An Algorithmic Framework for Decentralised Matrix Factorisation

Erika Duriakova[1]✉ Wěipéng Huáng[1]✉ Elias Tragos[1]
Aonghus Lawlor[1] Barry Smyth[1] James Geraci[2]
Neil Hurley[1]

[1]Insight Centre for Data Analytics, University College Dublin
[2]Samsung Electronics Co., Ltd.

Abstract

We propose a framework for fully decentralised machine learning and apply it to latent factor models for top-N recommendation. The training data in a decentralised learning setting is distributed across multiple agents, who jointly optimise a common global objective function (the loss function). Here, in contrast to the client-server architecture of federated learning, the agents communicate directly, maintaining and updating their own model parameters, without central aggregation and without sharing their own data. This framework involves two key contributions. Firstly, we propose a method to extend a global loss function to a distributed loss function over the distributed parameters of the decentralised system; secondly, we show how this distributed loss function can be optimised using an algorithm that operates in two phases. In the learning phase, a large number of steps of local learning are carried out by each agent without communication. In a following sharing phase, neighbouring agents exchange messages that enable a batch update of local parameters. Thus, unlike other decentralised algorithms that require some inter-agent communication after one (or a few) model updates, our algorithm significantly reduces the number of messages that need to be exchanged during learning. We prove the convergence of our framework and demonstrate its effectiveness using both the Weighted Matrix Factorisation and Bayesian Personalised Ranking latent factor recommender models. We demonstrate empirically the performance of our approach on a number of different recommender system datasets.

Recommender Systems Distributed Learning Decentralised Matrix Factorisation Latent Factor Models Matrix Factorisation Communication Efficiency Convergence Proof.

1 Introduction

There is much interest recently in the development of machine learning (ML) algorithms that can be carried out on distributed architectures. The reasons that drive this shift are related with (i) the increased server maintenance costs for handling big data, (ii) the attempt to avoid centralised single points of failure (and attack) and (iii) the greater awareness and concern about data

privacy (storing user data on central servers) that stems from legislation such as the European Union's General Data Protection Regulation (GDPR). The shift to distributed architectures poses a challenge for machine learning systems that have been traditionally centralised until now, requiring user data to be gathered and stored on a central computational server that runs the learning algorithm. In distributed architectures, ML algorithms should be developed to run on user devices without the need for those devices to share their private data.

While the field of distributed optimisation has a long history (see, e.g. [22]), the recent surge in research on this topic can be traced to [14], in which *federated* optimisation or learning is distinguished from distributed ML, by emphasising that a federated system is expected to run outside a data-centre, on very large number of user devices, typically leading to very highly unbalanced and not identically distributed data across these devices. Since this work, a number of papers have proposed federated learning architectures in different ML contexts. The vast majority of these works have focused on client-server architectures. Such systems still rely on a central server, which, rather than storing the user data itself, instead coordinates the learning process among the user devices, receiving local model updates from the devices and aggregating these into a global model. Nevertheless, as pointed out in [11] for example, such a central server is not always desirable, as it represents a single point-of-failure and may become a bottleneck when the number of clients is very large. Hence, in this paper, we focus on fully decentralised learning.

We propose a general algorithmic framework and apply it to latent factor models for recommender systems.

Recommender Systems (RS) learn from user interaction data with the products or services that they recommend, which we will refer to generally as *items*. RS are commonplace nowadays in many e-commerce contexts, providing personalised recommendations of books, movies, news and so on. The interactions on which recommendations are learned may correspond to explicit ratings of items, or implicit actions, such as clicks or views. The recommendation problem is typically formulated as a top N problem, in which, given a budget of N recommendations, the system predicts the N items that are most likely to satisfy a user's need. In common with other machine learning methods, traditional RS models are learned on centralised systems, where interaction data from all users is gathered into a single database. Among many recommendation models, latent factor models, in which user- and item-embeddings are learned and used to make the predictions, have proven among the best in terms of accuracy.

We develop a fully-decentralised latent factor model for the top-N recommendation task, using our proposed framework and empirically test it on a number of different RS datasets. While there exists a few other works that have developed decentralised latent factor models for recommendation, our work may be distinguished from the state-of-the-art in so far as we provide a solid foundation for our optimisation algorithm, by proposing a distributed loss function to properly control learning across devices; and develop a decentralised optimisation algorithm, that manages the communication between neighbours in a way that reduces the number of messages required, in comparison with the state-of-the-art. Moreover, we provide a proof of the convergence of the framework, for certain classes of loss function. In summary, our contributions are as follows:

- We propose a framework for fully decentralised learning, consisting of a general form for a distributed loss function and an optimisation algorithm to minimise the loss;
- We extend beyond the common *gossip* approach of parameter averaging, by introducing a "*cross-loss*" term in the distributed loss function to accelerate convergence; moreover, our decentralised approach avoids the communication of private parameters;
- We provide a theoretical analysis on the convergence of the algorithmic framework. Unlike some other work, such as [2], we do not rely on convexity of the loss function but only require that the loss function is *L-smooth*;
- We apply the framework to the decentralised learning of latent factor top-N recommendation models;
- We empirically show the convergence of our method and compare with state-of-the-art on a number of different recommender datasets.

The remainder of the paper is structured as follows. In Section 2, we put our work in context with the state-of-the-art. In Section 3 we provide a background on latent factor models that are used within our framework. In Section 4, we introduce our proposed framework for decentralised learning. In Section 5 we present the evaluation results of our framework, while in Section 6 we provide the final remarks on the conclusions of the work. The theoretical convergence proof of our algorithm is provided in the supplemental material.

2 Related Work

Federated Learning (FL) systems [14, 16] are receiving a lot of attention recently. The standard FL architecture is composed of two main entities: the client and the server. The client encapsulates the private user data, it applies local updates to the global ML model during training, and executes the inference, i.e. the classification or recommendation, once the model is learned. The server initialises and coordinates the training of the global model, aggregating updates provided by clients and distributing the latest model back to clients. In this architecture, a powerful and reliable central server is a key part of an FL system. In [11], the case for a fully decentralised learning model is made. In this scenario, clients communicate directly in a peer-to-peer manner, each learning a local model without central aggregation to a global model.

