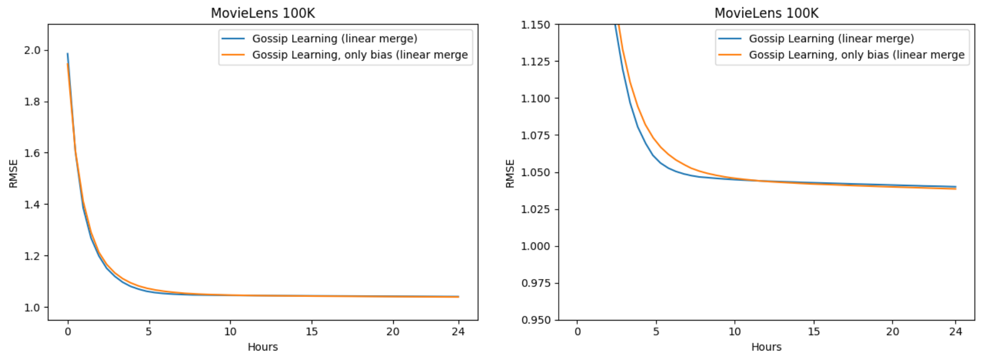


Figure 5.5: A comparison of GL with and without latent vectors, in blue and orange respectively. The version without latent vectors, the only bias variant, achieved an identical score to the version with with latent vectors.

dataset. On Figure 5.4 one can see that the user bias tends to quickly dominate the other components. While the item bias very slightly increases, the dot product actually decreases.

When using the model update rule described in Section 2.3.6, the user bias of an agent is adjusted one time for every rating that the agent holds every time that the update rule is applied. Since an agent holds the ratings given by a single user it will never have more than one rating for a given item. An item bias will therefore be adjusted at most a single time for every application of the update rule. This means that the user bias will be adjusted significantly faster than the item bias. This leads to the user bias capturing a significant part of the output.

Due to the importance of the biases in the previous experiments, it could be interesting to see how well a model using only biases would compare to the one described by Hegedűs *et al.* This model will calculate the predicted rating $\hat{r}_{ij}$ using Equation 5.1:

$$\hat{r}_{ij} = b_i + c_j \tag{5.1}$$

An average prediction closer to the true average rating will in general yield a better score. The models should therefore predict the same rating on average at the start of the experiment to make the comparison fair. The models using only biases will be initialized with all user and item biases set to one, such that the expected rating prediction of both models will be two at the start of the experiment.

A comparison of these models can be seen on Figure 5.5. Surprisingly the model using only biases achieves an identical score to the original model at the end of the training. The original model has a slightly better performance at the start of the experiment, but this seems to be mainly due to the fact that the original version increased its average rating prediction to match the true average rating slightly faster due to it being able to increase the output of both the dot product of the latent vectors and biases at the same time. The bias-only model catches up to the model of Hegedűs *et al.* at about the same time that the average value of the dot product goes back to its starting value of one, as can be seen on Figure 5.4.

5.3 Data-based Bias Initialization

To try to further improve the bias-only model, the local data of an agent was utilized to initialize the biases. Each user bias is set to the average of all ratings made by that user. For each item that the user has rated, the corresponding item bias is set to the difference between that rating and user bias. The item bias for unrated items are set to zero. See Algorithm 4 for the algorithm used to initialize the biases of a user i.

Algorithm 4 Data-based Bias Initialization

```
 1: procedure INIT($i$)                                    ▷ $i$: the index of the user
 2:     $D \leftarrow \{j \in \{1, \ldots, n\} | r_{ij}$ is defined in training set$\}$
 3:     $b_i \leftarrow \frac{1}{|D|} \sum_{j \in D} r_{ij}$
 4:     for $j \leftarrow 1 \ldots n$ do
 5:         if $j \in D$ then
 6:             $c_j = r_{ij} - b_i$
 7:             $t_j = 1$
 8:         else
 9:             $c_j = 0$
10:             $t_j = 0$
11:         end if
12:     end for
13:     return $(t, b_i, c)$
14: end procedure
```

This initialization improved the test-RMSE quite significantly as can be seen on Figure 5.6. This is without using the latent vectors for prediction. As the latent vectors for the items would not have to be transferred when using this model one could also drastically decrease the model transfer time. The original model needs to send six values per item, five of which is the latent factors for that item, and the sixth value being the item bias. The bias-only variants only needs to send the item biases, shrinking the model size to a sixth of the original size, which in practice would enable much faster training. This was however not taken into account in the previous experiments.

Interestingly the bias-only model with data-based bias initialization did not work that well if exponential merging was used. The reason for this will be further investigated in Section 5.4.

It is worrying that bias-only version of the GL-algorithm can achieve comparable or even better performance to the GL-algorithm introduced in [30]. A recommender system built on top of a model that only uses biases to predict a rating will as

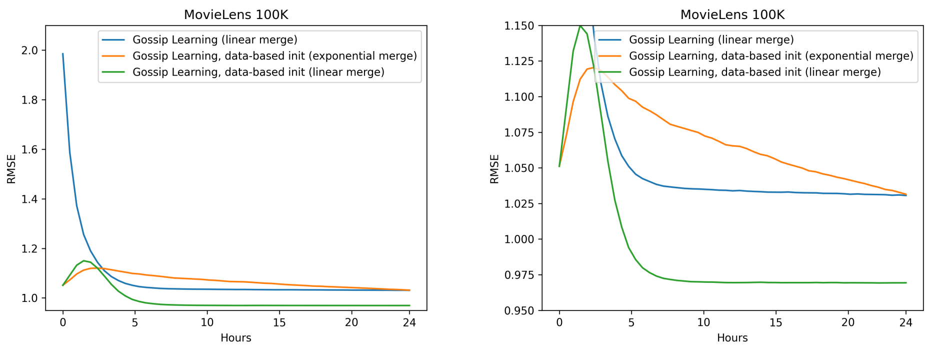


Figure 5.6: A comparison of GL with latent vectors and linear merge, GL without latent vectors with data-based initialization and exponential merge, and GL without latent vectors with data-based initialization and linear merge, shown in blue, orange, and green respectively. The variant using data-based initialization and linear merge merge outperformed the other variants, without using latent vectors. Notice that GL without latent vectors with data-based initialization and exponential merge performed worse the the linear variant.

mentioned in Section 4.5 recommend the same items for every user, i.e. the items with the highest average rating.

5.4 Analysis of item bias evolution

To better understand the impact of different merge strategies on different models it could be useful to try to track how item biases change over time. This is done by simply recording the value of the item biases throughout the training process. The evolution of five item biases when using the original initialization used by Hegedűs *et al.* can be seen on Figure 5.7. Each line represents the agent-average value of an item bias over the entire training process, with the fill-area representing the standard deviation.

It becomes apparent that each merge strategy significantly affects how the item biases evolve over the training period. When using linear merge the biases changed slightly in the beginning, but the agents in the network quickly achieved a fairly consistent consensus and the biases did not change a lot after this. On the opposite extreme there was the keep oldest merge strategy. When using this strategy the biases changed a lot over time, and the value for each bias deviated significantly from agent to agent. In the middle of these extremes was the exponential merge strategy. The values of the biases changed significantly over time, but for most of the biases there seemed to be a higher degree of consensus than with the keep oldest merge strategy.

