Contributions to distributed multi-task machine learning

Amaury Bouchra Pilet

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Amaury BOUCHRA PILET

Contributions à l'apprentissage machine distribué multitâche

Contributions to distributed multi-task machine learning

Thèse présentée et soutenue à Rennes, le 10/11/2021
Unité de recherche : Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA)
UMR CNRS 6074

Rapporteurs avant soutenance :

Marc TOMMASI	Professeur des Universités, Université de Lille
Giovanni NEGLIA	Chargé de Recherche, Inria Sophia Antipolis

Composition du Jury :

Président :	Alexandre TERMIER	Professeur des Universités, Université de Rennes 1
Rapporteurs :	Marc TOMMASI	Professeur des Universités, Université de Lille
	Giovanni NEGLIA	Chargé de Recherche, Inria Sophia Antipolis
Examinateurs :	Sara BOUCHENAK	Professeur des Universités, INSA Lyon
	Aurélien BELLET	Chargé de Recherche, Inria Lille
Dir. de thèse :	Davide FREY	Chargé de Recherche, Inria Rennes
Co-dir. de thèse :	François TAÏANI	Professeur des Universités, Université de Rennes 1

UMR IRISA

Thèse de Doctorat

Contributions to distributed multi-task machine learning

Amaury Bouchra Pilet

Institut de Recherche en Informatique et Systèmes Aléatoires
Université de Rennes
Ancien de l'École Normale Supérieure « Ulm »

Directeurs de thèse :
Pr. François Taïani
Dr. Davide Frey

226 - 229

[EN] Advice to reader

For legal reasons, this thesis includes a summary in French ("Résumé"). Non-french speaking readers can safely skip this part.

[FR] Avis au lecteur

L'essentiel du contenu de cette thèse a été rédigé en langue anglaise, conformément aux pratiques usuelles dans le domaine de l'informatique. Comme prévu par la loi, un résumé en français est inclus et constitue légalement le corps de la thèse ; la partie en anglais constituant des annexes, détaillant le contenu scientifique de ce travail.

[...] And I'm a stodgy old scientist who believes, naively, that there exists an external world, that there exist objective truths about that world, and that my job is to discover some of them.

Alan Sokal

L'ensemble des problèmes résolubles en temps polynomial est inclus strictement dans celui des problèmes dont la solution est vérifiable en temps polynomial, tout comme l'ensemble des êtres censés est inclus strictement dans celui des humains, encore que je ne sois pas tout à fait sûr pour $P \neq NP$.

Pour paraphraser Albert Einstein

Contents

Résumé

I Introduction

L'apprentissage machine est un des domaines les plus importants et les plus actifs dans l'informatique moderne. Ses applications vont de l'analyse de grandes quantités de données dans des centres de données à la fourniture de services basés sur l'intelligence artificielle sur de petits appareils personnels [Haz+18; Sae+17] ; des applications qui connaissent un développement rapide et un intérêt certain de l'industrie.

Cependant, la plupart des systèmes d'apprentissage machine actuels utilisent encore une architecture essentiellement centralisée. Même si l'application finale doit être délivrée sur de nombreux systèmes, parfois des millions (voire des milliards) d'appareils individuels, le processus d'apprentissage est toujours centralisé dans un centre de calcul. Ce peut être un problème notamment si les données d'apprentissage sont sensibles, comme des conversations privées, des historiques de recherche ou des données médicales [Che+17].

Une solution serait d'exécuter le processus d'apprentissage sur des systèmes appartenant aux utilisateurs finaux, plutôt que dans un centre de calcul. Cela permettrait également de transférer de couteuses tâches de calcul vers les systèmes des utilisateurs finaux, plutôt que de les effectuer sur une infrastructure dédiée. Ces avantages ont amené de grands noms de l'industrie, tels que Google et Facebook, à commencer des recherches sur des architectures *fédérées*, dans lesquelles un serveur central organise une tâche de calcul collaborative exécutée sur plusieurs systèmes distants.

Actuellement, la catégorie la plus connue d'algorithmes d'apprentissage machine est très probablement celle des réseaux de neurones. Bien qu'il existe d'autres types d'algorithmes, de nombreuses applications remarquables sont basées sur les réseaux de neurones. Ils permettent aux ordinateurs de se comparer aux humains dans des tâches qui seraient très difficiles, voire impossibles, à effectuer avec des systèmes moins avancés. Leurs principaux inconvénients sont leur complexité et leur comportement peu prévisible, ce qui rend les applications complexes, telles que les applications distribuées, plus difficiles à réaliser qu'avec des algorithmes plus simples.

Un autre défi important pour l'apprentissage machine moderne est l'apprentissage multitâche. Pour des applications telles que la reconnaissance de l'écriture manuscrite ou de la parole, la tâche spécifique consistant à reconnaitre l'écriture ou la voix d'un individu précis est différente d'un utilisateur à l'autre. De plus, certains utilisateurs peuvent avoir des tâches plus proches que d'autres, notamment en fonction de leur localisation. L'apprentissage machine multitâche a déjà reçu un certain intérêt dans un contexte centralisé, mais les approches distribuées manquent. Nous voudrions que l'apprentissage machine distribué soit capable d'apprendre des tâches similaires mais différentes et, idéalement, d'exploiter les différents niveaux de similarité entre les utilisateurs pour optimiser encore plus le processus.

Bien qu'une telle optimisation (regrouper certains utilisateurs pour l'apprentissage) puisse être réalisable manuellement dans certains contextes, cela pourrait être plus difficile dans d'autres. Par exemple, si, pour la reconnaissance vocale, les données de localisation peuvent être utilisées pour optimiser le processus (en regroupant les utilisateurs par localité), il serait en revanche difficile dans le cadre de la reconnaissance de l'écriture manuscrite, de classifier les utilisateurs (en fonction de leur « style » d'écriture). Un bon système d'apprentissage machine distribué devrait intégrer une méthode

générale pour évaluer et exploiter les différents niveaux de similarité entre tâches.

II Problème & défis

Afin de rendre réalisables les applications mentionnées précédemment, nous avons besoin d'un système d'apprentissage machine distribué multitâche remplissant certains critères : large champ d'application, données d'apprentissage gardées locales, parallélisation effective, exploitation des différents degrés de similarité entre utilisateurs, flexibilité sur l'architecture (fédérée (avec un serveur pour la coordination) ou complètement décentralisée). Nous listons ci-après les objectifs non atteints par les solutions existantes et que nous nous proposons d'atteindre dans ce travail.

Nous avons besoin d'un système qui fonctionne avec des types de modèles (d'intelligence artificielle) variés, et notamment avec les réseaux de neurones. Afin de garantir cette généricité, nous assumerons simplement dans la suite de ce travail que les modèles appris peuvent se réduire à des collections (finies) de valeurs réelles ($\in \mathbb{R}$). Ce système devra permettre l'apprentissage distribué sans que les utilisateurs aient à divulguer leurs données privées, tout en étant capable d'apprendre tous en même temps (calcul parallèle). Notre système devra permettre aux utilisateurs d'apprendre des modèles similaires mais différents, en même temps et de manière collaborative. Il devra prendre en compte des degrés variables de similarité.

Notre système devra fonctionner avec les deux types d'architectures distribuées : *décentralisées* et *fédérées*. Les architectures fédérées ont un serveur central coordonnant (et/ou agrégeant) le travail de plusieurs clients. Dans une architecture décentralisée, les clients deviennent des pairs qui communiquent entre eux et collaborent sans la coordination d'un serveur central.

Nous devons également fournir un système complémentaire, capable de gérer automatiquement les différents niveaux de similarité entre tâches et de configurer le système d'apprentissage en conséquence. Ce système devra être capable de déterminer quels utilisateurs ont les tâches les plus proches afin de les faire collaborer plus étroitement dans le processus d'apprentissage.

III Contributions

Afin de répondre aux exigences ci-avant, nous proposons trois contributions dans cette thèse :

- Apprentissage machine distribué multitâche en lui-même [Chapitre 3]
- Formation de coalition applicable à l'apprentissage machine distribué multitâche [Chapitre 4]
- Application effective de la formation de coalition à l'apprentissage machine distribué multitâche et exploration des implications pratiques de la vision associée [Chapitre 5]

III.1 Apprentissage machine distribué multitâche en lui-même

Notre première contribution est notre système d'apprentissage machine distribué en lui-même. Notre système est basé sur le *moyennage*, une méthode communément utilisée en apprentissage distribué. L'idée est simple ; pour permettre l'apprentissage machine sur un modelè constitué d'une collection de paramètres réels, les apprenants travaillent séparément et, périodiquement, un modèle commun est construit en prenant la moyenne des valeurs apprises par les apprenants individuels, qui prendront ce modèle commun comme base pour leur prochain cycle d'apprentissage. L'avantage de cette méthode est qu'elle peut fonctionner sur n'importe quel type de modèle basé sur des valeurs réelles ($\mathbb{R}$), indépendamment de l'algorithme d'apprentissage utilisé, tant que celui-ci implique uniquement de modifier les valeurs, pas d'en ajouter où d'en enlever ; dans ce travail, cependant, nous nous concentrerons sur les réseaux de neurones. Ce système peut également fonctionner aussi bien avec

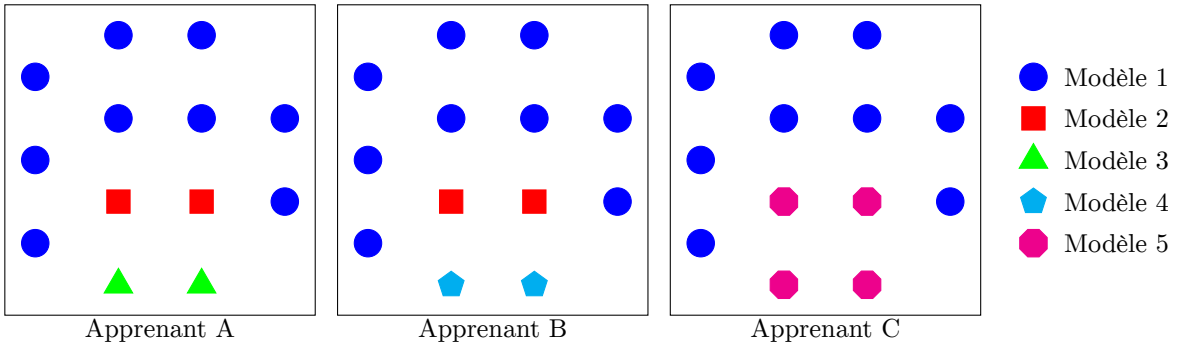


Figure 1: 3 apprenants avec 5 modèles partiels intégrés dans la géométrie des réseaux de neurones des apprenants

une architecture fédérée (les modèles sont moyennés par un serveur central) que décentralisée (avec le moyennage épidémique [JMB05]).

Cette méthode en elle-même ne permet par l'apprentissage multitâche. Pour le permettre, deux types de solutions sont possibles (nous introduisons la terminologie) : le *partage souple* de tous les paramètres et le *partage rigide* de certains paramètres. Partage souple signifie que, plutôt que de faire converger tous les apprenants vers un modèle unique, il leur est permis de dévier légèrement de celui-ci. Partage rigide (de certains paramètres) signifie que le modèle commun ne couvre qu'une partie du modèle appris, certains paramètres étant spécifiques à chaque apprenant.

Nous avons choisi la deuxième solution : le partage rigide de certains paramètres. La raison de ce choix est que les réseaux de neurones ont généralement une structure « fonctionnelle » : les réseaux sont divisés en plusieurs parties, chacune étant supposée avoir une fonction plus ou moins spécifique. Par exemple, en vision artificielle, les réseaux de neurones sont généralement divisés en couches, dont les premières sont supposées reconnaitre des structures simples alors que les secondes devraient les interpréter. Contrairement au partage souple de tous les paramètres, un partage rigide de paramètres prédéterminés permet aux ingénieurs de choisir les parties des modèles qui doivent être partagées et celles qui doivent rester personnelles. Notez que les réseaux de neurones comportent généralement un grand nombre de neurones équivalents : mêmes valeurs d'entrée et sortie utilisée comme entrée par les mêmes neurones. Cela permet d'utiliser le partage rigide sans savoir précisément quelles parties doivent être partagées, en désignant dans chaque partie certains neurones comme partagés et d'autres comme personnels. Avec une telle configuration, le processus d'apprentissage lui-même devrait être capable de combiner les éléments partagés et personnels de manière optimale (dans la mesure de l'optimisation permise par les algorithmes d'apprentissage des réseaux de neurones), le processus d'apprentissage réglant les poids de manière appropriée.

Nous formalisons ce système avec la notion de *modèle partiel*. Les modèles partiels peuvent être vus comme des briques que l'on peut utiliser pour construire un modèle complet pour chaque utilisateur. Ces modèles peuvent être liés à des portions spécifiques du réseau de neurones de chaque utilisateur, permettant ainsi de sélectionner les portions les plus pertinentes pour le partage (concrètement, moyennage) si des informations préalables sont disponibles permettant d'optimiser ce choix (sinon, il est possible de simplement partager des portions de chaque partie et de laisser l'algorithme d'apprentissage régler les poids de manière appropriée). Les modèles partiels peuvent être partagés par tous les pairs (*modèle global*), être spécifiques à un utilisateur (*modèle local*) ou être partagés par un sous-ensemble précis d'utilisateurs (*modèle semi-local*). La Figure 1 présente un exemple de trois apprenants avec cinq modèles partiels implémentés à des emplacements précis dans la géométrie des réseaux de neurones. Le modèle 1 (cercles bleu) est global, le modèle 2 (carrés rouges) est semi-local et les modèles 3 (triangles verts), 4 (pentagones cyan) et 5 (octogones magenta) sont locaux.