Seminal work on distributed optimisation was carried out by Tsitsiklis et al. [22], in the 1980's. This work points out that synchronous algorithms, by which they mean algorithms in which processors communicate their partial results after each update, have certain drawbacks, including that they may introduce bottlenecks into the speed of the algorithm and may require far more communications than are actually necessary. They develop a number of asynchronous algorithms, where processors do not need to communicate after every update. Given a set of model parameters θ_ℓ, denote by $\theta_{u\ell}^{(t)}$ the value of the parameter on agent u at time-step t. They consider algorithms that apply updates

of the general form:

$$\theta_{u\ell}^{(t)} = \sum_{i=1}^{n} w_{uv}\theta_{v\ell}^{(t_v)} + \gamma_u^{(t)} s_{u\ell}^{(t)} \tag{1}$$

where n is the total number of agents, $t_v \leq t$ is the time at which the value received by u was computed on v, w_{uv} are non-negative weights such that $\sum_v w_{uv} = 1$, $s_{u\ell}^{(t)}$ is the update step computed from the loss function and $\gamma_u^{(t)}$ is the learning rate. Thus, these algorithms reach consensus over time, by applying a weighted average of their parameter updates with parameter values received from other processors. As described later, we will take a different approach to developing a distributed algorithm, by firstly defining an objective over the distributed parameters of the model $\theta_{u\ell}$ and then developing an algorithm in which the updates are computed as stochastic gradients of this distributed objective. A good survey of decentralised optimisation is provided in [18].

More recently, decentralised algorithms are often referred to as *gossip* algorithms [12]. A number of such decentralised ML algorithms have been proposed. For instance, Lian et al. [15] and Tang et al. [21] propose a parallel stochastic gradient algorithm. Also, Nedic [17] proposes a sub-gradient algorithm which essentially follows the update rule of Eqn. (1). The convergence of this work was later studied by Yuan et al. [24]. Another work [7] differs only in so far as the weight matrix is not stochastic and hence a normalising factor is also communicated between processors. In [4], a gossip algorithm for a pairwise loss function (an example of such being the BPR loss function discussed later), is proposed, which also is based on the averaging of parameters exchanged between processors, as well as the exchange of some training data. Other work, such as that of [13] has focused on minimising communication in gossip algorithms by applying sparsification and quantisation to compress the messages between processors. The work of [23] has some similarities to the framework that we propose, in so far as a loss function is defined on each agent with parameters shared with neighbouring agents, but it is solved by imposing equality constraints on the shared parameters. Also, Bellet et al. [2] design a loss function including a constraint on the norm of the difference between parameters across different agents, but use a coordinate descent algorithm, in which communications with neighbouring processors are required on each update step. A critical contribution of that work is the use of differential privacy to add security to the updates. Another gossip-based algorithm [1] again relies on updates based on Eqn. (1), but considers the overlap of computation and communication, by implementing non-blocking communication and applying the update from neighbours every $\tau \geq 1$ updates, rather than every update.

Now, let us focus in particular on decentralised algorithms in RS, there has been some research, such as [23] and [19] on decentralised user-based and item-based kNN algorithms. Client-server FL architectures for the latent factor recommendation model are proposed in [5, 10]. The only examples of a fully decentralised latent factor recommendation models in the state-of-the-art are that of [3, 8, 6]. While each of these develops a latent-factor model, the works [8, 6] focus on the rating prediction problem—that of predicting the rating a user would give to an item—rather than the top-N recommendation problem. Further, although Chen et al. [3] solve top-N recommendation problem, the model communicates the partial results at each update which, as discussed earlier, is

a major drawback in real-world scenarios.

3 Latent Factor Models for Top-N Recommendation

We illustrate our decentralised learning algorithm using two latent factor RS algorithms, namely the *weighted matrix factorisation* (WMF) model of [9] and the *Bayesian Personalised Ranking* (BPR) model of [20]. Each of these models is designed to yield top-N recommendations given a database of implicit user-item interactions.

Notation. Let U represent a set of users, such that $|U| = n$ and I represent the catalogue of items, such that $|I| = m$. The training set consists of a set of implicit interactions $\{r_{ui} \mid u \in U, i \in I\}$. We will write R_u to represent the set of items rated by user u and $\mathbf{r}_u$ to represent the m-dimensional vector of ratings associated with u. A latent factor model consists of low-dimensional user- and item-embeddings[1]. In particular, for a given dimension $k \ll n, m$, let P be a $n \times k$ matrix of user factors, and write $\mathbf{p}_u$ for the k-dimensional vector corresponding to each row of P; and similarly let Q be a $m \times k$ matrix of item-factors, and write $\mathbf{q}_i$ for the i^{th} row of Q. A top-N prediction for user u is formed by computing the m-dimensional vector $\hat{\mathbf{r}} = \mathrm{Q}\mathbf{p}_u$, sorting the components of $\hat{\mathbf{r}}$ in decreasing order and recommending the top N corresponding items. For any $S \subseteq I$, we will use $\mathbf{q}_S$, to represent the components $\{\mathbf{q}_i \mid i \in S\}$.

The two models differ according to the loss functions used to determine the parameters $\{\mathrm{P}, \mathrm{Q}\}$. In general, the loss function for latent factor models can be written as

$$\mathcal{L}(\mathrm{P}, \mathrm{Q}) = \sum_{u \in U} \mathcal{L}_u(\mathbf{p}_u, \mathrm{Q}, \mathbf{r}_u)\,.$$

In particular, for WMF, setting $\tilde{r}_{ui} = 1$ for $r_{ui} > 0$ and $\tilde{r}_{ui} = 0$ for $r_{ui} = 0$, then

$$\mathcal{L}^{\text{WMF}}(\mathrm{P}, \mathrm{Q}) = \sum_{u} \sum_{i \in I} c_{ui}(\mathbf{p}_u^\top \mathbf{q}_i - \tilde{r}_{ui})^2$$

where, for some input parameter α, $c_{ui} = (1 + \alpha r_{ui})$ represents the confidence.