The evolution of the item biases when using data-based bias initialization can be seen on Figure 5.8. There are some similarities to the results in Figure 5.7. Linear merge achieved consensus, "keep oldest"-merge changed a lot, and exponential

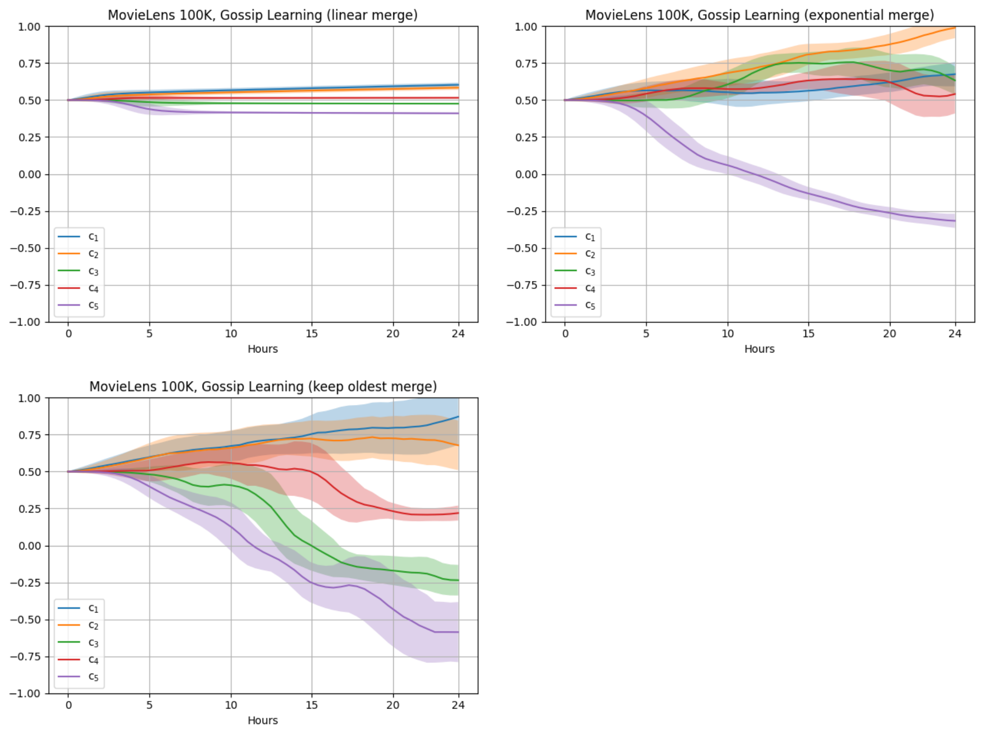


Figure 5.7: The evolution of item biases when using the original initialization. Three merge types are used: linear merge, exponential merge, and keep oldest merge.

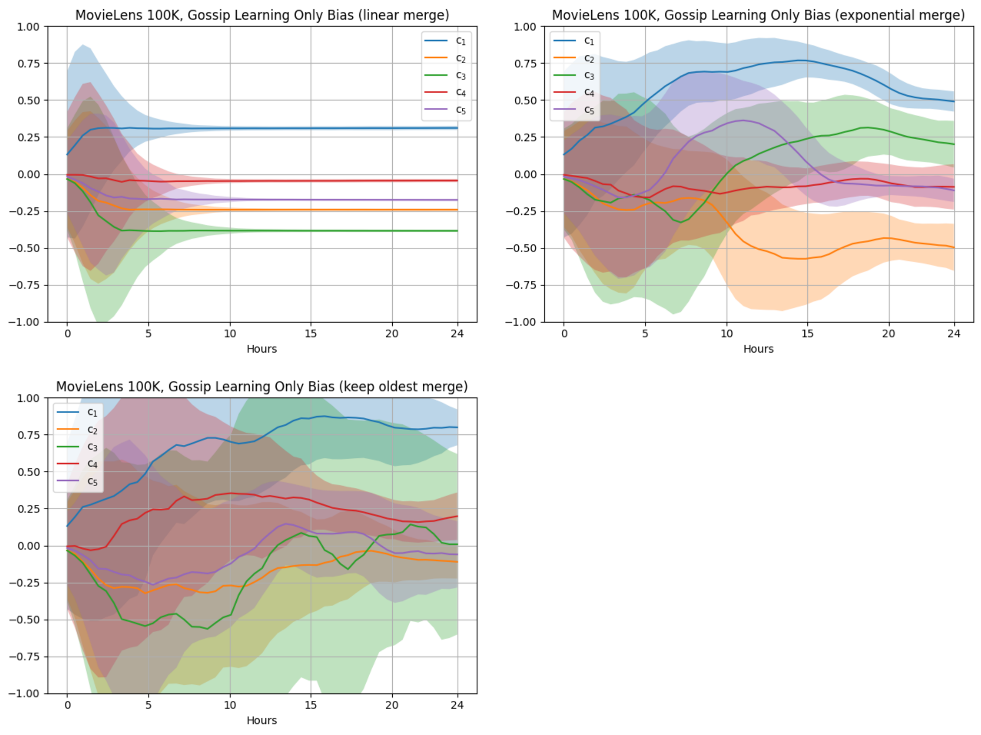


Figure 5.8: The evolution of item biases when using data-based initialization. Three merge types are used: linear merge, exponential merge, and keep oldest merge.

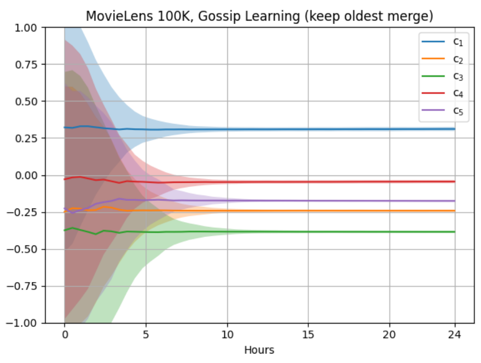


Figure 5.9: The evolution of item biases when using data-based initialization and linear merge. Item biases that are initialized to zero due to not having a rating for that item locally are not included in the calculation of the mean and standard deviation for that item until they receive that item bias from another agent.

merge is somewhere in between. One of the differences when using data-based initialization is that the biases had a much higher degree of variation from agent to agent, especially right after the beginning of the experiment.

When using data-based initialization an item bias is not set to zero during initialization by an agent if that agent holds a rating for that item. This leads to some variation at the very start of the training process, especially for items rated by a lot of users. This can be seen on Figure 5.8 where the item bias c_1, the item bias for *Toy Story (1995)*, has the highest amount of deviation at the very start of the experiment, which makes sense considering how popular the movie is.

Right after initialization the deviation for all of the item biases start to increase even more as non-zero biases are sent to agents that previously set them to zero due to not having a local rating for that item. The deviation tends to decrease after this, some merge methods decreasing it more than others. Linear merge seems especially suited for achieving a consensus across the network, quickly reducing the deviation from agent to agent.

So why is linear merge especially good when using data-based bias initialization? When comparing the results in Figure 5.7 to the results in Figure 5.8 it seems that when using data-based bias initialization the network is a lot more flexible, being able to find fairly distinct values for the biases pretty quickly, while when not using it the biases do not deviate too much from their initial value. This comparison can be fairly misleading. The mean of an item bias is calculated from all agents, including agents that has put that item bias to zero during the initialization because they hold no rating for that item. The first time an agent receives an item bias for an item they hold no rating for, they will simply replace that zero-valued item bias with the received item bias. See Algorithm 2 for more information about this. Note that linear merge is a polynomial merge of degree one.