Nous avons conduit des expériences sur différents jeux de données qui ont montré que notre concept fonctionne comme prévu. Dans ces expériences, nous avons testé différentes configurations pour notre

système, ce qui nous a permis de déterminer quelles configurations donnaient les meilleurs résultats. Notamment, nous avons montré que la flexibilité permise par nos modèles semi-locaux peut être exploitée pour améliorer les résultats.

Ce travail a été présenté au « 19th Symposium on Intelligent Data Analysis (IDA 2021) » sous le titre « Simple, Efficient and Convenient Decentralized Multi-Task Learning for Neural Networks » [BFT21b].

III.2 Formation de coalition applicable à l'apprentissage machine distribué multitâche

Notre système d'apprentissage multitâche suppose que les participants soient déjà regroupés en fonction des tâches qu'ils souhaitent apprendre. Une question assez naturelle est de savoir si l'assignation des participants à des sous-modèles particuliers pourrait être automatisée. Le problème est le suivant : nous avons différents agents qui pourraient vouloir collaborer pour atteindre un certain objectif (dans notre cas, l'apprentissage machine) et leur intérêt pour cette collaboration dépend d'avec qui ils vont effectivement collaborer (ils souhaitent que leurs collaborateurs aient des objectifs proches et soient suffisamment nombreux). Les agents sont des agents économiques rationnels et ils peuvent rejeter la recommandation si celle-ci n'est pas optimale pour eux, en termes d'intérêt individuel. Cela correspond à la classe de problèmes connus sous le nom de *jeux hédoniques* [DG80], un sous-domaine de la théorie des jeux en lien avec l'économie.

Résoudre des jeux hédoniques signifie proposer aux agents une affectation à des groupes (coalitions) telle que les agents n'ont aucun intérêt (individuellement) à dévier de cette affectation (quitter leur groupe et/ou en rejoindre un autre). Résoudre les jeux hédoniques d'une manière générique, c'est à dire proposer une solution d'équilibre dont aucun agent n'a intérêt à dévier, est mathématiquement impossible [AS16]. À notre connaissance, toutes les solutions existantes se concentrent sur des sous-classes du problème [Saa+10; Ang+18; ZCY17; Mei+19]. Ces solutions sont, ou bien centrées sur des applications auxquelles la recommandation de coopération ne peut être réduite, ou bien font des assertions mathématiques (existence d'un certain type d'équilibre) qui ne sont, en général, pas valides pour la collaboration en apprentissage machine. Nous proposons donc d'abandonner l'existence d'un équilibre et fournissons une heuristique qui produit des solutions approximatives proches d'un équilibre.

La notion centrale dans les jeux hédoniques est l'*utilité* : l'intérêt d'un agent donné pour la coalition proposée. Pour l'apprentissage machine collaboratif, nous avons développé une formule d'utilité basée sur deux variables : une distance entre les agents dans un certain espace métrique et le nombre d'agents par groupe. Cela transforme en fait notre problème en un problème de partitionnement, que nous avons donc décidé de résoudre avec un algorithme de partitionnement. À notre connaissance, aucun algorithme de partitionnement existant n'est conçu pour résoudre un tel problème, nous devons donc en créer un nouveau. Nous avons pris comme base le fameux algorithme des k-moyennes [Llo82]. Cet algorithme se concentre uniquement sur la distance, mais nous avons pu adapter sa conception basée sur une optimisation locale gloutonne à notre fonction d'utilité. En utilisant deux boucles imbriquées, contrairement à l'algorithme des k-moyennes qui n'en a qu'une, nous avons réussi à créer un algorithme de partitionnement qui optimise la fonction d'utilité bivariée que nous avons définie.

Notre évaluation a montré de bonnes performances (comparées à la référence) de notre algorithme de partitionnement, tant sur des données artificielles que réelles.

Ce travail a été présenté à la « 9th International Conference on Networked Systems (NETYS 2021) » sous le titre « AUCCCR: Agent Utility Centered Clustering for Cooperation Recommendation » [BFT21a].

III.3 Application effective de la formation de coalition à l'apprentissage machine distribué multitâche et exploration des implications pratiques de la vision associée

Une fois notre système d'apprentissage machine distribué multitâche créé et notre algorithme de partitionnement testé sur des problèmes synthétiques ainsi que sur des problèmes réels de collaboration, la dernière étape est l'application effective de notre algorithme de partitionnement à notre système d'apprentissage machine distribué multitâche, montrant ainsi qu'il permet une assignation automatique des modèles profitable. Pour rendre cette application effective possible, nous proposons une méthode pour associer un vecteur réel à chaque apprenant (sans quoi notre algorithme de partitionnement ne pourrait fonctionner) sans divulguer ses données, en se basant sur une séquence de tests commune, effectuée par chaque apprenant sur un réseau de neurones local pré-entrainé (courte séquence d'apprentissage).

Nous avons exploré plus avant la notion d'intérêt individuel, montrant comment elle interagit avec l'apprentissage machine collaboratif et révélant les limitations des configurations les plus « collectivistes ».

Nos résultats montrent que notre système d'apprentissage machine distribué multitâche et notre algorithme de partitionnement fonctionnent ensemble comme prévu. Nous voyons également que notre système respecte les intérêts individuels des utilisateurs (les gains en termes d'intérêt général ne se font pas au prix de pertes individuelles pour certains utilisateurs).

IV Conclusion

Dans cette thèse, nous nous sommes attaqué au problème de l'apprentissage machine distribué multitâche et avons fourni une solution effective à ce problème présentant de bonnes caractéristiques que nous pouvons considérer comme représentant désormais « l'état de l'art » en la matière.

Nous espérons maintenant que cette solution sera reprise par des acteurs industriels pour une application pratique et que ce type d'approche deviendra usuel dans le domaine de l'intelligence artificielle.

V Publications

L'auteur de cette thèse, en collaboration avec ses encadrants, a publié les articles revus par les pairs suivants (les deux premiers font partie de cette thèse, les deux autres concernent des travaux plus anciens) :

- **Simple, Efficient and Convenient Decentralized Multi-Task Learning for Neural Networks [BFT21b]:** Cet article présente notre système d'apprentissage machine distribué multitâche. Présenté au *19th Symposium on Intelligent Data Analysis (IDA 2021)*. Auteurs: Amaury Bouchra Pilet, Davide Frey et François Taïani.

- **AUCCCR: Agent Utility Centered Clustering for Cooperation Recommendation [BFT21a]:** Cet article présente notre algorithme de partitionnement pour la recommandation de coopération. Présenté à la *9th International Conference on Networked Systems (NETYS 2021)*. Auteurs: Amaury Bouchra Pilet, Davide Frey et François Taïani.

- **Robust Privacy-Preserving Gossip Averaging [BFT19]:** Réalisé avant cette thèse, cet article présente un algorithme de moyennage épidémique privé, qui permet à notre système d'apprentissage distribué multitâche de protéger la vie privée dans un contexte décentralisé. Présenté au *21st International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2019)*. Auteurs: Amaury Bouchra Pilet, Davide Frey et François Taïani.

- **Foiling Sybils with HAPS in Permissionless Systems: An Address-based Peer Sampling Service [BFT20]:** Réalisé avant cette thèse, cet article s'intéresse au problème des attaquants essayant de submerger un système décentralisé ouvert avec des pairs malveillants. Présenté au *25th IEEE Symposium on Computers and Communications (ISCC 2020)*. Auteurs: Amaury Bouchra Pilet, Davide Frey et François Taïani.

Chapter 1

Introduction

I Introduction

Machine learning is one of the most important and active fields in present computer science. Its applications range from analyzing large amount of data in datacenters to providing artificial intelligence-based end-user services on small personal devices [Haz+18; Sae+17]; applications which enjoy a rapid growth and major industrial interest.

Most current machine learning systems, however, are still using a mainly centralized design. Even when the final application is to be delivered in several systems, potentially millions (and even billions) of personal devices, the learning process is still centralized in a large datacenter. This can be an issue if the training data is sensitive, like private conversations, browsing histories, or health-related data [Che+17].

A solution to this is to make the training process happen on user-owned devices rather than in datacenters. This would also allow offloading costly computation tasks to users' devices, rather than having to handle them on dedicated infrastructure. These advantages have led industry leaders such as Google [Bon+19] and Facebook [Li+20] to start working on *federated* setups, were a central server organize a collaborative computation task performed on multiple distant devices.

Currently, the best known class of machine learning algorithms are most probably neural networks. While there are other kinds of systems, many remarkable applications are relying on them. They allow computers to effectively compete with humans in tasks that would be very difficult or impossible to solve with less advanced systems. Their drawback are complexity and hardly predictable behavior, which makes advanced applications, including distributed one, more complex than with simpler systems.

Another important challenge for modern machine learning is multi-task learning. For applications such as handwriting recognition or speech recognition, the precise task of recognizing an individual user's handwriting or speech is different. And some users may also be closer than other, notably depending on their location. Multi-task machine learning has already received interest in a centralized context, but distributed approaches lack. We would like distributed machine learning to be able to learn similar but different tasks, and ideally, exploit different levels of similarity between users to further optimize the process.

While such optimization (grouping users for learning) may be doable manually in some contexts, that would be more difficult in others. For example, while for speech recognition, one may use location data to set up an optimized process (grouping learners by location), for handwriting recognition, classifying hand-writers (depending on their scripting "style") is not an obvious process. A good distributed multi-task machine learning system should come with a proper way to evaluate and exploit varying levels of similarity between tasks.

II Problem & challenges

To enable the above mentioned applications we need a distributed multi-task machine learning system that will have to meet several requirements: large range of application, training data kept local, effective parallelism, exploit varying level of similarity between learners, flexibility on setup (federated (with a server for coordination) or fully decentralized). Following are the objectives that are not meet by currently existing solutions, and we are giving ourselves for this work.

We need a system that works with various kinds of models and notably with neural networks. To support this genericity, we only assume in the rest of this work that the learned models can be reduced to (finite) collections of real ($\in \mathbb{R}$) values. This system must enable distributed learning without learners needing to disclose their private data, with learners being able to all learn at the same time (parallel computation). Our system must be able to allow learners to learn similar but different models, to solve similar but different tasks, at the same time, collaboratively. It must be able to take varying level of similarity between tasks into account.

Our system must be able to work with the two classes of distributed setups: *federated* and *decentralized*. Federated setup have a central server coordinating (and/or aggregating) the work of several clients. In a distributed setup, clients become peers that communicate with each other and collaborate without coordination of a central server.

We also need to provide an additional system, that would be able to automatically manage the varying levels of similarity between tasks, configuring the learning system accordingly. This system should be able to determine which learners have closer tasks so that they will collaborate more strongly during the learning process.

III Contributions

To address the above challenges, we propose three contributions in this thesis:

- Distributed multi-task learning proper [Chapter 3]
- Coalition formation applicable to distributed multi-task learning [Chapter 4]
- Effective application of coalition formation to distributed multi-task learning and exploration of the associated vision's practical implications [Chapter 5]

III.1 Distributed multi-task learning proper

Our first contribution is our distributed multi-task learning system itself. Our system is based on *averaging*, a commonly used method in distributed learning. The idea is simple, to allow collaborative learning of a model consisting in a collection of real parameters, learners work separately and, periodically, a common model is build by taking the average of the values learned by individual learners, which take this common model as base for the next cycle. The advantage of this method is that it should work with any kind of model based on real ($\mathbb{R}$) values, independently of the learning algorithm used, as long as it only involves changing the values, not adding and removing values; in the present work, though, we focus on neural networks. It also can work either in a federated setup (models averaged on a central server) of in a decentralized setup (using gossip averaging [JMB05]).

By itself, this method does not allow multi-task learning. To enable it, two kinds of solutions are possible (we introduce this terminology): *soft sharing* of all parameters and *hard sharing* of some parameters. Soft sharing means that, rather than having all learners converging on a common model, they are allowed to slightly diverge from it. Hard sharing (of some parameters) means that the common model does not cover the whole model learned, some parameters are specific to each learner.