The BPR loss models the probability that an item i is preferred to an item j, using a sigmoid function of the difference between their corresponding latent factor products and represents the loss as the negation of the sum of the log-probabilities, resulting in

$$\mathcal{L}^{\text{BPR}}(\mathrm{P}, \mathrm{Q}) = -\sum_{u} \frac{1}{|R_u|} \sum_{i \in R_u} \frac{1}{m - |R_u|} \sum_{j \notin R_u} \log \sigma(-x_{uij}),$$
$$\text{s.t. } x_{uij} = \mathbf{p}_u^\top (\mathbf{q}_i - \mathbf{q}_j),$$

where σ is the sigmoidal function s.t. $\sigma(x) = (1 + e^x)^{-1}$. To mitigate against overfitting, these loss functions are typically regularised by adding the terms:

$$\lambda_p \sum_u \|\mathbf{p}_u\|^2 + \lambda_q \sum_i \|\mathbf{q}_i\|^2$$

[1] We ignore bias terms for simplicity, though these can easily be incorporated into the framework

for weights $\lambda_p \geq 0$ and $\lambda_q \geq 0$.

One approach to optimise these loss functions is to apply a stochastic gradient descent (SGD) algorithm (although the WMF algorithm is optimised in [9] using alternative least squares). Thus we can decompose the loss into summands:

$$\mathcal{L}^{\text{WMF}} = \sum_u \sum_i \mathcal{L}^{\text{WMF}}_{ui}; \quad \mathcal{L}^{\text{BPR}} = \sum_u \sum_{ij} \mathcal{L}^{\text{BPR}}_{uij}$$

and, sampling at random a pair (u, i) for WMF[2] (or resp. a triple (u, i, j) for BPR), update the parameters $\theta \in \{\mathbf{p}_u, \mathbf{q}_i\}$ by

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L}^{\text{WMF}}_{ui} \quad \text{or} \quad \theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L}^{\text{BPR}}_{uij} \,,$$

for a learning rate η.

We extend this standard SGD approach, which requires the data to be collected into a central repository, to a fully-decentralised algorithm, in the next section.

4 Framework for Decentralised Learning

Consider a set of n agents, co-operating on an ML optimisation task. Let $G(V, E)$ be a communication graph, where V is the set of agents, and an edge $(u, v) \in E$, if agent u is able to communicate with agent v. We will write $N_u(G)$ for the set of neighbours of agent u in this graph. Each agent maintains its own data x_u, which is *not shared* with other agents. For loss functions based on empirical risk, it is possible to write

$$\mathcal{L}(\Theta) = \sum_u \mathcal{L}_u(\Theta, x_u)$$

where $\mathcal{L}$ only depends on x_u through $\mathcal{L}_u$. In a fully decentralised setting, each agent u maintains and updates its own set of parameters Θ_u. In the case of the RS models we are considering here, unlike some other domains, some parameters are too private for sharing as they are strongly correlated to user data. Hence, unlike typical gossip models which share all parameters, we consider different privacy risks associated with different parameters. We write $\Theta_u = [\Theta^p_u, \Theta^s_u]$, where Θ^p_u are private parameters and Θ^s_u are sharable.

Decentralised algorithms generally require agents to communicate to reach a consensus such that at the end of learning, $\forall u\ \Theta_u = \Theta^*$ and $\mathcal{L}(\Theta^*)$ is minimised. Commonly, this consensus is reached via gossip updates such as shown in Eqn. (1), which do not distinguish the privacy characteristics of the parameter set. Instead, we propose to extend the global objective over the full set of distributed parameters. The loss function must be constructed in such a way that local parameters Θ_u are affected by a loss associated with data x_v, where $v \neq u$; otherwise, a model that generalises to unseen data will not be possible.

[2]SGD schemes for WMF typically sample negative items ($i \notin R_u$) at a different rate to positive samples. In this case, the stochastic gradients are unbiased gradient estimators of a loss function where the confidence term is $c_{ui} = \pi_i(1 + \alpha r_{ui})$, where π_i is the probability of sampling item i.

Considering privacy risks, we wish to consider loss functions for which it is possible to develop a learning algorithm where it is allowed to share parameters Θ_u^s or functions of Θ_u^s or Θ_u^p, but not directly Θ_u^p. With these considerations, we will consider loss functions of the general form:

$$\mathcal{L} = \frac{1}{n}\left(\sum_u \mathcal{L}_u(\Theta_u, x_u) + \sum_u \sum_{v \in N_u(G)} \mathcal{L}_{uv}(\Theta_u, \Theta_v)\right)$$

where $\mathcal{L}_{uv}$ depends on parameters that are local to u or local to v. In particular, we construct $\mathcal{L}_{uv}$ as

$$\mathcal{L}_{uv}(\Theta_u, \Theta_v) = w_{uv}\left(\lambda_r \|\Theta_u^s - \Theta_v^s\|^2 + \lambda_l \mathcal{L}_u([\Theta_u^p, \Theta_v^s], x_u)\right) ,$$

where $\sum_v w_{uv} = 1$ are non-negative weights and $\lambda_l \geq 0$, $\lambda_r \geq 0$ control the contribution that each term makes to the loss-function. The first term above is used, similarly to other decentralised algorithms, to push the optimisation towards a consensus among the shared distributed parameters. The second term, which we refer to as a "cross-loss" term as it mixes the u and v parameter set, ensures that local parameters update in a way that takes account of the data in their neighbourhood. Note that, as we will see, optimisation can still proceed without direct sharing of the private parameters or private training data and the use of this cross-loss term can significantly accelerate learning.

We are concerned about the efficiency of the learning algorithm, in a context where communication between agents may be expensive, in comparison to local computation. Hence, we are interested in developing a learning algorithm in which the number of communications is minimised. We note that in some state-of-the-art decentralised learning algorithms such as [3] and [6], a communication takes place in *every* update step and hence this sort of algorithm is not appropriate for the setting that we have in mind.