Only calculating the mean from the non-zero values for the item biases produces

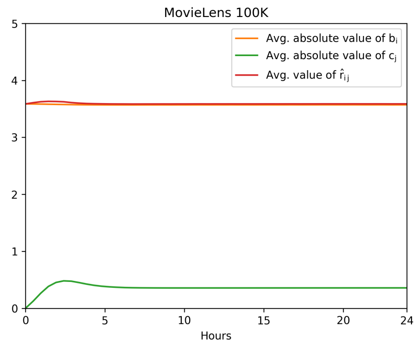


Figure 5.10: The agent-average absolute value for each of the components of predictions $\hat{r}_{ij}$ made when testing on the testset. These values are plotted for the whole training process when using GL with linear merge and data-based initialization on the MovieLens 100K dataset. The average rating of the used MovieLens 100K testset is 3.59, the predictions are therefore almost equal to the mean from the start.

the results that can be seen in Figure 5.9. Here it becomes clear that linear merge is not somehow more dynamic when using data-based bias initialization. The average value of the item biases arguably change even less in Figure 5.9 than the linear merge variant in Figure 5.7. The strength of linear merge seems to be that it quickly achieves a consensus that tends to be close to the average value of the initial values. In the special case of data-bases bias initialization this proved to be quite useful, simply because the item biases are initialized to the initial item bias. Each item bias will therefor converge to an estimate of the average deviation from the user bias for that specific item for each user, which is not a terrible item bias.

The reason for exponential and keep oldest merge outperforming linear merge in the first experiment seems to mainly because they are more dynamic and flexible than linear merge. Linear merge did not manage to deviate a lot from the initial values it was assigned and was therefore unable to fit the data in a particularly useful way. Exponential and keep oldest merge managed to change the item biases quite significantly. This might have been especially important in the first experiment since the value of the biases has to increase quickly for the predictions to come closer to the mean rating.

5.5 Reintroducing user and item vectors

The user and item vectors are necessary to achieve any real personalization. They should therefore be reintroduced. The initialization introduced in Section 5.3 was an improvement compared to the original implementation, it will therefore be used when the vectors are reintroduced. When using this initialization the mean prediction starts at the mean rating of the testset. This can be seen on Figure 5.10. The expected value of $x_i \cdot y_j$ should therefore be zero to prevent the average

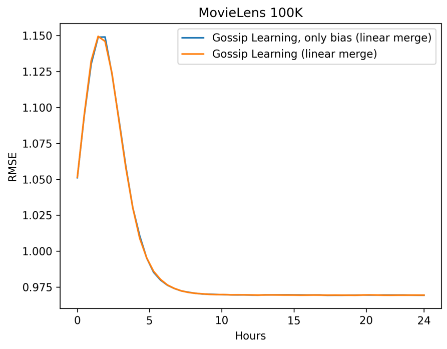


Figure 5.11: The performance of GL with data-based initialization without user and item vectors, compared to the performance of GL with data-based initialization and user and item vectors. As can be seen on the Figure, the reintroduced user and item vectors did not make a noticeable difference in performance.

prediction from becoming either too high or too low. The vectors will therefore be randomly initialized by picking values from a normal distribution $\mathcal{N}(\mu = 0, \sigma = 0.1)$

The results when reintroducing the item and user vectors can be seen on Figure 5.11. Reintroducing the user and item vectors did not change the performance significantly. This was disappointing as the latent vectors are necessary for any real personalization. This could be due to the agents requiring more iterations to be able to produce useful latent vectors. A simple experiment was therefore run where a ten times faster model transfer speed was assumed. This is not unreasonable, considering that the original experiment by Hegedűs *et al.* assumes an extremely low-bandwidth scenario having a bandwidth of only 0.35 kbps for the MovieLens 100K dataset [30]. A performance comparison between this experiment and the original experiment is however no longer valid, but this is not the goal of the experiment. The goal is to see if a fully decentralized matrix factorization can provide useful personalization.

The results of this experiment can be seen on Figure 5.12. Even with ten times higher bandwidth, leading to ten times more iterations, the agents did not manage to create latent vectors that improved upon the performance.

An interesting observation is that the numerical value of the gradients for each latent factor is generally much lower than for example the gradients for the biases. This is because the gradient of the latent factors in X is dependent on the values of the latent factors in Y and vice-versa, see Equation 2.9 and Equation 2.10. The latent factors are picked from a normal distribution $\mathcal{N}(\mu = 0, \sigma = 0.1)$ during initialization, and are therefore close to zero on average. Perhaps the agent will learn useful latent vectors faster if a higher learning rate is used for the latent factors? Let η_v denote the learning rate for the vectors, while η_b denotes the learning rate for the biases. An experiment was run using a higher learning rate

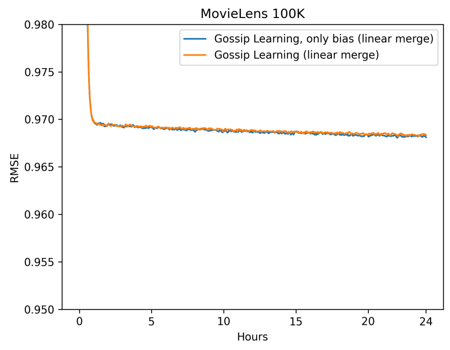


Figure 5.12: The performance of GL with data-based initialization without user and item vectors, compared to the performance of GL with data-based initialization and user and item vectors. The bandwidth in this experiment is ten times higher than in the experiment that can be seen on Figure 5.11. As can be seen on the Figure, the reintroduced user and item vectors did not make a noticeable difference in performance.

for the latent vectors, while keeping the same learning rate for the biases, i.e. $\eta_v = 0.1$ and $\eta_b = 0.01$.

The results of this experiment on all datasets can be seen on Figure 5.13. A higher learning rate for the latent vectors improved the performance of the model for all datasets except the Yahoo! dataset. Useful latent vectors was therefore produced for all datasets except the Yahoo! dataset. The latent vectors produced for the Yahoo! dataset decreased the performance of the model. The Yahoo! dataset has a very high amount of users, but a very low amount of ratings per user. This shows that collaborative filtering is challenging with little data per users, which lead to the algorithm overfitting to the small amount of data in the training set.

5.6 Conclusion

Throughout this chapter it was attempted to improve upon the results achieved in Chapter 4. Multiple techniques that improved the performance was developed and analysed. It was also discovered that the latent vectors did not provide useful personalization in the original experiment. A method that increased their usefulness again was discovered. From this chapter one can conclude that it is possible to create personalized recommendation in a fully decentralized way.

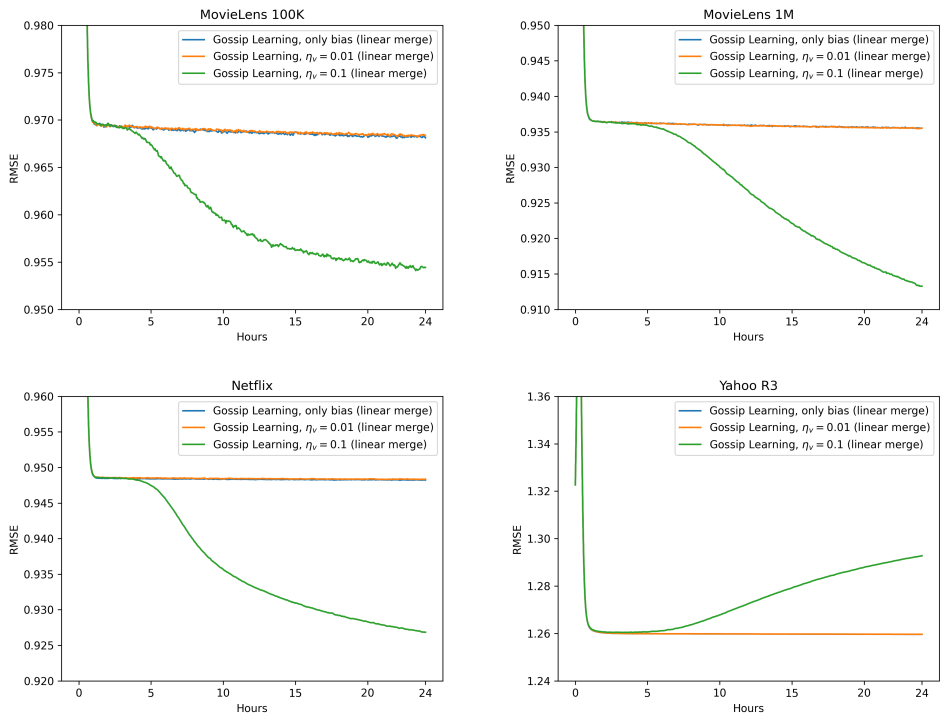


Figure 5.13: Results with data-based initialization, ten times higher bandwidth, and $\eta_v = 0.1$