We chose the second solution: hard sharing of some parameters. The reason behind this choice is that neural networks commonly have a "functional" structure: networks are divided in parts, each being

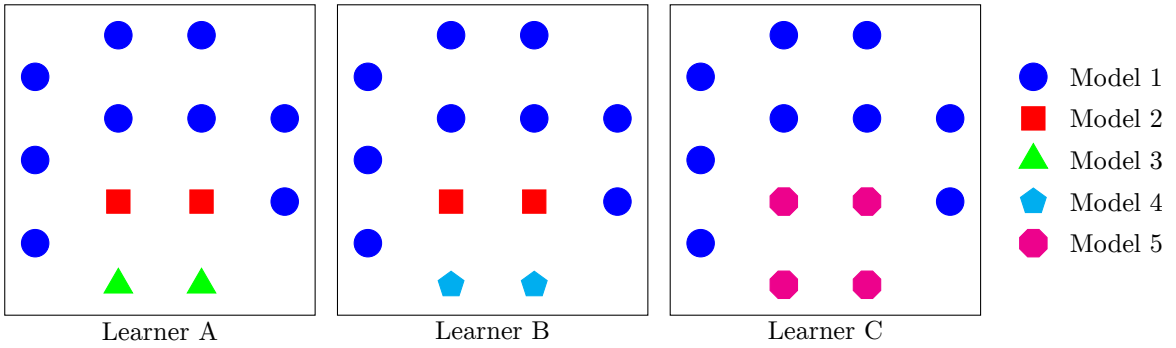


Figure 1.1: 3 learners with 5 partial models integrated in learners' neural networks geometry

supposed to have a more or less specific function. For example, in artificial visions, neural networks are commonly divided into layers, the first of which being supposed to recognize simple shapes and the last one to interpret those shapes. Unlike partially sharing all parameters, completely sharing predefined parameters allows engineers to select which parts should be shared and which should be kept individual. Notice that neural networks usually include significant amount of equivalent neurons: same input values and output used as input by the same neurons. This allows to use hard sharing without knowing exactly which parts must be shared, putting some neurons of a part in a common model and keeping the rest individual. With such a configuration, the learning process itself should be able to combine common and individual elements optimally (to the extent neural networks training algorithms can optimize), the learning process itself setting the weights appropriately.

We formalize this system with the notion of *partial model*. Partial models can be seen as bricks that can be used to build a complete model for each learner. Those models are mapped to specific portions of each learner's neural network, which allows to select the most relevant parts to be shared (concretely, averaged) if previous knowledge about that is available (else, one may just set portions of each part to be shared and let the training algorithm set the weights appropriately). Partial models can be shared by all peers (*global model*), be specific to a single learner (*local model*) or shared among a specific subset of learners (*semi-local model*). Figure 1.1 show an example of three learners with five partial models implemented at specific places in the neural networks' geometry. Model 1 (blue circles) is global, model 2 (red squares) is semi-local and models 3 (green triangles), 4 (cyan pentagons) and 5 (magenta octagons) are local.

We conduce experiments on various dataset which show that our design works as expected. In these experiments, we tested different configurations for our system, allow us to see which configurations give the best results. Notably, we showed that the flexibility allowed by our semi-local models can be exploited to improve results.

This work has been presented at the 19th Symposium on Intelligent Data Analysis (IDA 2021) under the title "Simple, Efficient and Convenient Decentralized Multi-Task Learning for Neural Networks" [BFT21b].

III.2 Coalition formation for automated model assignation

Our multi-task learning approach assumes that participants have already been grouped together according to the learning tasks they seek to perform. A natural question is whether this assignation of participants to particular sub-models can be automated. The problem is the following: we have several agents that may want to collaborate towards some goal (in our case machine learning) and their interest in this collaboration depends on whom they will actually collaborate with (they want their collaborators to have close objectives and be numerous enough). The agents are rational economic agents and may reject the recommendation if it is not optimal for them in terms of individual interest. This corresponds to a class of problems known as *hedonic games* [DG80], a subdomain of game theory

linked with economics.

Solving hedonic games consists in proposing to agents an affectation to groups (coalitions) such that agents have no interest (individually) to deviate from this affectation (leaving their group and/or joining another). Solving hedonic games in a generic manner, in terms of proposing an equilibrium solution which no individual agent has interest to deviate from, is mathematically impossible [AS16]. To the best of our knowledge all existing solutions focuses on subclasses of the problem [Saa+10; Ang+18; ZCY17; Mei+19]. These solutions are either centered on applications to which collaboration in machine learning can not be translated, or assume mathematical conditions (existence of some kind of equilibrium) that collaboration in machine learning does not fulfill. What we propose then is to abandon the existence of an equilibrium and provide an heuristic that outputs approximate solutions close to an equilibrium.

The central notion in hedonic games is *utility*: the interest an agent has for the proposed coalition. For collaborative machine learning, we developed an utility formula based on two variables: a distance between agents in some metric space and the number of agents per group. This actually makes our formalism a clustering problem, which we decide to solve using a clustering algorithm. To the best of our knowledge, no existing clustering algorithm is designed to solve such a problem, so we need to create a new one. We take as base the well known k-means algorithm[Llo82]. This algorithm focuses exclusively on distance, but we found that its greedy local optimization-based design could be adapted to our utility formalism. Using two imbricated loops, unlike k-means which uses a simple loop, we managed to create a clustering algorithm that optimize the utility function we defined.

Our evaluation shows good performance (compared to baseline) of our clustering algorithm on both, artificial and real data.

This work has been presented at 9th International Conference on Networked Systems (NETYS 2021) under the title "AUCCCR: Agent Utility Centered Clustering for Cooperation Recommendation" [BFT21a].

III.3 Evaluation of our automated model assignation system and exploration of agents individual interest

After creating our distributed multi-task machine learning algorithm and testing our clustering algorithm on synthetic clustering problems and real collaboration-related datasets, the final step done is the actual application of our clustering algorithm to our distributed multi-task machine learning system, showing that it allows profitable automated model assignation. To allow this actual application, we propose a way to associate a real vector to each learner (required for clustering to work) without disclosing their data, relying on a common benchmark ran by each learner with a locally trained (short pre-training) neural network.

We further explored the notion of individual interest, showing how it interacts with collaborative machine learning and revealing the limitations of the more "collectivist" configurations.

Our results show that our distributed multi-task machine learning algorithm and our clustering algorithm work together as expected. We also see that our system respects learners individual interest (its gains in terms of general interest do not come at the cost of individual losses for some learners).

IV Plan

We start this thesis with a literature review in the relevant fields in Chapter 2, introducing neural networks, presenting their history and the modern developments related to our work as well as the field of clustering. We the present our distributed multi-task machine learning system and its evaluation in Chapter 3 on artificial tasks and three real datasets: MNIST [LeC+98], FEMNIST [Cal+19] and VSN [DH04], showing its efficiency and the interest of its flexible design. Our clustering system and the associated experiments are presented in Chapter 4, showing the superiority of it against baseline

on both, artificial data and two real datasets: BuddyMove[RA14] and Wholesale [Fer11]. Finally, the combined evaluation of our distributed multi-task machine learning system and our clustering algorithm on a real application (MNIST and VSN datasets) is shown in Chapter 5, as well as an analysis of how different learners are impacted with our distributed multi-task machine learning system (more profitable for some than others).

V Publications

The author of this thesis, in collaboration with their advisors, produced to following peer-reviewed papers (the two first are part of this thesis, the two others are previous work):

- **Simple, Efficient and Convenient Decentralized Multi-Task Learning for Neural Networks [BFT21b]:** Which presents our distributed multi-task learning system. Presented at the *19th Symposium on Intelligent Data Analysis (IDA 2021)*. Authors: Amaury Bouchra Pilet, Davide Frey and François Taïani.
- **AUCCCR: Agent Utility Centered Clustering for Cooperation Recommendation [BFT21a]:** Which presents our clustering algorithm for collaboration recommendation. Presented at the *9th International Conference on Networked Systems (NETYS 2021)*. Authors: Amaury Bouchra Pilet, Davide Frey and François Taïani.
- **Robust Privacy-Preserving Gossip Averaging [BFT19]:** Done previous to this thesis itself, this paper presents a private gossip averaging algorithm, which empowers our distributed multi-task learning system to protect privacy in a decentralized context. Presented at the *21st International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2019)*. Authors: Amaury Bouchra Pilet, Davide Frey and François Taïani.
- **Foiling Sybils with HAPS in Permissionless Systems: An Address-based Peer Sampling Service [BFT20]:** Done previous to this thesis itself, this paper tackles the issue of attackers trying to flood open decentralized systems with malicious peers. Presented at the *25th IEEE Symposium on Computers and Communications (ISCC 2020)*. Authors: Amaury Bouchra Pilet, Davide Frey and François Taïani.

Chapter 2

Field review

The present work is mostly part of the machine learning field, which is a major and rich field. In this chapter we present this field and, more specifically, neural networks. We explore its history and recent advances, particularly distributed and multi-task learning. Also presented is the field of clustering and how hedonic games, a subject of game theory, can link those two fields.

I Introducing Neural Networks

Artificial neural networks are designed to reproduce the behavior of a real brain. They consist of *formal neurons*. They are many ways to define a formal neuron, but we are going to give one that is representative of how formal neurons are used in modern systems.

You may refer to Figure 2.1 for a graphical representation of what we are going to describe. A formal neural has several inputs x_i. They are usually real numbers (often between 0 and 1 or -1 and 1) but may also be binary. They also have a unique output y (when you see multiple outputs for a neuron, they are just "copies" of that only output sent to various receivers). Neurons also have an *activation function* f. The output y of a formal neuron is the value of the activation function evaluated on the sum of all inputs x_i weighted by a certain input weight w_i. The function f can also have additional parameters, $\mathbf{p}$, commonly a bias, a value added to the sum before computing the function. Commonly used activation functions include sigmoids ($f(x) = \frac{1}{1+e^{-\alpha x}}$), the hyperbolic tangent ($f(x) = \tanh(x)$) and simple binary thresholds ($f(x) = \mathbb{1}_{x \geqslant \theta}$).

Weights w_i and parameters of the activation function p are learned using some training algorithm. The (by far) most commonly used nowadays are based on Stochastic Gradient Descent with Back-propagation, in which weights are adjusted based on the errors (backpropagated through the network from the output) the network makes on training cases for which its expected output is known. This will be discussed in more detail in the relevant place of the historical section (II.3.a).

To perform actual computations, formal neurons have to be combined is large networks (usually more than several hundreds, potentially much more). They are various possible layouts. Figure 2.2 presents a very small neural network with a classical multi-layer architecture. In this network, the first layer (neurons to the left) are not actual neuron (with an activation function) but simple inputs. The output of the network is the output of the neuron (their may also be several) to the right (which is an actual formal neuron).

II History

II.1 First works

Work on artificial intelligence began with the study of natural intelligence.

In 1943, Warren McCulloch and Walter Pitts proposed a mathematical modelization of animal

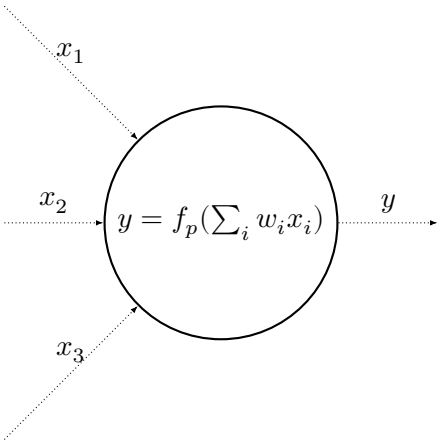


Figure 2.1: A formal neuron

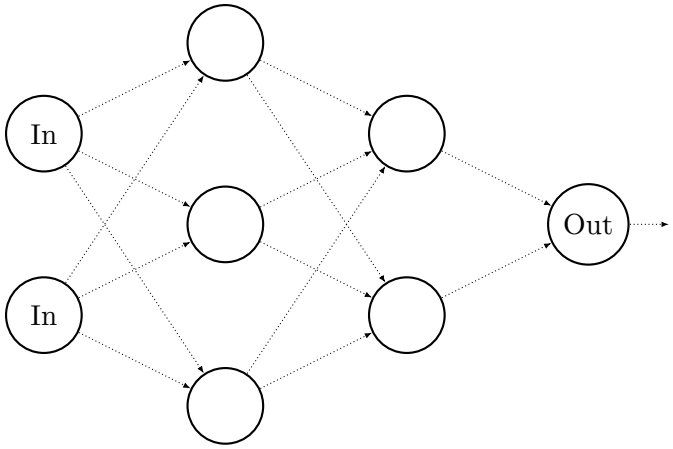


Figure 2.2: A neural network

brains [MP43]. At this time, however, the focus was on modeling and studying natural brain rather than creating artificial intelligence, those researchers being more focused on confronting their work with experimental biological reality than considering their application to computing [Let+59]. These researchers were the first to introduce the idea of formal neuron, but with the goal to study actual neurons rather than to perform real computation.

In the same spirit, Donald Hebb formulated a learning rule that has been later used for machine learning in a neuroscience context [Heb49]. This learning rule is based on the idea that when neurons get activated at the same time, they would "link" together and (a connection between them would be created, or the weight of an existing connection will be increased), latter, one of those neurons activating would tend to activate the other. While this rule is not the most common in artificial neural networks today, it is still considered by some researchers in machine learning [Haw+19] and is used (secondarily) in this work[1].

We can consider the *Perceptron*, introduced by Frank Rosenblatt in 1958 [Ros58] to be the first draft of a "modern" neural network, in the sense that derivatives of it are still very common today. This old perceptron, however, was still very limited. Other neural network models proposed in this period [Roc+56] did not encounter significant success.

The research on neural networks mostly stalled in the late 60s, due to the limited capabilities of old neural networks models, not being able to compute non-linear functions as simple as XOR [MP69].

[1] Associative Neural Network presented in Section IV.2

Also, even the best computers of this era were far too limited to effectively perform computations considered very basic in current machine learning.

II.2 The Dark Age

During this dark age for machine learning in the West, sometimes called the "Winter of Artificial Intelligence", Vladimir Vapnik and Alexey Chervonenkis, working in the "Motherland of Statistics": the Soviet Union, established the theoretical basis of learning a predictive function from a data sample, initiation what will become *Statistical Learning Theory* [VC71; HTF09]. However, despite being translated to English, their work was largely ignored in the context of the Cold War. This work was rediscovered in the West after the end of the war.