We introduce an algorithmic framework in which the algorithm proceeds in two bulk steps of computation followed by communication, rather than interweaving message-passing with model updating. In particular, in the first *learn* phase, learning on each agent focuses on the L_u term, and carries out a large number of local updates to reduce L_u. Following this, in the *share* phase, learning focuses on the functions L_{uv}, which require communication between the agents u and v. We construct our algorithm so that *sharing* occurs infrequently in comparison to local *learning* and many updates are carried out in the learn phase, before sharing. Our focus is to complete the optimisation with as few communications as possible.

4.1 Decentralised Latent Factor Model

In the case of latent factor models, $\Theta_u^p = \mathbf{p}_u$ are the user factors, which should be kept private, since, if accurate item factors are learned across the system, then knowing $\mathbf{p}_u$ is sufficient to predict $\hat{\mathbf{r}}_u$. To convert the loss functions of Section 3 to the distributed setting, we introduce the local parameters as $\Theta_u = [\mathbf{p}_u, \mathrm{Q}_u]$, such that each agent now maintains its own item factors $\mathbf{q}_{ui}$ and,

$$\mathcal{L}_{uv}(\mathbf{p}_u, \mathrm{Q}_u, \mathrm{Q}_v) = \lambda_r \|\mathrm{Q}_u - \mathrm{Q}_v\|_F^2 + \lambda_l \mathcal{L}_u(\mathbf{p}_u, \mathrm{Q}_v, \mathbf{r}_u) ,$$

where $\|\cdot\|_F$ represents the Frobenius norm.

Algorithm 1: Decentralised Algorithm for MF

Input : user-item preference rating vector $\{\mathbf{r}_u \mid u \in U\}$, communication graph G, such that $N_u(G) = \{v \mid v \in U, (u, v) \in G\}$ is the set of neighbours of u in G, learning rate η_e and cross-loss regulariser λ_l.

1 **forall** e *epoch* **do**

// Learn Phase

2 **forall** $u \in U$ ***in parallel*** **do**

3 Draw T_e item samples $\mathcal{T}$ `// The sampling depends on the objective`

4 **for** $\ell \in \mathcal{T}$ **do**

5 Update $\mathbf{p}_u = \mathbf{p}_u - \eta_e \frac{\partial \mathcal{L}_{u\ell}}{\partial \mathbf{p}_u}(\mathbf{r}_u)$

6 Update $\mathbf{q}_{u\ell} = \mathbf{q}_{u\ell} - \eta_e \frac{\partial \mathcal{L}_{u\ell}}{\partial \mathbf{q}_{u\ell}}(\mathbf{r}_u)$

// Share Phase

7 **forall** $u \in U$ ***in parallel*** **do**

8 Sample $I_u \subseteq N_u(G)$ `// agents sampled from neighbours of` u

9 **forall** $v \in I_u$ **do**

10 $S = \textsc{SyncItems}(u, v)$ `// Set of items shared between` u `and` v

11 Gather $\mathbf{q}_{vS}$ from v

12 **if** $\lambda_l > 0$ **then**

13 Update and Send $\frac{\partial \mathcal{L}_{uv}}{\partial \mathbf{q}_{vS}}$

14 Gather $\frac{\partial \mathcal{L}_{vu}}{\partial \mathbf{q}_{uS}}$

15 Update $\mathbf{q}_{uS} = \mathbf{q}_{uS} - \eta_e \frac{\partial \mathcal{L}_{vu}}{\partial \mathbf{q}_{uS}}$

16 Update $\mathbf{p}_u = \mathbf{p}_u - \eta_e \frac{\partial \mathcal{L}_{uv}}{\partial \mathbf{p}_u}$

[Decentralised MF Objective] The distributed latent matrix factor (MF) objective is written as

$$\mathcal{L}(\mathrm{P}, \mathrm{Q}_1, \ldots, \mathrm{Q}_n) = \frac{1}{n} \sum_u \left(\mathcal{L}_u(\mathbf{p}_u, \mathrm{Q}_u) + \sum_{v \in N_u(G)} \mathcal{L}_{uv}(\mathbf{p}_u, \mathrm{Q}_u, \mathrm{Q}_v) \right). \quad (2)$$

While minimising over $\mathcal{L}_u$ sets the local parameters to fit the loss associated with local data as well as possible, and the terms $\mathcal{L}_{uv}$ are necessary in order to ensure that the local model generalises to unseen data.

4.2 Decentralised Algorithm

The distributed algorithm that we propose to minimise Eqn. (2) is shown in Algorithm 1. The algorithm involves the following steps:

- A local *learn* phase (lines 3-6 in Algorithm 1), in which the loss $\mathcal{L}_u$ is reduced through an SGD process.
- A *share* phase (lines 7-15 in Algorithm 1), which randomly samples from the neighbourhood of each agent u to get a set of neighbours $I_u \subseteq N_u(G)$ (line 8 in Algorithm 1); and during which u and $v \in I_u$, synchronise to reduce the $\mathcal{L}_{uv}$ loss over some agreed set of items S (lines 9-15 in Algorithm 1).

In the learn phase, the parameter updates are calculated by sampling from the summands of the local loss (i.e. sampling pairs (u, i) for WMF or triples (u, i, j) for BPR, see Section 3), and updating in the direction of the stochastic gradients associated with these summands. In the share phase, selecting neighbours v at random, the pair (u, v) agrees a set of item factors to share (Algorithm 1 line 14) and the parameters are updated in the direction of the stochastic gradient associated with *all* these items. Hence a single message results in an update of $|S|$ item factors, where S is the set of shared items (line 15 in Algorithm 1). Note that if $\lambda_l = 0$, the synchronisation step only requires an update for the item factors, which can be applied after receiving the item factors, $\mathbf{q}_{vi}\, \forall i \in S$, from the neighbour v. On the other hand, if $\lambda_l \neq 0$, then the gradient of the cross-loss term in $\mathcal{L}_{vu}$ is required by agent u, but must be computed by neighbour v, since it requires agent v's private data (see lines 16-19 in Algorithm 1). Hence, the update takes place in the following steps:

- Agents u and v exchange item factors
- Agent v computes the gradient of the cross-loss for agent u and vice versa
- Agents u and v exchange cross-loss gradients.