Chapter 6

Semi-decentralized recommendation

In the previous chapter it was shown that a fully decentralized algorithm was able to create useful personalization. However, it is important to see how this algorithm compares to a similar centralized algorithm. It is expected that the centralized algorithm will have a performance edge over the decentralized algorithm. It would however be interesting to see if a compromise between centralization and decentralization can be made that achieves comparable performance to the centralized algorithm, while also keeping several benefits associated with decentralized algorithms.

Hypothesis: The centralized algorithm will have a performance edge over the decentralized algorithm. Increasing centralization gradually will increase performance gradually as well.

6.1 Comparison to centralized model

The first goal of this chapter is to see how a centralized algorithm compares to a decentralized algorithm. A centralized version of the decentralized algorithm developed in Section 5.5 is considered. A centralized data-based initialization is used. This initialization sets the user biases to the mean of their ratings. Each item bias is then set as the mean rating of the corresponding item minus the mean user bias of the users who has rated that item. The same learning rates are used as in the final decentralized algorithm in Section 5.5, i.e. $\eta_v = 0.1$ and $\eta_b = 0.01$. Since there is no other agents when training the centralized model, there will be no sending and merging of models. To make the centralized and decentralized algorithms comparable, the centralized model will only perform one epoch of training for each round of the decentralized algorithm.

The result of the experiment on all datasets can be seen on Figure 6.1. In the figure the centralized model converges a lot faster than the decentralized algo-

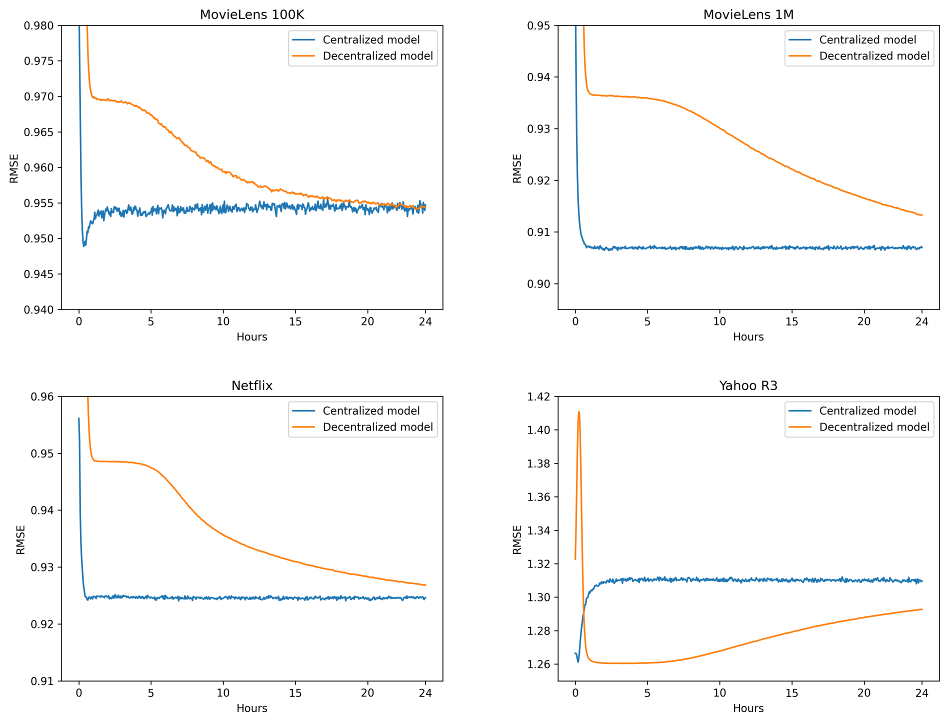


Figure 6.1: A comparison of the centralized algorithm to the decentralized algorithm. In this experiment both algorithms uses $\eta_v = 0.1$.

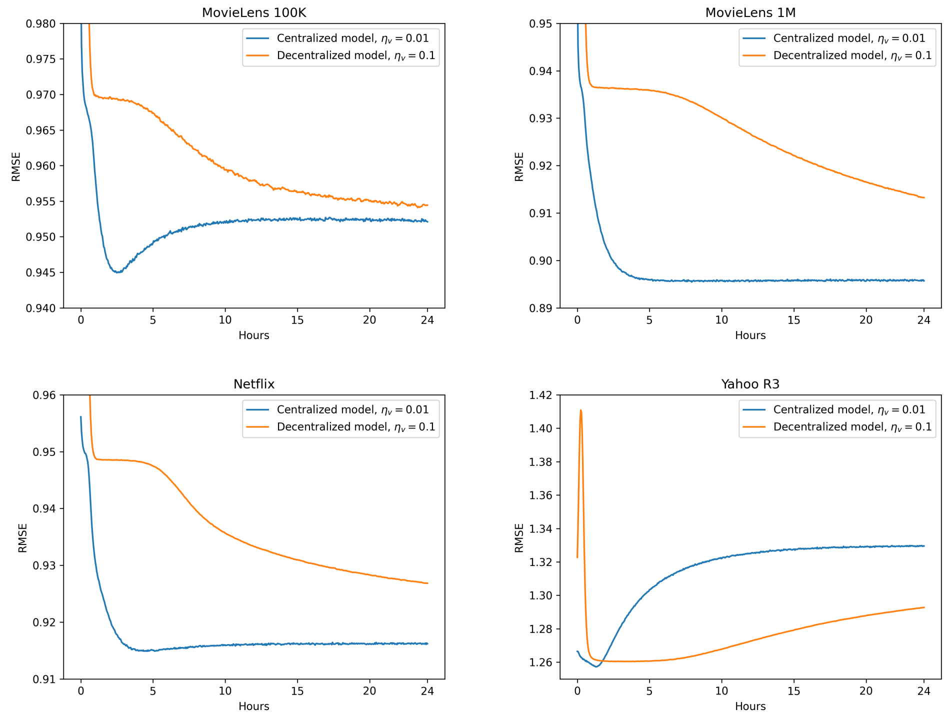


Figure 6.2: A comparison of the centralized algorithm to the decentralized algorithm. In this experiment the centralized algorithm uses $\eta_v = 0.01$, and the decentralized algorithm uses $\eta_v = 0.1$.

rithm. This is to its detriment for the Yahoo! dataset, as it overfits faster than the decentralized algorithm. The centralized algorithm did, in general, get a better score than the decentralized algorithm for the rest of the datasets. The decentralized algorithm did however get comparable results, and Figure 6.1 suggests that the decentralized algorithm could have gotten even closer in performance to the centralized algorithm if allowed more rounds of training.

One could argue that this comparison is not fair to the centralized model. The hyperparameters used for the centralized algorithm in the previous experiment was tuned for the decentralised algorithm. The fact that the centralized model achieves its best performance in the very beginning when training on the MovieLens 100K dataset before the loss increases again hints at the hyperparameters not being optimal. The centralized algorithm is therefore run again with the same parameters, except with $\eta = 0.01$.