In the same period in the West, Paul Werbos developed the idea of *backpropagation* of errors in his PhD thesis [Wer74] (not for neural network, but for statistical models used in social science), which was going to become a key element of neural networks training after their renewal.

In the 80s, several models of neural networks where proposed, notably *Hopfield networks* [Hop82] and *Boltzmann machines* [AHS85; Smo86] but none achieved significant success. Hopfield networks, though, are significant because they introduced the concept of *Recurrent Neural Network (RNN)*. The specificities of these networks is that they have loops in their structure (in the simplest case, a neuron taking its own output as input for a next iteration). Those loops allow RNNs to process sequences of elements, keeping information from previous elements in "memory", unlike other neural networks. While they were too limited for real applications, later improvements to the concept allowed it to meet success.

Hopfield networks are an attempt to allow storing information in a neural network. The original principle dates back to 1974 [Lit74]. In these network, each neuron is associated with a bit and all neurons output are connected to all others input. Weights are learned with an associative rule, based on real neurons, "neurons that fires together bind together". This would, in theory, allow Hopfield networks to memorize bit patterns, this never encountered practical application though (poor performance a lack of viable application).

Boltzmann machines are designed to learn a probability distribution. Unlike Hopfield networks, they include hidden neurons (neurons not corresponding to bits of data to learn). They may, in theory, learn any probability distribution, but in practice, outside the smallest (and not really useful) ones, are computationally impossible to train. Paul Smolensky came with a solution with his *Harmonium* (nowadays, the term *Restricted Boltzmann machines* is used). In this kind of machines, neurons are strictly divided between visible and hidden and only inter-class connections, not intra-class ones, are allowed (making the network a bipartite graph). This enabled more efficient training but was not enough to get these machines to large application success.

II.3 Renewal

In the late 80s, however, initial works on neural network models still in use nowadays (in a form close to what is in use today) began. In these years, the focus of research had shifted from reproducing biological neurons to applying known mathematical methods to neural networks.

II.3.a Stochastic Gradient Descent

The backpropagation method was combined with the classical Gradient Descent used in numerical analysis, already known in the 19th century [Cau47]. This method, when used with data selected randomly from a training data set, will become the most common method used for training neural networks in current applications: Stochastic Gradient Descent with Backpropagation (SGD-BP) [RHW86; WL90] (though modern applications of this method often include some variations [GBC16]). The main principle of Gradient Descent is to compare the actual output of a neural network to the excepted output for some training case and modify the networks weights to make its output closer to the expectation (following the gradient). Denoting the parameters learned by the network by m, the

difference between the expected output and the actual output (loss) by $L(m)$ (estimated by evaluating a batch of training cases) the generic formula for updating parameters with gradient descent is $m(t+1) = m(t) - \lambda \nabla L(m)$. In this formula, λ is a parameter call the *learning rate*. If it is too low, the convergence will take too long, if it is too high, convergence may not happen. SGD adds the idea of randomly picking training cases from a pool. This is commonly implemented with *mini-baches*, which are sets of elements randomly selected from the whole pool of training data (training set). In SGD, the training process is done on a succession of such mini-batches, by opposition to classical Gradient Descent, where a succession of batches, consisting of the full training set, is used. Neural networks need a large amount of samples for training, but while it is necessary to have numerous different samples, processing the same sample several times, with other different samples in between, is also beneficial. Finally, backpropagation (applied in this context) consist in propagating the gradient value from output to input over the multiple layers of a multi-layer neural network (or over the multiple iterations of a recurrent neural network), effectively allowing using gradient descent to train modern neural networks.

The 90s saw interest in neural networks works grow gradually. We notice that, as early as in 1991, the concept of "transfer learning", which is the origin of works in both distributed machine learning and multi-task machine learning, was introduced [PMK91] (even though most researchers in these fields seem to have forgotten this pioneering work, maybe due to the lack of outcomes with the limited neural network models and computing power of the early 90s).

The renewal of neural networks was complete in 1998 with Yann LeCun's work with the *Multi-Layer Perceptron (MLP)* [LeC+98]. In this kind of neural network, neurons are organized in successive layers from the input (data to classify) to the output (classification of data). In each layer, each neuron is connected to all neurons of the previous layer. Using this kind of neural network in conjunction with SGD, they managed to achieve very good accuracy in handwritten digit recognition, opening the world of real-life applications to neural networks. MLPs can be applied in many situations involving the interpretation of multiple real-valued inputs; for example when analyzing Fourier transformed acoustic or seismic waves [DH04].

II.4 Variants

In this history, we have focused on important direct-line ancestors of the work presented in this manuscript but other branches of the research on neural networks deserve to be mentioned, either because, while not used in this work, they are important in model artificial intelligence or because they inspired some secondary elements of this work.

A very well known type of neural network not directly liked with the present work is *Long Short-Term Memory (LSTM)* [HS97; Ger99; GSC00], which, unlike MLP, focuses on the analysis of sequences and next element prediction. LSTM is a Recurrent Neural Network (RNN) able to learn with Stochastic Gradient Descent with Backpropagation. While SGD-BP has always been technically applicable to RNN [RHW86], it faced a serious issue called *vanishing gradient*, with the loops in the network causing the gradient to be lost over iterations due to numerous multiplications with < 1 values. LSTM's architecture solved the issue, introducing "Constant Error Carousel", additional values only present to keep errors over iterations by reducing the number of multiplications, allowing gradient to propagate over loops iterations. LSTM is probably the most used kind of neural network when it comes to prediction in sequences.

We should also mention *Convolutional Neural Netowrks (CNN)* [Fuk80], based on an idea that some structure should repeat at different position in a layer of neurons and often used with MLP. Being closer to how actual vision works [Lin20]; this kind of neural networks met success in image recognition. MLP used in modern applications to image recognition usually have several convolutional layers as first layers.

We would also mention Hierarchical Temporal Memory [Haw+19] which is an example of modern neural network research still based on the old principles of using biology as a source of inspiration

and, while not being a mainstream approach, achieving good results in moving objects recognition. A significant particularity of this kind of neural network is its use of *Sparse Distributed Representation (SDR)* [AH15] to represent data. Unlike what is usually done in computers, were we use numerical indices, or a specific bit for each value, SDR represent each value with a sparse binary vector. Values having more "in common" (closer concepts) have more similar vectors. This is inspired from how human brain actually represent concepts. SDR inspired a data representation used in a few of our experiments[2].

III Distributed learning

With the development of both machine learning and distributed systems, distributed machine learning saw significant work in the past decade. In this work, we use the term *distributed* to refer to situations involving computers connected by a large scale network (typically Internet). For the more classical case of a workload done on several computers in a single datacenter (called "distributed" by some authors), we use the term *parallelized*. Distribution can address two issues of machine learning. The first is data privacy; in applications involving, for example, health data [Che+17], people involved may not want such sensitive data to be sent to a third party over the Internet. The second is the large amount of computing power required for training. Distribution allows to spread the computational process over multiple infrastructures, improving performance [Li+14]. For these reasons, industry leaders like Google [Bon+19] or Facebook [Li+20] got involved in research on distributed machine learning.

III.1 Definitions

For large-scale applications over the Internet, two kinds of approaches of distribution can be distinguished. Figure 2.3 illustrates these two concepts, *federation* and *decentralization*, compared to centralization. Ω designate elements in charge of coordinating the system, Σ elements in charge of pure computation tasks. In a *federated* setup, most computation is done locally by peers but a central server synchronizes the work of all peers. This kind of setup is considered, for example, in [Kon+16]. In a *decentralized* setup, in which peers work together without a coordinating central server (self-organization). The approaches proposed in [Bel+18] is of this class. Not all distributed learning methods are necessarily bound to one of these two kinds of setups. Methods made for one situation may sometimes be trivially adapted to the other, and sometimes, methods designed for parallelization in a data-center can be used in a distributed setup over the Internet.

III.2 Challenges

Distributed computing in general implies multiple issues which researchers interested in this field have to solve. The central challenge is, of course, to distribute the computation. Proposing a solution that also works without a central sever (decentralized rather than federated) is a significant extension of this challenge.

It is also important that the proposed approaches actually benefit of advantages offered by distributed computing. For example, it is not very interesting to propose a distributed algorithm which does not effectively parallelize computation over the network. One may also like that distribution allow them to keep their data private, at least to some extent. While this is not a key element of distributed computing, it is something that it may provide over centralized systems.

Finally, distributed computing comes with its own set of problems which researcher wants to tackle. Issues such as latency (and jitter), reliability (of the network) and large amounts of data to transfer are typical problems that affects distributed specifically. Not even talking about malicious actions.

[2] Associative Neural Network presented in Section IV.2

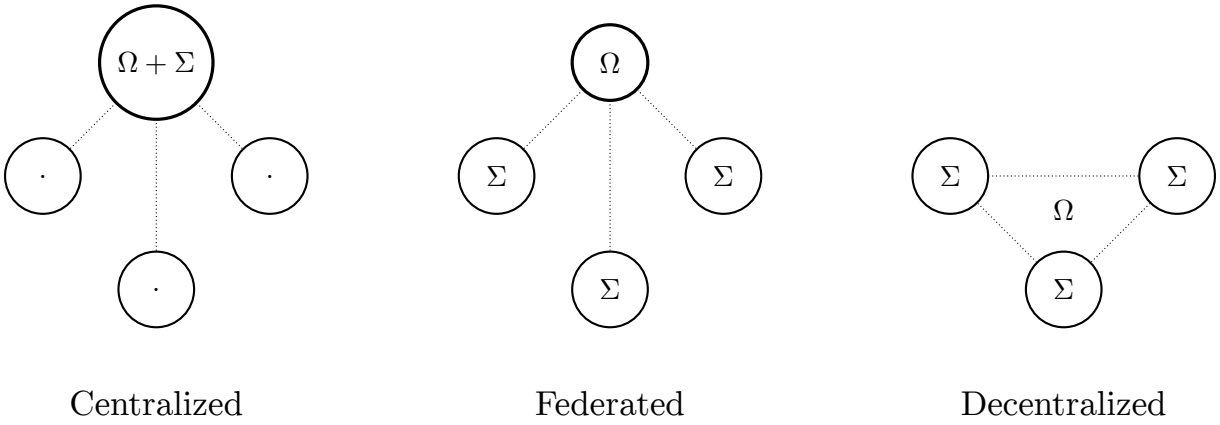


Figure 2.3: Illustration of the concept of centralized, federated and decentralized architectures. Ω indicates coordination tasks, Σ computation tasks.

III.3 Approaches

Distributed learning as been studied in the context of linear models in [OHJ12]. This approach proposes a fully decentralized learning method for linear models (gossip algorithm [Dem+87]), relying on model averaging. The idea of model averaging is that, when you have multiple learners trying to learn the same set of parameters and those parameters are averageable values (typically, real number ($\mathbb{R}$)), you can make those learners learn together by periodically averaging the values they are learning.

For neural networks, parallelization of Stochastic Gradient Descent, the most common learning algorithm for neural networks, was proposed in 2011 [Rec+11]. At this time, the idea was more to provide parallelization in a large computing infrastructure rather than on the Internet. Still, this approach had the significant advantage, over previous propositions, that it did not require costly synchronization methods, making it more suitable for larger-scale application. However, being based on communicating gradients it was limited to gradient-based training and could not be applied to other methods.

Most approaches focus on Stochastic Gradient Descent, since it is by far the most commonly used training algorithms, but not all actually require the use of a gradient. Some approach rather rely on communicating models, which allow them to be applied to other learning algorithms (such as associative learning). The *Federated Averaging* approach proposed in [McM+17], could work with other learning algorithm, being based on averaging the learned parameters (models) rather than gradients.

Some researchers also tried to improve some properties of previous approaches. For example, the authors of [Smi+16] and those of [Kam+18] focused on reducing communication overhead. They achieved this by replacing strictly periodic synchronization by adaptive synchronization, communicating only when peers have sufficiently diverged from the common model. A system called HeteroFL [DDT20] has also been proposed to manage setups in which some, but not all, devices may have stronger computation or communication limitations. In this system, some (limited) clients may use simplified models, with fewer parameters, to reduce computation and communication costs[3].

It is actually possible to define an approach that can work in both kinds of setups, *federated* and *decentralized*, under the condition that this approach relies on some primitive that can be implemented for both kinds of setup. This is the case of averaging, which can be done on a central server or in a decentralized fashion with *gossip averaging* [JMB05]. This is the choice we made.

[3]This work uses the term "local model" to designate each client's instance of the generic model the system is learning and "global model" to designate the common model managed by the server. This terminology shall be distinguished from the (anterior) "local model", "semi-local model" and "global model" terminology we use in this work, which designates portions of a complete model that are implemented by either a single, a subset or all clients/peers.

IV Multi-task learning

Multi-task learning comes rather naturally when thinking about distributed learning. If we have different users learning collaboratively, then maybe the tasks they want to learn are not exactly the same. For example, if we consider handwriting recognition, everyone has their own script and the optimal model to recognize it differ from people to people. Multi-task learning has first been considered outside a distributed context, when a single system is learning similar but different tasks. Let us start by exploring this area before coming back to distributed.