The share phase updates for the WMF algorithm are given in Algorithm 2. The BPR updates may be found in the supplemental material.

Algorithm 2: Decentralised WMF Share Phase Updates

1 **forall** $v \in I_u$ **do**
2 **forall** $i \in S$ **do**
3 Exchange $\mathbf{q}_{ui}$ and $\mathbf{q}_{vi}$ between u and v.
 `// u computes & sends cross-loss gradient wrt q_vi to v.`
4 $\frac{\partial \mathcal{L}^{\text{WMF}}{}_{uv}}{\partial \mathbf{q}_{vi}} = 2\lambda_l c_{ui}(\mathbf{p}_u^\top \mathbf{q}_{vi} - \tilde{r}_{ui})\mathbf{p}_u$
 `// v computes & sends cross-loss gradient wrt q_ui to u.`
5 $\frac{\partial \mathcal{L}^{\text{WMF}}{}_{vu}}{\partial \mathbf{q}_{ui}} = 2\lambda_l c_{vi}(\mathbf{p}_v^\top \mathbf{q}_{ui} - \tilde{r}_{vi})\mathbf{p}_v$
6 u updates: $\mathbf{q}_{ui} = \mathbf{q}_{ui} - \eta_e \frac{|N_v| w_{vu}}{|I_v||S|}(\frac{\partial \mathcal{L}^{\text{WMF}}{}_{vu}}{\partial \mathbf{q}_{ui}} + 2\lambda_r(\mathbf{q}_{vi} - \mathbf{q}_{ui}))$
7 v updates: $\mathbf{q}_{vi} = \mathbf{q}_{vi} - \eta_e \frac{|N_u| w_{uv}}{|I_u||S|}(\frac{\partial \mathcal{L}^{\text{WMF}}{}_{uv}}{\partial \mathbf{q}_{vi}} + 2\lambda_r(\mathbf{q}_{ui} - \mathbf{q}_{vi}))$
 `// u computes gradient wrt p_u, using received q_vi.`
8 $\frac{\partial \mathcal{L}^{\text{WMF}}{}_{uv}}{\partial \mathbf{p}_u} = 2\frac{\lambda_l}{|S|} c_{ui}(\mathbf{p}_u^\top \mathbf{q}_{vi} - \tilde{r}_{ui})\mathbf{q}_{vi}$
 `// v computes gradient wrt p_v, using received q_ui.`
9 $\frac{\partial \mathcal{L}^{\text{WMF}}{}_{vu}}{\partial \mathbf{p}_v} = 2\frac{\lambda_l}{|S|} c_{vi}(\mathbf{p}_v^\top \mathbf{q}_{ui} - \tilde{r}_{vi})\mathbf{q}_{ui}$
10 u updates: $\mathbf{p}_u = \mathbf{p}_u - \eta_e \frac{|N_u| w_{uv}}{|I_u|} \frac{\partial \mathcal{L}^{\text{WMF}}{}_{uv}}{\partial \mathbf{p}_u}$
11 v updates: $\mathbf{p}_v = \mathbf{p}_v - \eta_e \frac{|N_v| w_{vu}}{|I_v|} \frac{\partial \mathcal{L}^{\text{WMF}}{}_{vu}}{\partial \mathbf{p}_v}$

4.3 Theoretical Results

We state certain theoretical results about the decentralised framework in this section, but enclose all the proofs in the supplemental material. Write $\mathbf{S} = \{\Theta_u \mid u \in U\}$ and $\mathcal{L}^{(t)} = f(\mathbf{S}^{(t)}) + g(\mathbf{S}^{(t)})$ as the value of the loss at time-step t, where $f(\mathbf{S}^{(t)}) = \sum_u \mathcal{L}_u$ and $g(\mathbf{S}^{(t)}) = \sum_{u,v} \mathcal{L}_{uv}(\mathbf{S}^{(t)})$. Given that the decentralised

objective $\mathcal{L}$ is L-smooth, the decentralised latent factor algorithms, using the learning rate $\eta_t = \eta/L$, have convergence rate $O\left(\frac{4LG}{\eta T_e T} + 2\eta\left(\sigma_1^2 + \frac{\sigma_2^2}{T_e}\right)\right)$, where

- $T_e = |\mathcal{T}|$ which is size of the sampled items for each iteration in Algorithm 1
- $G \triangleq \mathcal{L}^{(0)} - \mathrm{E}\left[\mathcal{L}^{(t)}\right]$
- $\sigma_1^2 : \mathrm{E}[\|f(\mathbf{S}^{(t)})\|^2] \leq \sigma_1^2$, $\sigma_2^2 : \mathrm{E}[\|g(\mathbf{S}^{(t)})\|^2] \leq \sigma_2^2$, $\forall t$

An everywhere differentiable function that has L-Lipschitz gradients is said to be L-smooth, and L-Lipschitz continuity indicates that a function's derivatives are bounded everywhere in its scope. We appeal to this property as it is known that the latent factor objective is non-convex. We have the following two corollaries, On any bounded domain of the objective, WMF is L-smooth. On any bounded domain of the objective, BPR is L-smooth.

5 Experiments

We run our experiments on a single multicore server using shared-memory parallelisation. The WMF model is implemented in `C++` using `OpenMP` and the BPR model is implemented in `Matlab`, using the parallel toolbox. The cooperating agents are simulated as independent threads. The entire parameter set of $n(m+1) \times k$ parameters $\{\mathbf{p}_u, \mathrm{Q}_u \mid u \in U\}$ are stored in the shared memory. For the Movielens 1M dataset, with $k = 20$, this amounts to > 3.5GB of storage.