The results of this experiment on all dataset can be seen on Figure 6.2. The performance of the centralized algorithm increased for all datasets except the Yahoo! dataset. This shows that the performance of the centralized algorithm can be improved by optimizing its hyperparameters. The hyperparameters that were good for the decentralized algorithm were not necessarily the best hyperparameters for the centralized model. It is also worth noting that the hyperparameters of the centralized algorithm can most likely be improved even more.

6.2 Shared-data clusters

The second goal of this chapter is to see if there is any merit to the idea of making a compromise between centralization and decentralization.

A semi-decentralized algorithm is therefore considered. In this algorithm, all users are divided into "shared-data clusters". In a shared-data cluster, all users in the cluster share their private data with an agent responsible for training the model, referred to as the updating agent. A prerequisite for this scenario is that the users trusts the updating agent. This could be because they belong to a close-knit community where trust is high. Another scenario that is arguable more likely is a scenario where the users in the cluster share their data with an organization. This organization then takes on the role of an updating agent.

6.2.1 Experiment setup

In this experiment, the users were divided as evenly as possible across several clusters. The amount of users per cluster, as well as the amount of clusters is determined by the *maximum cluster size* parameter together with the amount of users in total. The amount of clusters is the minimum amount of clusters needed to keep the size of the biggest cluster less than or equal to the maximum cluster size. The users are then as evenly divided between the clusters as possible. The amount of ratings per user was not taken into account, the total amount of ratings per cluster therefore varied.

The training process was very similar to the training process for regular Gossip Learning, see Section 4.1. The main difference is that the user agents do not directly participate in the training process. Instead, the designated updating agents for each cluster are the only agents who are directly participating. The updating agents have access to the data of all users in its cluster. Linear merge was be used in-between clusters, and the models were initialized using the centralized version of the data-based initialization on the rating data for users in the cluster. The centralized version of this initialization algorithm was introduced in Section 6.1. The learning rates used are $\eta_v = 0.1$ and $\eta_b = 0.01$.

6.2.2 Results

The results of the experiment can be seen on Figure 6.3. From the figure one can get the impression that a relatively small cluster size, i.e. a max cluster size of five or ten, is to be preferred, due to these sizes performing better than or equal to most other sizes for every dataset except for the Yahoo! dataset. However, the hyperparameters were optimized for the fully decentralized scenario, it is therefore not unlikely that algorithms with a higher degree of decentralization will be more

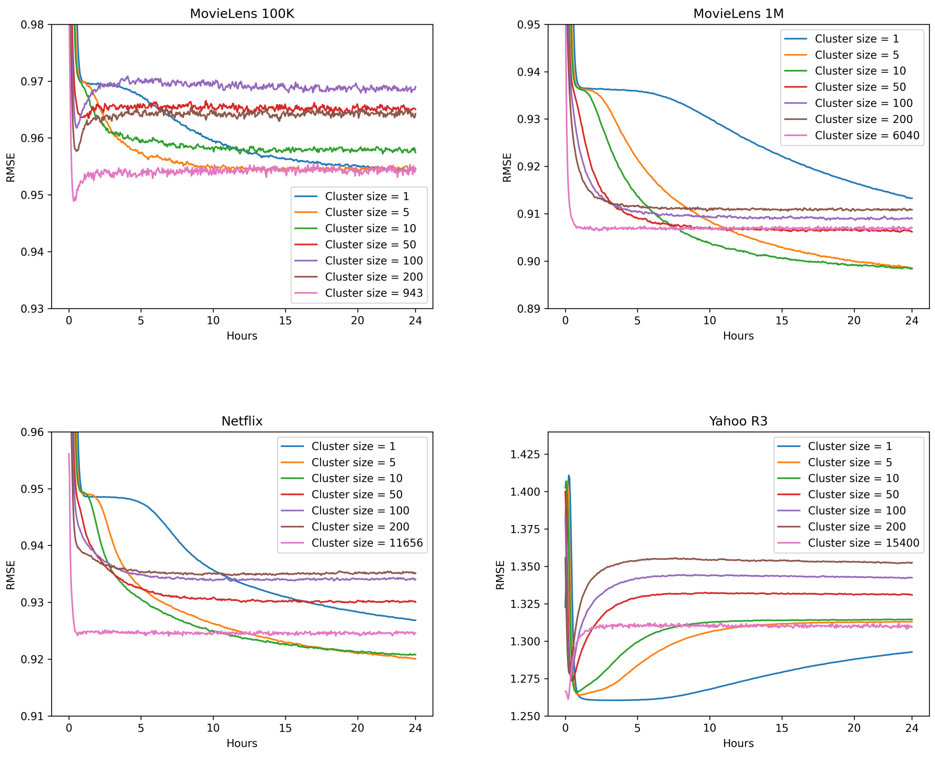


Figure 6.3: The RMSE over time for shared-data clusters with different maximum cluster sizes with $\eta_v = 0.1$ and $\eta_b = 0.01$. Please note that a cluster size of one is identical to fully decentralized GL, and a cluster size equal to all users is identical to the centralized model. The decentralized and centralized algorithm are shown in blue and pink respectively.

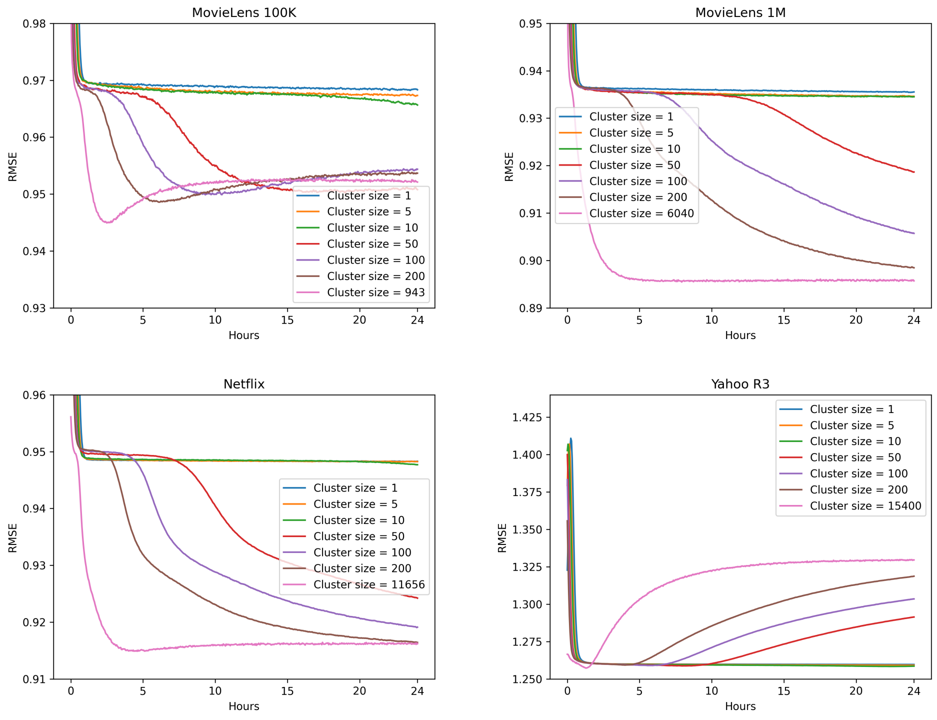


Figure 6.4: The RMSE over time for shared-data clusters with different maximum cluster sizes with $\eta_v = 0.01$ and $\eta_b = 0.01$. Please note that a cluster size of one is identical to fully decentralized GL, and a cluster size equal to all users is identical to the centralized model. The decentralized and centralized algorithm are shown in blue and pink respectively.

optimal with these hyperparameters. In Section 6.1 we learned that an $\eta_v = 0.01$ improved the performance of the centralized model. The previous experiment is therefore repeated using this learning rate for the latent vectors instead.