IV.1 Non-distributed multi-task learning

The idea between multi-task learning is that learning several similar but different tasks at the same time may be more efficient than learning them separately. This allows each task to benefit from the knowledge obtained from learning other tasks. For example, when training a neural network to perform some optic recognition task, knowledge from other optic recognition tasks may be useful, since even if the objects we want to recognize are not the same, all optic recognition tasks need to recognize some simple shapes. The FEMNIST dataset [Cal+19] is an example. Here, all tasks are character recognition but each task is associated with a specific hand-writer, with their own personal writing.

Learning multiple similar but different task with neural networks, in a non-distributed context, was first studied in 1991 [PMK91]. This approach divides a single network between a common part, used for all tasks, and several task-specific parts. This work can be considered as the beginning of transfer-learning, but this historical approach has largely been forgotten due to arriving "too early", before the real take-off of neural networks.

While the term "transfer-learning" is commonly used to refer specifically to a method of transfer-learning involving changing only the last layers of a multi-layer neural network to adapt it to different tasks, transfer-learning actually covers a wide range of cases and numerous methods exist [Rud17]. In fact, transfer-learning even covers cases that are not multi-task learning [PY09], which is actually a subfield of (inductive) transfer learning. For example, reusing and adapting a model learned for a first task to a second task is (transductive) transfer learning.

Multi-task learning approaches are numerous[4], but we can identify two base methods on which those approaches are based. The first is hard-sharing of a subset of parameters, in which only some parameters are shared (commonly, averaged) and are kept identical for all tasks. A modern example of such an approach is [Lu+17], where the first layers of a multi-layer neural network are shared, while the others are task-specific. The other possibility is soft-sharing, in which no parameters are kept identical for all tasks but all parameters used for each task are allowed to differ a little from other tasks, but not too much [Duo+15], which can be presented (depending on approaches) either as tying separated models together or personalizing a common model. This can be done by integrating a cost for diverging from the common model in an objective function. It is possible to combine hard-sharing of some parameters with soft-sharing of other parameters, as in [Lon+17], where the first layer of a neural network are shared and the others are task-specifics but still tied with multilinear relationships.

IV.2 Distributed multi-task learning

At the time the present work began, solutions for distributed multi-task learning were very limited. An approach for distributed multi-task learning limited to linear models (not relevant for neural networks in their modern, non-linear, acceptation) was proposed in 2016 in [WKS16] (local computation of client-specific solutions followed by aggregation on a server and final adaption of local solutions) a work later improved in [Smi+17] (communication cost, fault tolerance). This method relies on including a cost for models divergence as part of the optimization process.

An extension multi-task learning to non-linear models was proposed by [CST18], relying on kernel methods to capture non-linearity, albeit limited to convex problems. Another method, focused on

[4] Some authors use the term "personalization", rather than multi-task learning, to describe their approach.

convex models and allowing decentralized learning, was proposed in [Bel+18]. In this paper, then authors proposed to use neighborhood relationships in a collaboration graph to allow peers with more of less close relationships in their tasks to learn collaboratively. The optimization process keeps peers' models close by integrating a cost for divergence; this cost being dependent on the strength of the relationship between each pair of peers. Their work has since been improved, allowing the collaboration graph itself to be learned on the fly [ZBT19].

IV.3 Concurrent works

These existing works, however, are not satisfying. Neural networks in general are not limited to linear or even convex problems. Since several teams were working on solving the, apparently important, issue of generic distributed multi-task learning for neural networks at the same time, other researchers published their own methods while the present work was being prepared. Here we give a list of those concurrent publications and explain how our proposition is, by some criteria, superior.

A significant advance in solving the problem of general distributed multi-task learning with neural networks is [CB19], relying on soft-sharing of parameters. In this paper the server and its clients are modeled by a Bayesian network[5]; during training, a shared model a client-specific models are updated to minimize the divergence[6] between the observations (training data), the shared model and each client's local model. This process is done in sequence, client after client[7]. Their work, however, is limited to a federated setup (not decentralized) and, moreover, they do not propose effective parallelization. Clients are supposed to take turns for running their learning process, which makes this proposal unscalable and missing the most significant interest of distributed learning.

Taking a different approach, the authors of [Jia+19] also tried to provide a solution to the problem of general distributed multi-task learning with neural networks. This approach is relying on personalization of a common model, a form of soft-sharing of parameters. It consists mainly in training a shared model with Federated Averaging[8] and then continue the training locally for each client; no actual novel algorithm is proposed. This approach, however, is still limited to federated setups, not decentralized ones, and did not allow, unlike ours, to define some peers as closest to others. Also, its more a demonstration that using a pretrained generic model as base for local training is an interesting workaround rather than a direct method for distributed multi-task learning. Such a method would be a serious barrier for online training.

These imperfect solutions close the list of approaches developed concurrently to ours and published earlier. Let us now talk about interesting related approaches published after ours.

After our prepublication, some researchers also (independently) published an approach that can be regarded as a significantly restricted version of ours (averaging of some, but not all, layers of a multi-layer neural network) in [Ari+19] (authors did not claim decentralized learning explicitly but, since our own implementation of decentralized learning is also applicable to this method, we consider it as decentralized).

Authors of [FMO20] modified the Federated Averaging algorithm to make output a shared model that could easily be personalized with a method similar to that of [Jia+19]. Actual simultaneous training of both, shared and personalized models, was finally achieved with [DKM20] (mixing a federated-learned model with a jointly trained local model using a $a \in [0, 1]$ parameter).

All this competition demonstrates the great interest of the community in the problem this work solves. Most significant distributed multi-task learning approached are summarized in Table 2.1. In this table, "flexibility" is a subjective notion referring to the ability of the proposed approach to be adapted to different use cases and tuned for optimal functioning in those cases.

[5] A way to model conditional dependencies between variables [Pea85].

[6] *Kullback–Leibler divergence* [Kul59], also called *relative entropy*.

[7] Authors themselves point the similarity of their work with a previous approach for non-distributed sequential learning of different tasks [Sch+18], even though they do not present this work as a source of inspiration.

[8] Authors include an additional fine-tuning phase with the Reptile algorithm [NAS18].

Work	Decentralized	Parallel	Non-linear	Generic NN	Direct multi-task	Flexibility
[WKS16]	✘	✔	✘	✘	✔	mid
[CST18]	✘	✔	✔	✘	✔	mid
[Bel+18]	✔	✔	✔	✘	✔	mid
[CB19]	✘	✘	✔	✔	✔	mid
[Jia+19]	✘	✔	✔	✔	✘	low
[BFT21b] (ours)	✔	✔	✔	✔	✔	high
[Ari+19]	✔	✔	✔	✔	✔	mid
[FMO20]	✘	✔	✔	✔	—[9]	mid
[DKM20]	✘	✔	✔	✔	✔	mid

Table 2.1: Distributed multi-task learning approaches in chronological order

V From distributed multi-task machine learning to hedonic games and clustering

In distributed multi-task learning in general (and in our approach in particular), it is useful to identify learners having closer tasks, as it allows to make them collaborate more closely, optimizing the learning process. While we can find come contexts in which such affectation is straightforward, for example with speech recognition, grouping users by language and/or regional variant, it is not always that obvious (we studied a such case in our experiments).

V.1 Hedonic games

An automated way to solve this grouping issue in the general case would be an important improvement. A possible approach is to adopt a formalism from economics and game theory. This formalism would consider peers as (rational) agents, with a utility function they want to maximize. The output of this function being dependent on which other agents an agent is grouped with. This kind of formalism is called a Hedonic Game [DG80].

While there are many algorithms targeted at hedonic games [AS16], there is no generic solution for them. It can actually be established that some hedonic games do not admit game-theoretically acceptable solutions (no Nash equilibrium, or similar equilibrium[10]) and that application to distributed multi-task machine learning can produce such unsolvable cases.

Some researchers considered using hedonic games to solve practical problems. For example, [Saa+10] worked on collaboration between roadside units in Intelligent Transportation Systems. [Ang+18] considered hedonic games in the context of Fog Computing. Application of these games to edge computing is also proposed in [ZCY17]. These games can also be applied to energy networks, as suggested by [Mei+19]. Each of these articles presents its own model and algorithm, to solve the specific problem they are considering. Those models being more constrained that what would be necessary (or, at least, desirable) for application to multi-task learning, they did not suffer from above mentioned equilibrium issue.

V.2 Solving hedonic games with clustering

A possible way to solve the considered problem is to rely on methods used for clustering, a subfield of data analysis. Clustering can be defined the following way: given a set of points in some space, find a pertinent way to put those points in groups (clusters) of relatively close points (the vagueness of this definition is a characteristic of clustering, and partly explains the variety of clustering algorithms) [Est02]. This problem can be related to community detection, which produces similar output

[10] Different formalizations of equilibrium exist for hedonic games, in which "options" available to players are not as clearly, and statically, defined as in other games, but all have the same philosophy: no player should have interest in leaving its group and/or joining a foreign group.

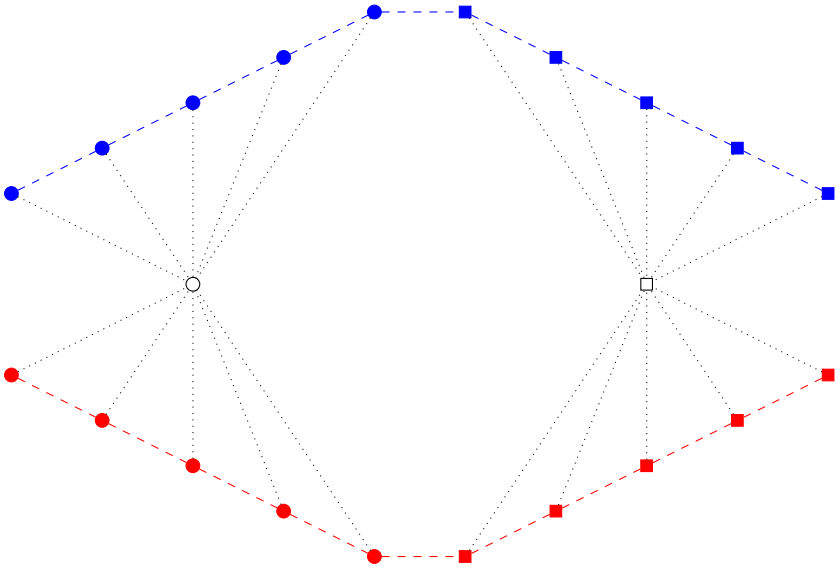


Figure 2.4: Centroid clusters (shapes) vs density clusters (colors)

but takes a (potentially weighted) graph as input (which, unlike clustering, does not fit the target use case [GN02; Blo+08]).

Let us now explore the field of clustering and the two main classes of approaches to this issue: centroid-based clustering and density-based clustering. Both define clusters with a notion of "closeness" but, for centroid-based clustering, the closeness is defined relatively to the barycenter of the cluster while for density-based clustering, it is defined relatively to neighbors. There is no "superior" vision, the reason to choose one system or the other is more philosophical: do you favor the global closeness of points or their closeness to their neighbors.

Figure 2.4 illustrates those two visions. Centroid clusters are identified by shape and density clusters by color.

V.2.a Centroid-based clustering

In this vision of clustering, the determining criterion is geometry. A cluster is a group of points determined by their distance to the barycenter of the group. This distance is the essential criterion to determine if a point is part of a certain cluster or not.

The problem of dividing a collection of elements in a metric space in k cohesive groups has first been formulated in 1956 [Ste56], though its current common name, *k-means* is more recent [Mac67].

The usual solution for this problem, Lloyd's Algorithm (commonly called *k-means* itself when used in the context of clustering), dates back to 1957, when it was designed as a method for pulse-code modulation (PCM, a usual way to numerically encode analog signals), even though its official publication happened in 1982 [Llo82]. The principle of this algorithm is to start from k random points as "cluster centers" and then assign all points to the closest center (for them). Cluster centers are then updated (as the barycenters of the clusters) and the process is iterated until convergence (or some quality criterion is met, to shorten the process). This process is summarized in Algorithm 1.

This algorithm received a notable improvement in 2007 when its initialization phase was modified to prevent the pathological solutions this randomized local-optimizing algorithm often produced. The new algorithm is refereed to as *k-means++* [AV07]. The principle of this new initialization process is to chose random starting points not uniformly but favoring locations far from already chosen points, to better distribute points in space.

A size-constrained variant of k-means was also proposed in 2014 [GCT14] and another in 2019 [Tan+19] but these are designed to produce clusters of some target size rather than integrating cluster size in the optimization process. This is an issue for an application where a small cluster of

Algorithm 1: k-means

```
function K-MEANS(P, k) is
    Randomly select k starting points in P
    Initialize the k groups g_i with the previously selected points
    repeat
        Compute barycenters b_i of groups g_i
        for p ∈ P do
            Add p to the group g_i for with the distance between b_i and p is minimal
    until No point changed group during last iteration
```

very close points is as acceptable as a large cluster of sparser points (this issue is clearly outlined by the graphs in [GCT14]). This also holds for density-constrained variants of k-means such as [DV05].

A significant limitation of k-means comes for its formulation: dividing a set of points into k clusters. k-means is not designed to leave alone points that are too far from all others. We would also note that all practical k-means algorithms are heuristics, converging to a local optimum. From a theoretical point of vue, it does not seem possible to provide a globally optimal solution to k-means in a practical way due to the k-means problem being NP-Hard [Alo+09].