5.1 SOTA Comparison

Gossip algorithms. When $\lambda_l = 0$, the update due to the L_{uv} terms amounts to:

$$\mathbf{q}_{ui} = \mathbf{q}_{ui} - 2\eta_e\lambda_r \frac{|N_u(G)|}{|I_u|}\left(\mathbf{q}_{ui} - \sum_{v \in I_u} w_{uv}\mathbf{q}_{vi}\right) = (1-\tilde{\eta})\mathbf{q}_{ui} + \tilde{\eta}\sum_v w_{uv}\mathbf{q}_{vi}\,, \tag{3}$$

for some $\tilde{\eta} > 0$, that is, a weighted average of the agent's item-factor with that of the neighbours. The method becomes a gossip-like algorithm for bringing the shared parameters $\mathbf{q}_{ui}$ to consensus. In [8], such a gossip algorithm is evaluated for recommendation using a matrix factorisation model, but applied to the rating prediction problem, rather than the top-N problem. In our experimental analysis, the case $\lambda_l = 0$ is used to compare with such a gossip algorithm.

Decentralised Matrix Factorisation (DMF). In the works [3, 6], a decentralised matrix factorisation algorithm is proposed in which communication with neighbours occurs after every update. In [3], rather than sending the item-factors between neighbours, instead neighbour v sends $\frac{\partial \mathcal{L}}{\partial \mathbf{q}_{vi}}$ to agent u and this gradient is combined into u's model via $\mathbf{q}_{ui} = \mathbf{q}_{ui} - \tilde{\eta}(\mathbf{p}_v^\top \mathbf{q}_{vi} - r_{vi})\mathbf{p}_v$ for some update weight $\tilde{\eta}$. Note that in this update rule, a gradient evaluated on v's parameters $(\mathbf{p}_v, \mathbf{q}_{vi})$ is used to update u's item-factor $\mathbf{q}_{ui}$. (Compare with Algorithm 2, in which the gradient of the distributed loss function involves a mix

of u's and v's parameters). As a result, the algorithm requires that $\mathbf{q}_{vi}^{(t)} \approx \mathbf{q}_{ui}^{(t)}$ at all iterations, t, and cannot sustain any lag between updates and communications. We show that, as well as being mathematically more sound, our method empirically out-performs this algorithm on the evaluated datasets.

5.2 Results

We evaluate the algorithm using three datasets:

- Movielens 1M(`ML1M`): $6,040$ users, $3,952$ items, 1M interactions, 5% sparsity;
- Movielens 100K(`ML100K`): 943 users, $1,682$ items, 100K interactions, 3%sparsity;
- Last.fm(`lastFM`): random sparse subset of 975 users, $4,000$ artists, $28,950$ interactions (normalised to range $[1, 5]$), 0.6% sparsity.

We train on 80% of the data, chosen at random, and test on the remaining 20%. To tune the parameters, we further split the training data, such that 90% of training data is used for training and the remaining 10% for validation. We evaluate performance using precision@N, the proportion of the top-N recommendations that are in the test set (where a test-set item is counted as long as some interaction between it and the user has been recorded)[3]. We show results for $N = 10$. In order to track the extent to which the distributed parameters reach a consensus, it is also interesting to plot the Q-norm, calculated as $\frac{1}{k}\sum_u \sum_{i \in R_u} \left\| \mathbf{q}_{ui} - \frac{1}{|\{v|i \in R_v\}|} \sum_{v|i \in R_v} \mathbf{q}_{vi} \right\|^2$. To gauge convergence of the decentralised optimisation, we track the following training set loss:

- BPR: The mean of $1 - \overline{\sigma(-\mathbf{p}_u(\mathbf{q}_{vi} - \mathbf{q}_{vj}))}$, over 10,000 randomly chosen pairs $\{(i, j) \mid i \in R_u, j \notin R_u\}$, and randomly chosen agents u and v
- WMF: The mean of $\sqrt{c_{ui}(\mathbf{p}_u^\top \mathbf{q}_{vi} - \tilde{r}_{u,i})^2}$, over all ratings in R and randomly chosen neighbours v of each agent u.

Further, we choose $w_{uv} = 1/|N_u G|$, $|I_u| = 1$, i.e. each agent communicates with a *single* randomly chosen neighbour. This is implemented as a random edge matching, in which, at the start of the share phase, pairs of neighbouring agents agree to share. Any node that fails to find a match, skips the share phase in this epoch. We examine the following research questions:

RQ1: The rate of convergence of the decentralised algorithm to the performance of a central algorithm.

RQ2: The effect of communication graph sparsity on the performance.

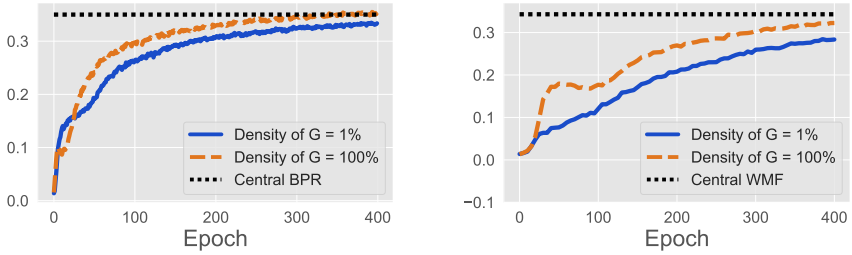
RQ3: The benefit gained from the cross-loss term, over parameter averaging.

RQ4: The communication overhead in terms of number of messages and message volume.