A lower η_v did lead to better performance in general when having bigger cluster sizes, except for the Yahoo! dataset. It is possible that these results can be improved even more by hyperparameter-tuning. Based on the results in this section it seems like an increase of centralization also increases the performance in general, as long as the correct hyperparameters are selected. The goal of these experiments are therefore fulfilled by showing that gradually increasing centralization can gradually increase performance.

6.3 Federated clusters

In the previous experiment, it was shown that users can make a privacy-related compromise to get better recommendations. This is not ideal – it would be better if there was a way for the users to get better recommendations without having to

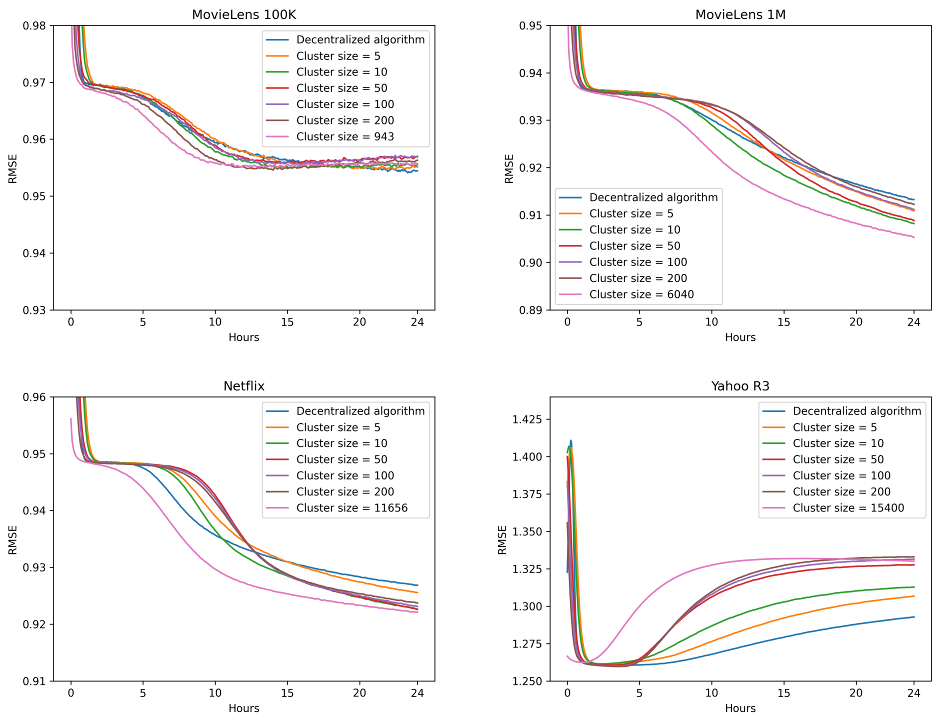


Figure 6.5: The RMSE over time for federated clusters with different maximum cluster sizes with $\eta_v = 0.1$ and $\eta_b = 0.01$.

give up their data. An algorithm inspired by both FL and GL is considered. The algorithm uses FL inside the cluster, while clusters exchange models like in GL.

The FL-experiment in Chapter 4 assumed that the federated learning master node is able to receive the gradients of all agents in the same amount of time needed for a full model transfer. A similar assumption has to be made in this experiment. One iteration of the algorithm is therefore equal to the time needed for two model transfers. The experiments in this chapter assume that one model transfer takes 172.8 seconds.

6.3.1 Experiment setup

The algorithm described uses a federated version of data-based initialization. Every user sets their user bias to the mean of their ratings. Each item bias is then calculated. Each user i calculates $r_{ij} - b_i$ for every rating they have. These values are then aggregated for all agents using the secure aggregation protocol for FL (see Section 2.2.4). The algorithm uses $\eta_v = 0.1$ and $\eta_b = 0.01$.

6.3.2 Results

See Figure 6.5 for the results of the experiment. The performance of the optimized fully decentralized algorithm introduced in Section 5.5 is included for comparison. The federated clusters performed worse than the shared data cluster in overall, especially if the hyperparameters of the shared data clusters is optimized.

Using federated clusters provided some benefit when comparing to the fully decentralized algorithm. The fully decentralized algorithm performs worse than the federated clusters for all cluster sizes for MovieLens 1M and Netflix. It performs slightly better for the smaller datasets MovieLens 100K and the Yahoo! dataset. Using FL gave better performance overall. The higher cluster sizes were generally better for MovieLens 1M and Netflix. For MovieLens 100K and the Yahoo! dataset it was the other way around, although very slightly for MovieLens 100K.

Using federated clusters with a cluster size equal to the number of users in the dataset is equal to using regular FL. An interesting observation is that the federated clusters consisting of all users in the dataset, shown in pink on Figure 6.5, outperforms GL for most datasets. This shows that FL can outperform GL when using the improvements developed for GL in Chapter 5.

6.4 Conclusion

In this chapter a fully decentralized algorithm was compared to a centralized version of the algorithm. The centralized version performed better, especially when the hyperparameters were tuned. A semi-decentralized version of the algorithm was also examined. A higher degree of centralization did in general give better performance. The performance was however sensitive to the hyperparameters, just like the fully centralized version.

An attempt at privacy-preservation was made for the semi-decentralized version of the algorithm. The introduction of privacy-preservation decreased performance, although, the semi-decentralized version with privacy preservation did seem to have some benefit over the fully decentralized algorithm for some scenarios.

Chapter 7

Discussion and future work

This chapter will answer the research questions asked in Section 1.2, and discuss what aspects could have been done differently.

7.1 Answers to research questions

7.1.1 RQ1 - What are the challenges associated with machine learning that do not violate privacy?

Machine Learning that does not violate privacy is Machine Learning where the information an agent can gain about the data of an individual user is minimal. Sending raw private data to any other agent is therefore out of the question, and as a result, traditional centralized methods can no longer be utilized, as they depend on having the full dataset available when training the model. In light of this, coming up with other methods was necessary in order to determine methods that are not dependent on having direct access to user data. In Chapter 3 two main approaches were tested.

The first approach tested exchanging locally trained models instead of raw data, and then aggregating the results of all of these weak models using ensemble methods. Regular Majority Voting and WMA was used. Although the individual weak models were extremely inaccurate, it was possible to achieve a significantly higher accuracy by using ensemble methods to aggregate the results of the models. WMA achieved the best results. The achieved accuracy was however lower than the centralized strong model in most scenarios, although WMA did achieve a better test-accuracy than the centralized strong model when the test-distribution changed dramatically over time.

Other than accuracy, both Majority Voting and WMA had some other challenges. The first major challenge is that the output of all models have to be computed before the compound prediction can be made. This is inefficient compared to the centralized model, and it scales poorly when the number of participants increases. It would have been interesting if methods to alleviate this problem was investigated. For example, one could choose to only calculate the output of the top k models with the highest weight. This could be especially useful for changing test distributions.

Another challenge is that an agent was dependent on receiving the models trained by all other agents. The amount of model transfers therefore increased quadratically with the size of the network. A solution would be to make every agent only send a fixed amount of models to a random selection of agents. The performance of the ensemble of each user would however most likely decrease as it would be smaller, but it would have been interesting to see by how much.