Several authors tried to speed k-means up [Cha03] or to use an alternate metrics [BMS96]. The Hartigan–Wong [HW79] is an alternative algorithm which starts with random clusters and then greedily perform exchanges between cluster to optimize the same objective as standard k-means. While the algorithm's presentation is different, this algorithm remains very similar to standard k-means. None of those work solve the limitations we identified in k-means.

V.2.b Density-based clustering

Density-based clustering identifies clusters based on density. Essentially, these algorithms consider that regions where points are closer to each other than elsewhere are clusters. The essential criterion for a point to be part of a cluster is that its closer neighbors are also part of that cluster.

The historical density-based clustering algorithm is DBSCAN, proposed in 1996 [Est+96]. Another similar algorithm was rapidly proposed to fix a defect in DBSCAN: its inability to operate in contexts where there are multiple clusters of different density. This algorithm is called OPTICS [Ank+99]. DBSCAN and OPTICS are based on the same principle: starting from one point, identify how distant new points are from previously encountered points, and derive clusters from this. The main difference is that, while DBSCAN affects clusters directly based a predefined density, OPTICS orders points by distance and outputs a representation of the distances between points, from which clusters can later be extracted, usually based on a maximum distance. This is how OPTICS manages to fix DBSCAN's inability to find clusters of different density. Figure 2.5 illustrates this.

This approach does not suffer from the inability to leave points alone, but the geometry of clusters it outputs is not satisfying for applications where the distance to the barycenter is the essential criterion (this criterion is ignored by OPTICS). From the output of OPTICS, it is easy to eliminate smaller clusters, leaving points alone, but this does not allow determining if those points may have joined or larger cluster. Also, there is no straightforward way to take relative distances into account when eliminating smaller clusters.

Despite those limitations, some authors still tried to use a variant of DBSCAN to solve a multi-task learning problem [SvW19]. In this work, the problem is not formulated as a hedonic game but, in practice, it is similar. This solution, however, necessarily suffers from the above mentioned limitations of density-based clustering when applied to this domain. Another similar approach is Clustered Federated Learning [SMS20], developed in parallel to our approach, which cluster clients based on their data distribution in the context of personalized federated learning. This approach,

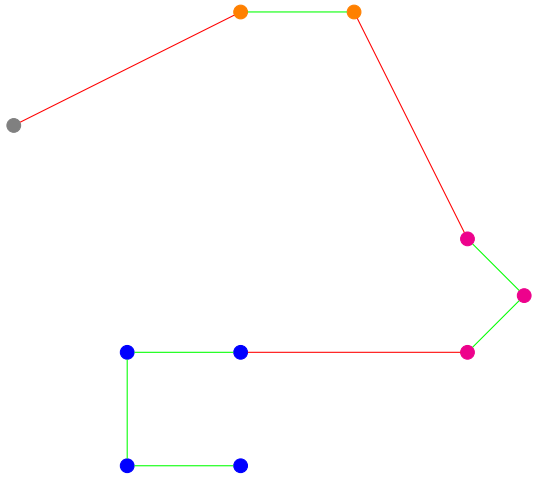


Figure 2.5: Illustration of OPTICS

however, does not take into account the size of clusters.

V.2.c Other visions of clustering

Outside the previous two main categories, we should also mention two other approaches of clustering. First, hierarchical clustering, in which the output is a hierarchy of clusters. Second, Expectation–maximization approaches, which tries to fit data with probability distributions.

Hierarchical clustering [Rok05] is an extension of clustering in which the objective is not only to identify clusters but also sub-clusters, potentially with numerous levels (of clusters). This kind of clustering can be achieved using algorithms from the two previous categories. In the case of k-means, a possibility is simply to recursively run k-means on each of the clusters extracted. In the case of OPTICS, the post-processing algorithm can be adapted to output a hierarchy of clusters. Optimizing OPTICS application to hierarchical clustering received specific attention from some authors [Ach+07]. A hierarchical variant of DBSCAN, HDBSCAN, also exists [Cam+15] (it is the algorithm used in [SvW19]).

Expectation–maximization's [DLR77] objective is to tune the parameters of some probability distribution to fit given data. If this method is used with a mixture distribution, clusters can be derived from its output, associating each point with the sub-distribution it is the most likely to be generated by. The optimization procedure, however, is much more complex than classical clustering algorithms, which makes it less practical. Also, this approach requires the user to choose the right kind distribution to fit the data.

V.3 Finding the right clustering algorithm

This review of literature led to the following conclusion: no existing clustering algorithm fitted our needs. k-means variants do not take group size into account and lack the ability to leave agents alone, which is something that should be possible with our formalism. OPTICS can leave agents alone but its design is not connected to the geometry of groups, which we want, and it also does not take size into account. We do not consider the size-constrained variant of k-means as a solution as size, in this algorithm, is an "external" constraint, it can not be balanced with geometrical considerations.

In Table 2.2 what can be achieved with state-of-the-art k-means and OPTICS, compared to what the approach we developed, AUCCCR (pronounced "okr", IPA: [okʁ]) can achieve.

Algorithm	Geometry	Leave alone	Cluster size
k-means	✔	✘	✘
OPTICS	✘	✔	✘
AUCCCR (ours)	✔	✔	✔

Table 2.2: Our clustering algorithm compared to state-of-the-art

VI Conclusion

Machine learning is a major and large field in modern computer science. Its history is rich of innovation, transdisciplinary work and creativity. While early works did not meet great success, causing artificial intelligence to remain regarded as science-fiction for decades, recent years took it to the path of glory. Machine learning is now an overactive field with numerous problems identified, formulated and solved. Distributed multi-task machine learning is one of those problems that receive significant attention since few years. A problem the present work contributes to solve.

While less prestigious than machine learning, clustering is also a rich domain of research. Subfield of the not less rich data analysis system, it has many applications and various formulations and associated algorithms. Adding distributed multi-task machine learning to the list of its applications and providing an associated algorithm is also a significant part of the present work.

Hedonic games are a rather theoretical subject, part of game theory and linked to economics. Making hedonic games a link between machine learning and clustering adds to the possible applications of this subject and contributes to enrich it.

Chapter 3

Simple, Efficient and Convenient Decentralized Multi-Task Learning for Neural Networks

I Introduction

A critical requirement for machine learning is training data. In some cases, a great amount of data is available from different (potential) users of the system, but simply aggregating this data and using it for training is not always practical. The data might for instance be large and extremely distributed and collecting it may incur significant communication cost. Users may also be unwilling to share their data due to its potential sensitive nature, as is the case with private conversations, browsing histories, or health-related data [Che+17].

To address these issues, several works have proposed to share model-related information (such as gradients or model coefficients) rather than raw data [Rec+11; Kon+16; McM+17]. These approaches are however typically mono-task, in the sense that all users are assumed to be solving the same ML task. Unfortunately, in a distributed setting, the problems that users want to solve may not be perfectly identical. Let us consider the example of speech recognition being performed on mobile device. At the user level, each has a different voice, and so the different devices do not perform exactly the same task. Similarly, at the level of a country or region, language variants also lead to variations between tasks. For example users from Quebec can clearly be separated from those from France. A decentralized or federated learning (a concept supported by industry leaders such as Google [Bon+19] and Facebook [Li+20]) platform should therefore accommodate for both types of differences between tasks: at the level of single users and at the level of groups of users.

Multitask learning has been widely studied in a centralized setting [Rud17]. Some distributed solutions for multitask learning exist, but they are typically limited to either linear [OHJ12] or convex [Smi+16; Bel+18; ZBT19] optimization problems. As a result, they are typically not applicable to neural networks, in spite of their high versatility and general success in solving a broad range of machine learning problems. Some recent preprints have proposed approaches for multitask learning with neural networks in federated setups, but they have more or less significant limitations (no parallelism, lack of flexibility, no decentralization) [CB19; Jia+19; Ari+19].

In this chapter, we introduce a novel effective solution for *decentralized multi-task learning with neural networks*, usable in a decentralized or federated setup. Its novelty is twofold. First, to the best of our knowledge, it is the first to combine *decentralization*, the ability to address *multiple tasks*, and support for *general neural networks*. Second, it provides a *flexible* approach that can operate without knowing the nature of the differences between peers' tasks, but can still benefit from such knowledge when available, as our experimental evaluation suggests.

Our approach relies on model averaging, already used in non-multi-tasks federated learning, but

its key idea consists in making the averaging partial. First, it enables partial averaging of peer models by making it possible to average specifics subsets of model parameters. Second, it makes it possible to average those subsets only among peers that share similar tasks if this knowledge is available. Experimental results show that partial averaging outperforms both local learning and global averaging.

In the following, we present our distributed multi-task learning system (model, principle, algorithm), perform a theoretical study of the associated learning process and experimentally evaluate the system on various tasks.

II The method

II.1 Base system model

We consider a federated/peer-to peer-setup where each local device (peer), p, has an individual dataset, D^p, and maintains a model consisting of a neural network, N^p. We use the notation $N^p.\text{train}(D^p)$ to mean that peer p trains its neural network N^p with some training algorithm on dataset D^p. Similarly, the notation $N^p.\text{test}(D^p)$ means that the network is tested using D^p, returning a success rate (accuracy) $\in \mathbb{R}^+$. Each peer seeks to maximize the success rate of its neural network when tested with its data provider.

Each neural network, N^p, is divided into *parts* (one or more) of neurons consuming the same inputs (with different weights), activation function (with different parameters) and with their outputs treated equivalently: $N^p = (N^p_0, ..., N^p_k)$[1]. The essential idea of a part is that neurons in a part can be treated equivalently and permuted without altering the computation. This corresponds, for example, to a fully connected layer, but not to a convolutional layer [LB95], which must be divided into several parts (since neurons in a convolution layer do not share the same inputs). Parts consist of several neurons and an associated activation function: $N^p_i = ((N^p_{i,0}, ..., N^p_{i,k}), f)$ (Theoretically this definition is actually equivalent to having specific activation functions for each neuron, since a real parameter can be used to index activation functions in a finite set of neurons)[2]. The activation function f is a real function with real parameters $\mathbb{R} \times \mathbb{R}^k \to \mathbb{R}$. Each neuron n(n is used instead of $N^p_{i,k}$ for readability) consists of a weight vector and a vector of activation-function parameters, both vectors are real-valued, $n = (\mathbf{w}, \mathbf{q}) \in \mathbb{R}^j \times \mathbb{R}^k$. $\mathbf{w}$ and $\mathbf{q}$ may be learned by any training method. Neurons take an input vector, $\mathbf{i} \in \mathbb{R}^k$, and return a real value, $f(\mathbf{i} \cdot \mathbf{w}, \mathbf{q})$.

The neurons of a part all have the same input (but some input coordinates may have a fixed 0 weight, as long as this is the case in all peers' neural networks), which can be either the output of another part or the input of the network itself. For modeling, we consider that the input arrives at designated input parts, consisting of neurons which just reproduce the input and are not updated in training. Bias neurons, outputting a constant value, are not considered, since their behavior is equivalent to introducing a bias as activation function parameter. Since they have fixed parameters, input parts are implicitly excluded from the collaborative learning process, since they simply do not learn.

A peer's neural network may include parts that do not exist for all peer's networks.

II.2 General idea

We reuse the mechanism of model averaging proposed for federated learning [McM+17], but adapt it to allow multi-task learning in a decentralized setup. More specifically, our intuition is that the model of similar but different tasks can be divided into different portions that represent different levels of

[1] i, j and k are used as generic index variables. Unless explicitly stated otherwise, i, j and k from different formulas have not the same meaning.

[2] $\#\mathbb{R} = \beth_1 > \beth_0 = \aleph_0 > k(\forall_{k\in\mathbb{N}})$, since the set of neurons is finite, it is not an issue to use a real index to define each neuron's activation function, even if $\#\mathbb{R}^{\mathbb{R}} = \beth_2 > \beth_1 = \#\mathbb{R}$ implies that a real value can not index the set of all real functions

"specialization" of each task. A simple application of this principle consists for instance in dividing the model of each task into a global section, shared by all tasks and used for collaborative learning, and a local section, specific to the task at hand. This is similar in a way to transfer learning [PMK91], albeit applied to a multi-task decentralized set-up.

A key property of our approach consists in not limiting this split to a pair of global/local sub-models. Since different peers involved in multi-task learning may have more or less similar functions to learn, we allow peers to have portions of their neural networks that are averaged with only a subset of other peers. The result is a partitioning of each peer's neural-network into several portions, each to be averaged with a specific subset of the peers. We refer to such portions as *slices* and to the model associated with a slice as a *partial model*. Slices are independent of parts. They can correspond to a part, a set of parts or span over several parts they cover only partially.

Summing up, our system uses averaging to enable decentralized learning; but we make this averaging partial along two dimensions. First, peers average only limited portions of their neural networks. This causes peers to retain *local* portions of their models thereby enabling multi-task learning. Second, averaging operations do not necessarily involve all peers, rather a peer can average each of its slices with a different subset of peers. This makes it possible for peers that share similar tasks to share some parts of their models while further specializing others.