[3]Other accuracy measures follow the trends we see for prec@10, achieving the well-known scores of the central algorithm when the decentralised algorithm converges, see e.g. www.librec.net/release/v1.3/example.html

Table 1: Prec@10, Central vs Decentralised Model at epoch(ep)

	BPR				
Dataset	Central	100ep	200ep	300ep	400ep
LastFM	0.145	0.105	0.119	0.133	0.140
ML100k	0.363	0.284	0.329	0.349	0.357
ML1M	0.334	0.165	0.225	0.269	0.293
	WMF				
Dataset	Central	100ep	200ep	300ep	400ep
LastFM	0.116	0.052	0.097	0.104	0.105
ML100k	0.343	0.276	0.279	0.317	0.327
ML1M	0.295	0.135	0.175	0.213	0.254

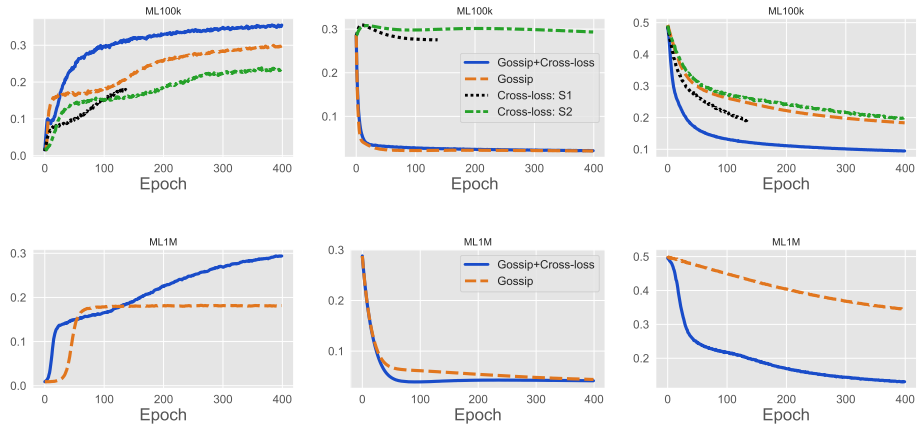


(a) BPR (b) WMF

Figure 1: Fully connected G vs random G with density=1% showing Prec@10 on ML100k

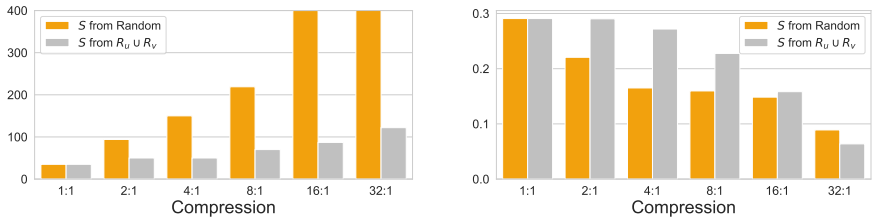
RQ1 and RQ2: We run the decentralised algorithm for 400 epochs on the three datasets. The results for a communication graph of density 0.2, synchronising over all items ($S = I$), are summarised in Table 1. We see convergence towards the performance of the central algorithm. Nevertheless, for ML1M for example, the central algorithm reaches its peak performance by 100 epochs, so decentralisation must pay a significant penalty over central algorithms in terms of convergence speed. We use ML100K to examine *RQ2*. In Figure 1, we show the convergence of the method, when agents are free to choose *any* node from a fully connected communication graph and when the communication graph has density of just 1%, i.e. each agent is connected to an average of 16.8 neighbours. It may be observed that the algorithms are very robust to communication graph sparsity, with just a small drop-off in convergence rate, even on a very sparse communication graph, so that we can expect convergence even in scenarios in which agents can only directly access to a few others.

RQ3: We examine *RQ3* using ML100k and ML1M. Due to space restrictions we only report results from the BPR model. Results obtained from the WMF model are along the same lines (see supplemental material). We fix a random communication graph of density 0.02. In Figure 2, the prec@10, the Q-norm and the loss are shown for a number of different settings of the hyperparameters to highlight the difference between the convergence of gossip learning and an



(a) Precision@10 (b) Q-norm (c) Loss

Figure 2: BPR on ML100k and ML1M Comparing Gossip and Cross-loss



(a) Epochs to prec@10≥0.2 (ML100k) (b) prec@10 at 400ep (ML1M)

Figure 3: Effect of Compression for BPR

addition of our cross-loss term, we report: (i) *Gossip* (i.e. $\lambda_r > 0$, $\lambda_l = 0$), (ii) *Gossip+Cross-loss* (i.e. $\lambda_l > 0$, $\lambda_r > 0$), (iii) *Cross-loss: S1* ($\lambda_r = 0$, $\lambda_l = 2.3$) and (iv) *Cross-loss: S2* ($\lambda_r = 0$, $\lambda_l = 2.5$).

It can be observed that the fastest convergence is obtained when the *Gossip+Cross-loss* setting is used, the algorithm is carrying out parameter averaging according to Eqn. (3). On ML100k convergence is slower and on ML1M, the learning gets trapped in a local minimum with maximum precision of 0.18. On the other hand, for ML100k, we also show two convergence plots when only the *Cross-loss* setting is used. In this case, there is no mechanism to drive the item-factors to a consensus. It *is* possible to achieve some learning in these settings, but the algorithm is more brittle, with a tendency for the Q-norm to diverge, unless the learning rate is carefully managed. Here, with *Cross-loss: S1*, the learning diverges after 238 epochs, while with *Cross-loss: S2*, convergence to a prec@10 of 0.23 is achieved. As pointed out previously, this setting is somewhat similar to the DMF algorithm of [3], which, for comparison, in our own C++ implementation, reaches prec@10 of 0.197 after 100 epochs and 0.217 after 400 epochs.

RQ4: We use ML100k, ML1M and the BPR model to examine the effect of compressing the message communicated during the share phase. With privacy

in mind, an agent may not wish to release any information to its neighbour about which items it has interacted with. Therefore, it may chose to select some random set, S, of item-factors to share. On the other hand, the algorithm is more robust to compression when interacted items are shared. Messages are compressed at a rate of 1:1 up to 32:1 (uncompressed to compressed message size), by selecting the shared items S at random, or selecting shared items from $R_u \cup R_v$ (and adding or removing items at random to reach the required size). In Figure 3a, we show the number of epochs it takes to reach prec@10$\geq$0.2 in `ML100k` and in Figure 3b, the prec@10 after 400eps in `ML1M`. Compression slows convergence, but considerably less so when interacted items are shared. The trade-off between the amount of data shared per epoch and the number of epochs required for convergence results in there being no benefit to compressing beyond 16:1 in `ML100k`, when sharing random items. Interacted items afford greater performance benefits, but at a greater privacy risk. For example, for 8:1 compression using rated items, in `ML1M`, the rated items, that average 132 per user, are hidden among $\approx$ 500 shared items. Each agent communicates $2|S|k$ double precision numbers per epoch in two messages with the neighbour, so that there are two messages and a total volume of communication of $16nk|S|$ bytes per epoch. The DMF algorithm of [3] requires a communication per update. Assuming that one update per data-point across the entire machine is carried out per epoch, this amounts to T_e (=800k for `ML1M`) separate messages, each of size $8k$ bytes. In a communication-bound environment, with high latencies, such a number of messages may be infeasible. While the uncompressed message we send is large, our analysis has shown that the communication volume can be significantly reduced, without significant loss of performance. Further details on algorithm complexity and hyper-parameter selection are given in the supplemental material.