The other approach that was tested in Chapter 3 was Federated Learning (FL). When using this approach a strong model is trained in a collaborative fashion by each agent exchanging gradients with a federated aggregation-server. This approach did manage to achieve comparable performance to the traditional centralized model if not too many users send gradients to the federated aggregation server at a time. Only one user sending gradients at the same time will render secure aggregation (see Section 2.2.4) useless as there would be no other users to securely aggregate gradients with. Having a higher amount of users participate in each iteration of the algorithm did lead to a lower accuracy of the model given the same amount of training epochs performed by each user. Having a smaller amount of users involved in each iteration of the algorithm also makes the overall training process take more time, as each update is then less optimal, and more iterations are needed.

FL does however suffer from another problem that was not examined in this chapter. The training data used for FL is typically not Independent and Identically Distributed (IID) across the devices. This is due to all data on one device being generated from the same source, i.e. the behavior of one user. Data being non-IID can decrease performance by a significant margin [41]. The data was however non-IID in Chapter 4, but a comparison was not made to a scenario were the data was IID across the devices.

7.1.2 RQ2 - What are the advantages and disadvantages of a fully decentralized user-to-user recommender system?

FL is dependent on having a central coordinator, and can therefore not be considered fully decentralized. The users are dependent on an organization to maintain the recommender system. GL on the other hand is fully decentralized. A comparison of FL and GL was therefore made in Chapter 4. In this comparison GL came out on top when performing a similar experiment to the experiment made by Hegedűs *et al.* in [30]. The GL results of Hegedűs *et al.* were recreated, but the results for FL were worse than in their experiment.

Some possible areas of improvement were identified in the GL-algorithm described in the original experiment. Chapter 5 was therefore used to explore ways to improve the performance of GL. Several strategies were found that improved the results of the algorithm compared to the original experiment.

The first type of developed strategies focused on different ways of merging models. Several types were tested, but merging models by exponentially weighting them based on parameter age improved the results of the original experiment the most. Euler's number was used as the base number for the exponential, but it would have been interesting to try a different base number. A combination of a polynomial and an exponential would also have been an interesting experiment.

The second strategy developed was an initialization strategy that used the local data to calculate initial values for the biases. This strategy significantly improved the performance of the algorithm compared to the original experiment. Interestingly the exponential merge strategy did not work well together with the initialization strategy developed. Other strategies that could have been looked into was model-compression techniques, as these did improve the performance of significantly in the original experiment.

After some analysis it was also discovered that the algorithm used in the original experiment did not offer any useful personalization. However, by letting the algorithm train for longer, and by increasing the learning rate of the latent vectors, the latent vectors learned useful personalization. This lead to another significant performance increase.

In Chapter 6 these methods were also applied to FL, which significantly increased the performance. With these algorithms applied, the performance of FL surpassed GL (see Section 6.3).

The disadvantage of GL therefore seem to be that a better RMSE can be achieved when using FL, given that a more optimal implementation of FL is used, like the one described in Section 6.5. However, all experiments using FL did assume that FL aggregation sever had an infinitely high download speed, i.e. it would never be the bottleneck of the algorithm. This is an assumption that favours FL, and this might not be true in real world applications. The advantage of GL therefore seems to be that it is not dependent on having on any central entity. It is therefore more robust and offers a cheaper scalability. Not being dependent on an organization maintaining this central server is also an advantage for the users, as they then have more leverage in deciding how their recommender system should be designed.

7.1.3 RQ3 - What are the benefits of semi-decentralized learning?

The performance of the decentralized algorithm tested in Chapter 5 was compared to a centralized version of the algorithm. After some hyperparameter-tuning, it became apparent that centralized learning still has a significant performance edge compared to decentralized learning.

Semi-decentralized learning was therefore investigated to see if making a gradual centralization-compromise could increase performance. The first version of the semi-decentralized learning algorithm also required a privacy-compromise. The experiments indicated that increasing centralization improved performance, as long as hyperparameters were tuned properly. The graphs suggested that further hyperparameter tuning could have increased the performance even more, it would therefore be interesting to tune the hyperparameters more thoroughly to test this.

An approach were users would not have to make a compromise on privacy was also investigated. This approach did however significantly decrease the performance when comparing to methods without data privacy. It did however seem like higher centralization provided slightly increased performance when compared to the fully decentralized method.

The benefit of semi-decentralized learning is an increase in performance when comparing to GL. Higher centralization increases performance in general, especially if privacy is not maintained. Semi-decentralized learning also offers a higher degree of decentralization than a centralized solution. However, semi-decentralized is less decentralized than a fully decentralized solution, and it has a lower performance than a centralized solution. It should therefore only be considered when a compromise between decentralization and performance is desired.

7.2 Future work

The different decentralized and semi-decentralized algorithms show promise, but there is room for improvement. The following section suggests ideas that are worth examining further.

7.2.1 Try different recommender system models

Only one recommender system model was tried, i.e. matrix factorization with biases. Other models might be more optimal and could therefore help provide better recommendations. A Factorization Machine [42] is an example of a model that would be interesting to look further into. It can be trained using SGD and is also able to take into account other sources of information about the item and the user. The model size is however bigger than matrix factorization with biases, how much depends on the amount of extra information about the user and the item it should take into account

7.2.2 Combination of ensemble methods and GL

Ensemble methods could be combined with GL. Each agent could save a fixed amount of models that performed well on local data. WMA could then be used for inference. Periodically these models can be recombined and updated. Good models are transmitted to the rest of the network. This approach would automatically filters out bad models, and could be employed as a defense against spam by malicious agents.

7.2.3 Try different optimization algorithm than plain SGD

The model was very sensitive to changes in the learning rate. It is therefore likely that a different optimization algorithm than plain SGD would provide some benefit. Adam [43] is an example of an optimization algorithm that could have been useful, as it performs well on sparse models.

7.2.4 Implement compression techniques

The compression techniques described by Hegedűs *et al.* in [30] improves performance. Implementing compression as well as the techniques introduced in this thesis could therefore potentially achieve an even better performance.

7.2.5 Implement differential privacy

Differential privacy is paramount for any of the decentralized algorithms presented in this thesis to not pose a privacy risk. It should therefore be further investigated if differential privacy measures, like the measures described in [24], could lead to decreased performance.

7.2.6 Implement secure aggregation

Secure aggregation is necessary for the privacy of the decentralized algorithms to be preserved. The protocol described in [17] can be used for FL, while the protocol described in [19] can be used for GL. It is important to further investigate if implementing these techniques decreases the performance of the algorithms.

Bibliography

[1] D. Shepardson, “U.s. senate panel to seek answers from facebook about data access report,” *Reuters*, Jun. 2018. [Online]. Available: `https://www.reuters.com/article/us-facebook-privacy-idUSKCN1J01HV`.

[2] C. A. Gomez-Uribe and N. Hunt, “The netflix recommender system: Algorithms, business value, and innovation,” *ACM Trans. Manage. Inf. Syst.*, vol. 6, no. 4, Dec. 2016, ISSN: 2158-656X. DOI: `10.1145/2843948`. [Online]. Available: `https://doi.org/10.1145/2843948`.

[3] X. Amatriain and J. Basilico, *Netflix recommendations: Beyond the 5 stars (part 1)*, Apr. 2012. [Online]. Available: `https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429`.

[4] “Facebook faces mass legal action over data leak,” *BBC News*, Apr. 2021. [Online]. Available: `https://www.bbc.com/news/technology-56772772`.

[5] G. J. Dance, M. LaForgia, and N. Confessore, *As Facebook Raised a Privacy Wall, It Carved an Opening for Tech Giants*, Dec. 2018. [Online]. Available: `https://www.nytimes.com/2018/12/18/technology/facebook-privacy.html`.