Unlike modern non-distributed multi-task learning approaches commonly referred to as "transfer learning" [Rud17], we do not constrain the task-specific local portions of a model to consists of its last few layers. Rather, we enable complete freedom in the definition of which portions of the neural network are averaged. Once slices have been defined, the decentralized learning process automatically captures the differences between the tasks associated with different peers. In general, the partitioning of a peer's neural network into slices can be completely independent of the nature or the number of tasks in the decentralized or federated learning environment. However, knowledge of the differences between tasks can enable further optimization, for example by defining that peers with similar tasks should average some of their slices together.

With our system, the differences between peers' respective tasks will automatically be integrated by the decentralized learning process, in the learned parameters, without requiring a specific learning process nor manual setting (using a generic configuration for hyperparameters). It would still be possible, though, to use knowledge of the differences to optimize the system (custom hyperparameters), by grouping peers with more similar tasks or choosing specific portions of the neural network for averaging.

In the following, we provide a formal description of our method.

II.3 Detailed approach

In local learning, each peer learns a specific model; in non-multi-task distributed learning, all peers learn a single model that is shared with every other peer. Our distributed multi-task learning system introduces the notion of *partial models*. Each peer learns a set of partial models; each of which can be learned by one, some or all peers. Figure 3.1 gives an example of peers with partial models. We can we classify models into three categories. A *global model* consists of a partial model shared by all the peers. A *semi-local model* consists of a partial model shared by several but not all peers, while a *local model* consists of a partial model associated with only one peer. A peer has at least one model of any type, but none of the types is required.

To implement our partial models, we define *slices* as disjoint sets of neurons that cover a peer's entire neural network: $N^p = \biguplus_i S^{p,i}$, where $S^{p,i}$ is a slice ($\biguplus$: disjoint union). Each slice consists of a portion of a peer's neural network that is averaged on a specific subset of peers. Figure 3.2 gives an example of neural network divided into parts (dashed rectangles) and slices (nodes (neurons) of same color and shape). A peer implements a partial model by associating one of its slices with it. If the implemented partial model is global or semi local, this involves averaging the slice with the peers that also implement the same partial model.

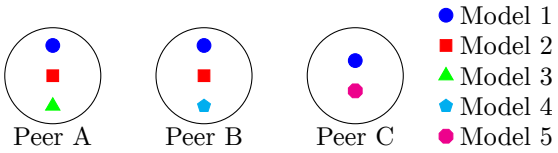


Figure 3.1: 3 peers with 5 models

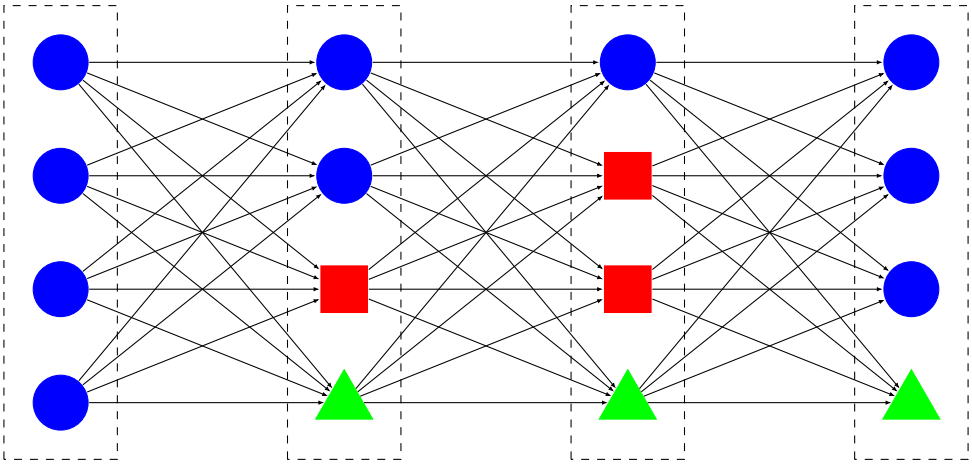


Figure 3.2: Parts (dashed rectangles) and slices (nodes (neurons) of same color and shape)

We also define a notion of *dependency* between partial models. If a peer implements a partial model (associating it with one of its slices) it must also implement all the models on which it depends. For example a partial model that recognizes Québécois French may depend on a model that recognizes generic French. Dependency defines a partial order (the relationship is antisymmetric and transitive) on the set of models; if A depends on B, we say that A is lower than B. This allows us to determine how to average inter-model weights. An example of models of different types with dependency relationships is given in Figure 3.3 (boxes ares partial models, arrows are dependency relationships).

II.3.a Partial averaging

Partial models allow us to implement partial averaging, as opposed to global averaging of the whole neural network, as done in non-multi-task collaborative learning [McM+17]. For each partial model, we only average the slices associated with it at the different peers that implement it.

When averaging the slices (one for each peer) associated with a partial model, we average all the activation-function parameter vectors, $\mathbf{q}$, of associated neurons, and a subset of the input weights (a subset of each of $\mathbf{w}$'s elements). Specifically, we average all the intra-model weights, i.e. those connecting two neurons in the same slice. For inter-model input weights, we rely on the notion of *dependency* between partial models. Specifically, we average only inter-model weights that connect a model to the models it depends on in either direction. If we have model A that depends on model B, we average inter-model weights of neurons of A from neurons of B and those of neurons of B from neurons of A. In other words, the input weights of neurons are not necessarily averaged with the model associated with the neurons they are part of, but rather with the model that is lower in the dependency hierarchy between the model of the neuron they are part of and the model of the neuron from which the input comes; if no such hierarchical relationship exists (the set of all models being only partially ordered by the dependency relationship), the weights are kept local.

The notion of dependency ensures that inter-model weights have the same meaning for all the peers implementing a model. For example, in Figure 3.4b, the red model can be Québécois, while the blue model may be generic French. We average inter-model weights among the peers that implement the Québécois model. If there is no dependency between two models (Figure 3.4c) inter-model weights

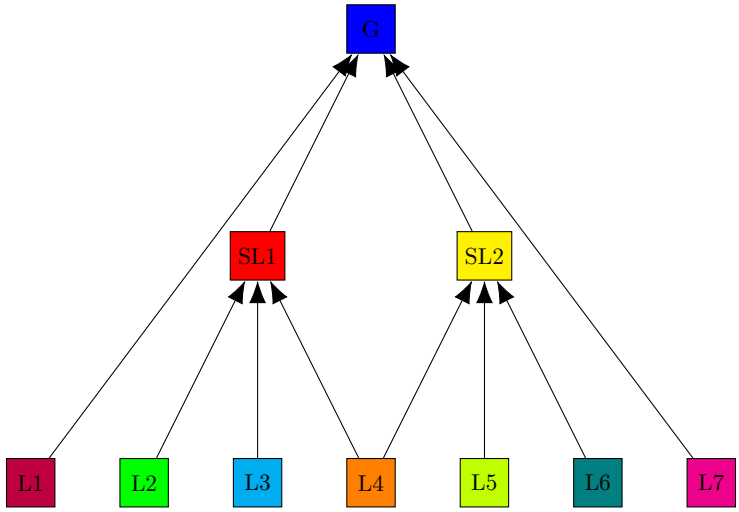
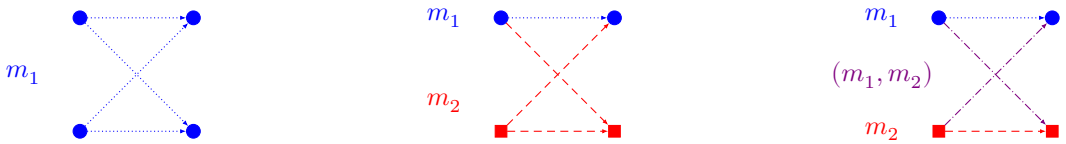


Figure 3.3: Models with dependency relationships (A →B ⇔ A depends on B)



(a) Only one model (m_1) (b) Red (m_2) depends on Blue (m_1) (c) No dependency

Figure 3.4: Averaging inter-model weights with different dependencies. A neuron's color and shape indicates the model associated with the neuron; an edge color and style indicates for which model this edge weight will be averaged. We represent two models: blue circles and dotted lines versus red squares and dashed lines. Purple edges refers to weights that are not averaged in either model.

remain local to each peer. It would in principle be possible to average such weights among all peers that implement both partial models at the same time, but this would add significant complexity to the process for limited gain. An example of 3 peers with 5 partial models and dependency relationships is given in Figure 3.5.

II.4 Mathematical formalization

Our system has been developed with an operational approach. For this reason, the description we provided, while detailing how our system effectively works in its application context, does not include a mathematical formalization. Averaging is nothing new or complex but our partial averaging and associated dependency system follows a logic which is interesting to formalize. Let us now present such a mathematical formalization. This generic mathematical formalization is abstracted from the neural network context, which would allow it to be extended to other machine learning systems.

As explained before (Section II.3.a), the dependencies are a design choice of engineers under the constraint that the dependency relationship must be an order relationship (antisymmetric and transitive). Since a model does not depend on itself, this order is strict and will be noted $m_2 \prec m_1$ (m_2 depends on m_1).

For this more generic formalism, we use the generic notion of *parameter*, which, in the case of neural network, is a generalization of weights and activation function parameters.

An important feature of neural network is their geometry, all parameters are not equivalent. This fact is largely used by our partial averaging system, so we need to preserve it for this more generic formalism. To this end, we introduce the notion of *cell*, in order to capture this geometrical organization of parameters. In the case of neural network, a cell represents a neuron, though it may be associated

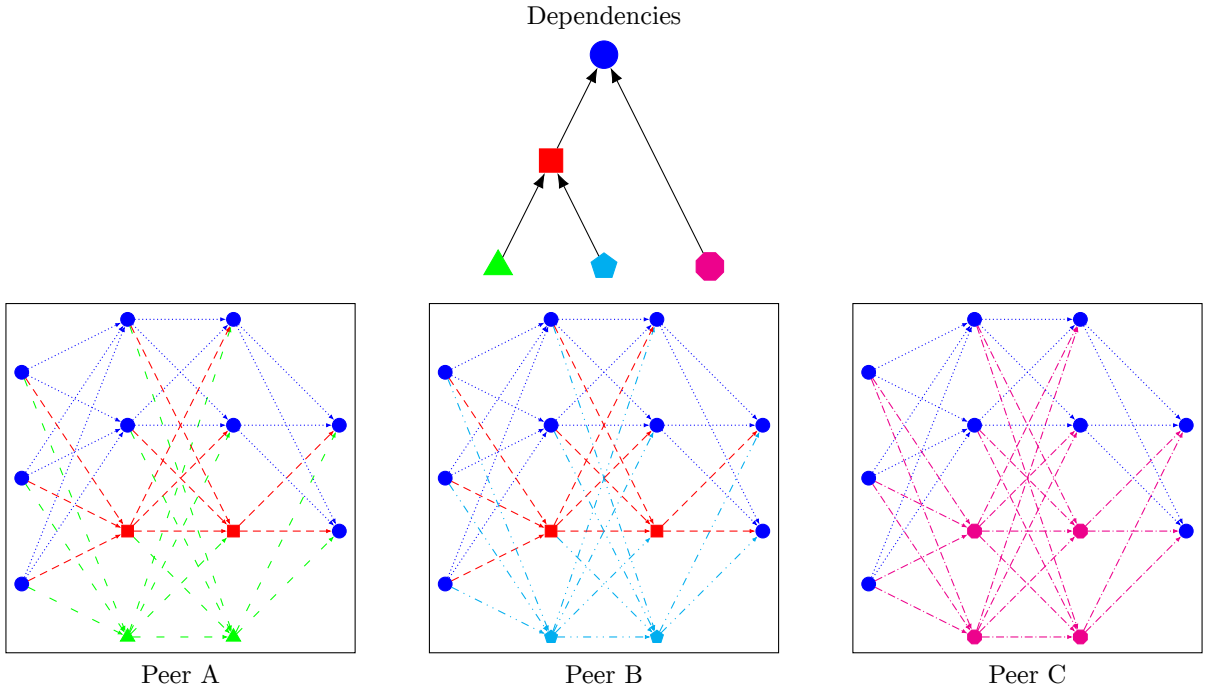


Figure 3.5: The neural networks three peers implementing five models (Blue global; Red semi-local; Green, Cyan and Purple local) and the corresponding dependency graph (A →B ⇔ A depends on B). We that links (weights) connecting two neurons from different models are considered par of the dependent model (if any) for averaging purposes.

with another notion in another class of machine learning systems. Cells are a general abstraction of what neurons are for neural networks: a way to organise parameters in each peer's neural network.

Obviously, the developers creating the models and implementing them locally have to ensure that all models implementation have the same internal topology on all peers, same for the connections between a model's implementation and its dependencies' implementations. Cells being local to each peer, the formalism itself does not guarantee that local topologies will be compatible. Models' topologies must be defined once before being implemented by peers. We denote the fact that some cell c is part of an implementation of model m by $c \vdash m$. Let us now explain how we formalize model topologies to allow the system to know exactly which elements should be averaged together.