6 Conclusions

We have demonstrated a framework for decentralised ML, consisting of a loss function over the distributed parameters, along with a communication efficient algorithm for minimising this loss function. Results on two latent factor recommendation algorithms, show that the framework achieves the performance of the central algorithm, with accelerated convergence in comparison to gossip algorithms that rely only on parameter averaging. Moreover, the algorithm converges on sparse communication graphs and is robust to message compression.

Acknowledgements

The work is supported by the Science Foundation Ireland under the grant number SFI/12/RC/2289_P2 and Samsung Research, Samsung Electronics Co., Seoul, Republic of Korea. We wish to thank the reviewers for the helpful feedback.

References

[1] Assran, M., Loizou, N., Ballas, N., Rabbat, M.: Stochastic gradient push for distributed deep learning. In: Proceedings of the ICML. Long Beach, California (2019)

[2] Bellet, A., Guerraoui, R., Taziki, M., Tommasi, M.: Personalized and private peer-to-peer machine learning. In: Proceedings of the 21st AISTATS. Lanzarote (2017)

[3] Chen, C., Liu, Z., Zhao, P., Zhou, J., Li, X.: Privacy preserving point-of-interest recommendation using decentralized matrix factorization. In: AAAI (2018)

[4] Colin, I., Bellet, A., Salmon, J., Clémençon, S.: Gossip dual averaging for decentralized optimization of pairwise functions. In: Proc. 33rd ICML (2016)

[5] Ammad-ud din, M., Ivannikova, E., Khan, S.A., Oyomno, W., Fu, Q., Tan, K.E., Flanagan, A.: Federated collaborative filtering for privacy-preserving personalized recommendation system. arXiv preprint arXiv:1901.09888 (2019)

[6] Duriakova, E., Z. Tragos, E., Smyth, B., Hurley, N., J. Pena, F., Symeonidis, P., Geraci, J., Lawlor, A.: PDMFRec: A decentralised matrix factorisation with tunable user-centric privacy. In: Proc. 13th RecSys. ACM (2019)

[7] He, C., Tan, C., Tang, H., Qiu, S., Liu, J.: Central server free federated learning over single-sided trust social networks. arXiv preprint arXiv:1910.04956 (2019)

[8] Hegedűs, I., Danner, G., Jelasity, M.: Decentralized recommendation based on matrix factorization: A comparison of gossip and federated learning. In: Cellier, P., Driessens, K. (eds.) Machine Learning and Knowledge Discovery in Databases. pp. 317–332. Springer International Publishing, Cham (2020)

[9] Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: 2008 Eighth IEEE International Conference on Data Mining. pp. 263–272 (Dec 2008)

[10] Jalalirad, A., Scavuzzo, M., Capota, C., Sprague, M.: A simple and efficient federated recommender system. In: Proceedings of the 6th IEEE/ACM International Conference on Big Data Computing, Applications and Technologies. pp. 53–58. ACM, New York, NY, USA (2019)

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

[12] Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science. p. 482. FOCS '03, IEEE Computer Society, USA (2003)

[13] Koloskova, A., Stich, S.U., Jaggi, M.: Decentralized stochastic optimization and gossip algorithms with compressed communication. In: Proceedings of the 36th International Conference on Machine Learning. Long Beach, California (2019)

[14] Konecný, J., McMahan, H.B., Ramage, D.: Federated optimization: Distributed optimization beyond the datacenter. ArXiv **abs/1511.03575** (2015)

[15] Lian, X., Zhang, C., Zhang, H., Hsieh, C.J., Zhang, W., Liu, J.: Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. In: Advances in Neural Information Processing Systems 30. pp. 5330–5340 (2017)

[16] McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. pp. 1273–1282 (2017)

[17] Nedic, A., Ozdaglar, A.: Distributed subgradient methods for multi-agent optimization. IEEE Transactions on Automatic Control **54**(1), 48–61 (Jan 2009)

[18] Nedić, A., Olshevsky, A., Rabbat, M.G.: Network topology and communication-computation tradeoffs in decentralized optimization. Proceedings of the IEEE **106**(5), 953–976 (2018)

[19] Papaioannou, T.G., Ranvier, J.E., Olteanu, A., Aberer, K.: A decentralized recommender system for effective web credibility assessment. In: Proceedings of the 21st ACM international conference on Information and knowledge management. pp. 704–713. ACM (2012)

[20] Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: BPR: Bayesian personalized ranking from implicit feedback. In: Proceedings of the 25th Conference on Uncertainty in AI. pp. 452–461. AUAI Press, Arlington, Virginia, USA (2009)

[21] Tang, H., Lian, X., Yan, M., Zhang, C., Liu, J.: D^2: Decentralized training over decentralized data. In: International Conference on Machine Learning. pp. 4848–4856 (2018)

[22] Tsitsiklis, J., Bertsekas, D., Athans, M.: Distributed asynchronous deterministic and stochastic gradient optimization algorithms. IEEE Transactions on Automatic Control **31**(9), 803–812 (Sep 1986)

[23] Vanhaesebrouck, P., Bellet, A., Tommasi, M.: Decentralized collaborative learning of personalized models over networks. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) (2017)

[24] Yuan, K., Ling, Q., Yin, W.: On the convergence of decentralized gradient descent. SIAM Journal on Optimization **26**(3), 1835–1854 (2016)