[6] European Parliament and Council of the European Union. (Apr. 27, 2016). “Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation) (text with EEA relevance),” [Online]. Available: `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679&from=EN` (visited on 06/29/2021).

[7] J. Creswell, “How amazon steers shoppers to its own products,” *The New York Times*, Jun. 2018. [Online]. Available: `https://www.nytimes.com/2018/06/23/business/amazon-the-brand-buster.html`.

[8] J. D’Urso, “Why are politicians getting ’schooled’ and ’destroyed’?” *BBC News*, Aug. 2019. [Online]. Available: `https://www.bbc.com/news/49165846`.

[9] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, A. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng, “Large scale distributed deep networks,” *Advances in neural information processing systems*, Oct. 2012.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[11] N. Littlestone and M. Warmuth, “The weighted majority algorithm,” *Information and Computation*, vol. 108, no. 2, pp. 212–261, 1994, ISSN: 0890-5401. DOI: https://doi.org/10.1006/inco.1994.1009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540184710091.

[12] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, *Communication-efficient learning of deep networks from decentralized data*, 2016. arXiv: 1602.05629 [cs.LG].

[13] I. Hegedűs, G. Danner, and M. Jelasity, “Gossip learning as a decentralized alternative to federated learning,” in *Distributed Applications and Interoperable Systems*, J. Pereira and L. Ricci, Eds., Cham: Springer International Publishing, 2019, pp. 74–90, ISBN: 978-3-030-22496-7.

[14] R. Ormándi, I. Hegedűs, and M. Jelasity, “Gossip learning with linear models on fully distributed data,” *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 556–571, May 2012, ISSN: 1532-0626. DOI: 10.1002/cpe.2858. [Online]. Available: http://dx.doi.org/10.1002/cpe.2858.

[15] I. Hegedűs, G. Danner, and M. Jelasity, “Decentralized learning works: An empirical comparison of gossip learning and federated learning,” *Journal of Parallel and Distributed Computing*, vol. 148, pp. 109–124, 2021, ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2020.10.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731520303890.

[16] L. Zhu, Z. Liu, and S. Han, “Deep leakage from gradients,” *CoRR*, vol. abs/1906.08935, 2019. arXiv: 1906.08935. [Online]. Available: http://arxiv.org/abs/1906.08935.

[17] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” Oct. 2017, pp. 1175–1191. DOI: 10.1145/3133956.3133982.

[18] A. Shamir, “How to share a secret,” *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979, ISSN: 0001-0782. DOI: 10.1145/359168.359176. [Online]. Available: https://doi.org/10.1145/359168.359176.

[19] G. Danner, Á. Berta, I. Hegedüs, and M. Jelasity, “Robust fully distributed minibatch gradient descent with privacy preservation,” *Secur. Commun. Networks*, vol. 2018, 6728020:1–6728020:15, 2018.

[20] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” vol. 5, May 1999, pp. 223–238, ISBN: 978-3-540-65889-4. DOI: `10.1007/3-540-48910-X_16`.

[21] C. Dwork and M. Naor, “On the difficulties of disclosure prevention in statistical databases or the case for differential privacy,” *Journal of Privacy and Confidentiality*, vol. 2, Sep. 2010. DOI: `10.29012/jpc.v2i1.585`.

[22] A. Wood, M. Altman, A. Bembenek, M. Bun, M. Gaboardi, J. Honaker, K. Nissim, D. O’Brien, T. Steinke, and S. Vadhan, “Differential privacy: A primer for a non-technical audience,” *SSRN Electronic Journal*, Jan. 2018. DOI: `10.2139/ssrn.3338027`.

[23] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, “The secret sharer: Evaluating and testing unintended memorization in neural networks,” in *USENIX Security Symposium*, 2019.

[24] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, *Learning differentially private recurrent language models*, 2018. arXiv: `1710.06963 [cs.LG]`.

[25] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, *Recommender Systems Handbook*, 1st. Berlin, Heidelberg: Springer-Verlag, 2010, ISBN: 0387858199.

[26] J. S. Breese, D. Heckerman, and C. Kadie, “Empirical analysis of predictive algorithms for collaborative filtering,” in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI’98, Madison, Wisconsin: Morgan Kaufmann Publishers Inc., 1998, pp. 43–52, ISBN: 155860555X.

[27] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30–37, 2009. DOI: `10.1109/MC.2009.263`.

[28] M. Ammad-ud-din, E. Ivannikova, S. A. Khan, W. Oyomno, Q. Fu, K. E. Tan, and A. Flanagan, *Federated collaborative filtering for privacy-preserving personalized recommendation system*, 2019. arXiv: `1901.09888 [cs.IR]`.

[29] A. Flanagan, W. Oyomno, A. Grigorievskiy, K. E. Tan, S. A. Khan, and M. Ammad-Ud-Din, “Federated multi-view matrix factorization for personalized recommendations,” *Lecture Notes in Computer Science*, pp. 324–347, 2021, ISSN: 1611-3349. DOI: `10.1007/978-3-030-67661-2_20`. [Online]. Available: `http://dx.doi.org/10.1007/978-3-030-67661-2_20`.

[30] I. Hegedűs, G. Danner, and M. Jelasity, “Decentralized recommendation based on matrix factorization: A comparison of gossip and federated learning,” in *Machine Learning and Knowledge Discovery in Databases*, P. Cellier and K. Driessens, Eds., Cham: Springer International Publishing, 2020, pp. 317–332, ISBN: 978-3-030-43823-4.

[31] H. Robbins and S. Monro, “A Stochastic Approximation Method,” *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951. DOI: `10.1214/aoms/1177729586`. [Online]. Available: `https://doi.org/10.1214/aoms/1177729586`.

[32] T. Chai and R. Draxler, “Root mean square error (rmse) or mean absolute error (mae),” *Geoscientific Model Development Discussions*, vol. 7, pp. 1525–1534, 2014.

[33] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. V. Steen, and H. Sips, “Tribler: A social-based peer-to-peer system,” *Concurrency and Computation: Practice and Experience*, vol. 20, 2006.

[34] F. Niu, B. Recht, C. Re, and S. J. Wright, *Hogwild!: A lock-free approach to parallelizing stochastic gradient descent*, 2011. arXiv: `1106.5730 [math.OC]`.

[35] B. Recht and C. Ré, “Parallel stochastic gradient algorithms for large-scale matrix completion,” *Mathematical Programming Computation*, vol. 5, pp. 201–226, 2013.

[36] C. Teflioudi, F. Makari, and R. Gemulla, “Distributed matrix completion,” *2012 IEEE 12th International Conference on Data Mining*, pp. 655–664, 2012.

[37] H.-F. Yu, C.-J. Hsieh, S. Si, and I. Dhillon, “Parallel matrix factorization for recommender systems,” *Knowledge and Information Systems*, vol. 41, pp. 793–819, 2013.

[38] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: `http://yann.lecun.com/exdb/mnist/`.

[39] L. Giaretta and Š. Girdzijauskas, “Gossip learning: Off the beaten path,” in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 1117–1124. DOI: `10.1109/BigData47090.2019.9006216`.

[40] N. Hug, “Surprise: A python library for recommender systems,” *Journal of Open Source Software*, vol. 5, no. 52, p. 2174, 2020. DOI: `10.21105/joss.02174`. [Online]. Available: `https://doi.org/10.21105/joss.02174`.

[41] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, *Federated learning with non-iid data*, 2018. arXiv: `1806.00582 [cs.LG]`.

[42] S. Rendle, “Factorization machines,” in *2010 IEEE International Conference on Data Mining*, 2010, pp. 995–1000. DOI: `10.1109/ICDM.2010.127`.

[43] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: `1412.6980 [cs.LG]`.