With our formalism, machine learning models (e.g. neural networks) are seen as (undirected) multi-hypergraphs (hypergraphs with "loops" and duplicate hyperedges)[3], with cells as vertices (which we consider to be numbered and, thus, ordered) and parameters as hyperedges[4]. In this formalization, we will consider cells and vertices, as well as parameters and hyperedges, to be equivalent. With this visualization, each peer has a local multi-hypergraph and each model an associated multi-hypergraph. Each peer's implementation of a model corresponds to the sub-multi-hypergraph of the peer's multi-hypergraph restricted to the cells (vertices) associated with the said model. We assume that each model's multi-hypergraph's vertices (cells) are identified by an index and, when extracting a sub-multi-hypergraph, vertices (cells) are renumbered from 0 (or 1) preserving their order. We call this index $\chi_m(c)$ (for cell c and model m). We also assume that hyperedges (parameters) sharing the same set of associated vertices are indexed by a function $\varpi(e)$ (these indexes are not unique among all

[3]Formally, a(n undirected) multi-hypergraph is a pair $H = (V, E)$ where V is the set of vertices and E the multi-set of edges, which are non-empty subsets of V.

[4]In the case of neural networks, these are, more simply, multi-graphs, or even simple graphs (with "loops") if neurons' activations functions have no more than one parameter (and no "loops" if activation functions are not parameterized). Notice that, while neural networks induce directed graphs, direction is not needed here and, thus, not included in the formalism.

hyperedges, just among those sharing the same "ends"). The structure of each model is capture by a multi-hypergraph, and each peer that implements this model must have a sub-multi-hypergraph, the implementation of the model, that is identical (indexes taken into account) to the multi-hypergraph of the model.

In addition, we assume that there is only one proper (known to all peers) way to connect a given set of cells from different models[5]. This means that, when two peers have parameters connecting different models, these parameters can be considered to be equivalent (with indexes respected). We also assume that models themselves share a common indexing $\chi(m)$ (i.e., a model is number 1, another number 2, etc, and these indexes are common to all peers, no matter which models they implement).

Since we work in a distributed context, we call the parameter vector of each peer $\mathbf{z}^p$, and individual parameters (elements of this vector) z_i^p ($z_i^p \in \mathbb{R}$). Now, we call Π the set of all valid references ("pointer") to some z_i^p ($\sim$ a couple (p,i)). In the following, $\pi \in \Pi$ is some element of Π (π is a generic way to refer to a parameter as a variable, not as its value). Each π is associated with one or more cell(s) (in a neural network, one for activation function parameters, two for weights). We denote $C(\pi)$ the set of cells associated with a parameter. We also denote by $\mu(c)$ the model a cell c is associated with. Now, we can define $M(\pi) = \{\mu(c) | c \in C(\pi)\}$ the set of all models associated with a parameter π, which we could also write $M(\pi) = \mu[C(\pi)]$.

Noting $\mathcal{C}$ the set of all cells and $\mathcal{M}$ the set of all models, we have: $C : \Pi \to 2^{\mathcal{C}}$, $\mu : \mathcal{C} \to \mathcal{M}$ and $M : \Pi \to 2^{\mathcal{M}}$. Taking $Ц : 2^{\mathcal{C}} \to 2^{\mathcal{M}}$ as the set variant of μ: $Ц(X) = \{\mu(c) | c \in X\}$, we can write $M = Ц \circ C$ ($M = \mu \circ C$ would have been an abuse of notation).

Figure 3.6 summarizes the functions and sets we introduced.

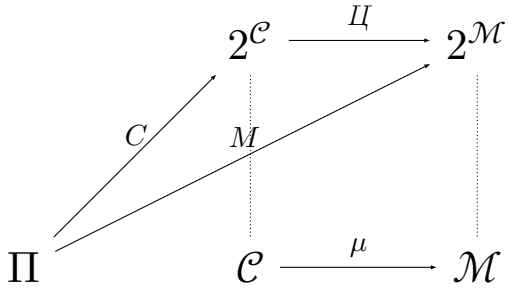


Figure 3.6: Sets and functions we defined. Π: set of all parameters. $\mathcal{C}$: set of all cells. $\mathcal{M}$: set of all models.

Now, we want to find a minimal set of models (ideally a singleton) to associate each π with for averaging. To this end, we define the reduced set of models associated with π as follows $R(\pi) = \{m | m \in M(\pi) \wedge \neg \exists m' \in M(\pi), m' \prec m\}$. In a less formal language, we obtain $R(\pi)$ from $M(\pi)$ by removing models which have other models depending on them in the set; $R(\pi)$ is the set of all minimal elements (relatively to $\prec$) of $M(\pi)$. For example, if $M(\pi) = \{m_1, m_2, m_3\}$ and $m_2 \prec m_1$ then $R(\pi) = \{m_2, m_3\}$. Let us prove the existence and unicity of $R(\pi)$.

Existence (constructive):

To construct $R(\pi)$ you can simply remove from $M(\pi)$ all elements which are greater than some other element of $M(\pi)$. $R(\pi)$ is not empty since, being a partial order on a finite set ($\mathcal{M}$), $\prec$ is well-founded, which means all subset of $\mathcal{M}$ (notably $M(\pi)$) contain at least one element m verifying $(\forall_{m' \in M(\pi)} \neg m' \prec m)$.

Unicity:

Suppose that we have two set different sets $R(\pi)$ and $R(\pi)'$ all $\in M(\pi)$. We can suppose without loss of generality that there is some m such that $m \in R(\pi) \wedge m \notin R'(\pi)$. Then there are two cases.

[5]We could impose the same restriction to all sets of cells or even to parameters associated with a single cell without much loss of generality but this is not necessary.

First case, $\exists m' \in M(\pi), m' \prec m$ which is in contradiction with the definition of $R(\pi)$. Second, $\forall m' \in M(\pi), \neg(m' \prec m)$, but in that case, m' should be $\in R'(\pi)$ by definition, so this is also a contradiction.

If $R(\pi)$ is a singleton, the parameter π will be averaged as part of the unique element (model) of $R(\pi)$. If $R(\pi)$ is not a singleton, the less complex solution is to keep π local, but theoretically π could be averaged among all peers implementing all elements (models) of $R(\pi)$. In that case, $R(\pi)$ as a set of models could be considered equivalent to a single model m such that $\forall m' \in R(\pi), m \prec m'$. This would however add significant practical complexity.

Note that mathematically we could have used something stronger than $R(\pi)$, a minimal lower bound of $M(\pi)$ (a minimal set L of elements such that $\forall m \in M(\pi), (m \in L \vee \exists m' \in L, m' \prec m)$) but such a set is not necessarily unique (if you have m_1, m_2, m_3, m_4, with $m_3 \prec m_1, m_3 \prec m_2, m_4 \prec m_1, m_4 \prec m_2$, then $\{m_3\}$ and $\{m_4\}$ both correspond to the definition for set $\{m_1, m_2\}$) and this has not much sens from engineering point of view due to the fact that such a set could include models not associated with any cell associated with π.

Figure 3.7 gives an example of what could be a neural network with our generic formalism. In this figure we have for example: $C(\pi_1) = \{c_1\}$, $C(\pi_{14}) = \{c_2, c_5\}$, $C(\pi_9) = \{c_1, c_3\}$, $\mu(c_1) = m_1$, $M(\pi_1) = \{m_1\}$, $M(\pi_{14}) = \{m_1\}$, $M(\pi_9) = \{m_2\}$.

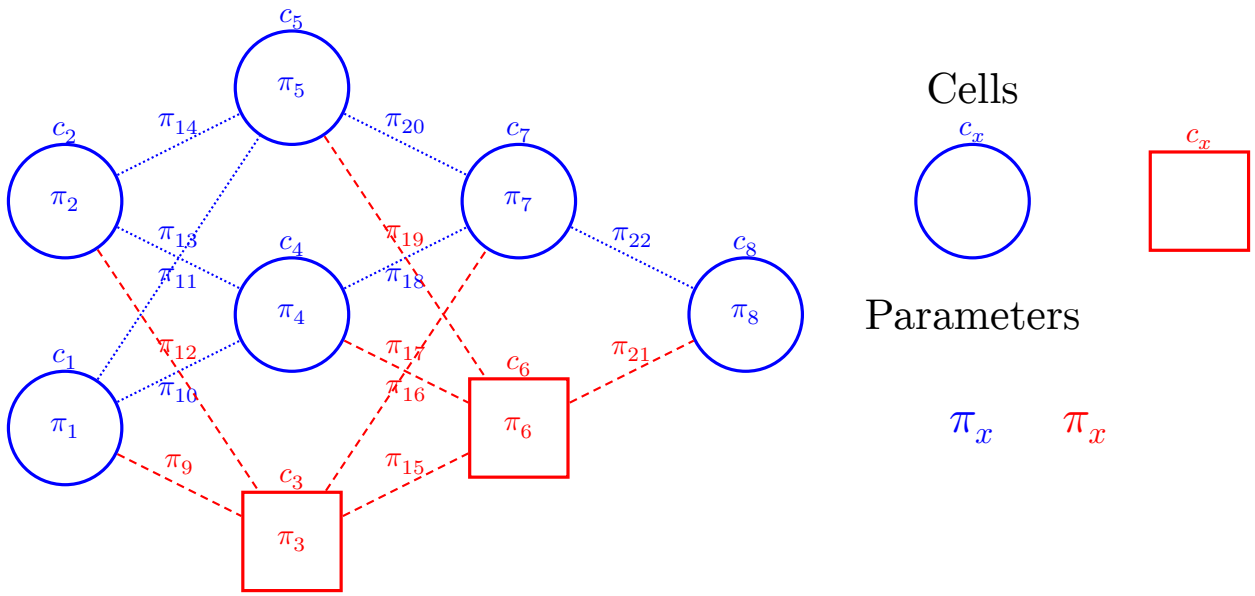


Figure 3.7: A neural network in our generic formalism. Two models with m_2 depending on m_1. Parameters can be both, activation function parameters (inside cells/neurons) or weights (on edges).

Once each parameter π reduced associated models set $R(\pi)$ defined, it can be averaged over all peers implementing all the models of $R(\pi)$. As explained before, averaging is only possible provided the different peers implement models respecting a common internal topology. To determine, among all parameters having the same $R(\pi)$, which should be averaged together, we rely on this common topology.

We defined common topology in terms of multi-hypergraphs, requiring compatible vertices (cells) and hyperedges (parameters) indexations. With these assumptions, we can assume that there is a generic way (function $\not{K}(\pi)$) to index all parameters associated with a certain set of models on all peers implementing it, so these weight can effectively be averaged (all parameters with the same index are averaged together). Such a common index can be derived from the sub-multi-hypergraphs associated with $\rho = R(\pi)$ extracted from each peer, which are identical due to assumptions (all

peers share identical implementations of models and connections between models are also identical, we just need to add the rule that cells in the sub-multi-hypergraphs must be indexed first based on their model's common index and then on their index in each model's multi-hypergraphs)[6]. It is then possible to average together all parameters π which have identical $R(\pi)$ and $\not{K}(\pi)$.

II.5 Algorithm

We detail our multi-task learning process in Algorithm 2. Algorithm 3 presents a modified main loop (and an additional function) to allow cross-model averaging. We use the following notation: $p \Vdash m$ means that peer p implements model m; $n \vdash m$ (resp. $i \vdash m$) means that the neuron n (resp. indexed by i) is part of (an implementation of) model m; $m.deps$ denotes the dependencies of model m. The notation $n \vdash m.deps$ (resp. $i \vdash m.deps$) is a short for $\bigvee_{m' \in m.deps} n \vdash m'$ (resp. $\bigvee_{m' \in m.deps} i \vdash m'$). For simplification, this formalization does not address index differences between peers for equivalent neurons and use I for the set of all neuron indices. This issue can be addressed in an implementation by adding index-offset values for each model and each peer.

Those algorithms use two primitives, *average* and *averageRestricted*. The first averages a set of vectors, the second averages predefined coordinates (second parameter) of the vectors from a set (first parameter). In a decentralized setup, these functions can be implemented with a gossip-based averaging protocol [JMB05], or with a private one [BFT19] for improved privacy protection. It is also possible to implement these primitives on a central server, to which peers send their models for averaging, in a federated, rather than decentralized, setup.

Essentially, all peers independently train their neural network, using some training algorithm with their local training data. Regularly, all peers synchronize and, for each model, relevant values are averaged for peers implementing this model. Relevant values being local parameters of neurons implementing the model, as well as their weights associated with neurons implementing the same model or one of its dependencies. In the cross-model averaging variant, an additional averaging is done for all pairs of models. In that case, we also average weights corresponding to links between neurons from the two models of the pair.

II.6 Discussion on implementation

II.6.a Decentralization

While our method has been designed to be applicable in a decentralized setup, it can also be used in a federated setup. Is this case, on just have to implement the averaging primitives on a central server, to which peers send their models for averaging, rather than using gossip averaging.

It is possible to have several servers as long as a model is associated with only one server. For cross-model averaging, it is required that each pair of models is associated to a unique server.

Since most computing work is done by peers and several servers can be used, this architecture belongs to Edge Computing [Shi+16].

II.6.b Privacy

While the decentralized multi-task learning method we propose does not include specific privacy protection mechanism, the way it works naturally offers some protection for users' privacy.

[6] With the previous assumptions, a possible way to get indexes (identical for all peers) for parameters is to take cells (vertices) common index (function $\xi_\rho(c)$, which is obtained by indexing all pairs $(\chi(m), \chi_m(c)) | m \in \rho \wedge c \vdash m$, with some integer pairs indexing function common to all peers) and hyperedges indexes (among other hyperedges share the same "ends") (function $\varpi(\pi) = \varpi(e)$ with e being the edge corresponding to π). Once this is done, you take for each parameter the set $\Xi(\pi) = \{n = 2^{\xi_\rho(c)} | c \in C(\pi)\} \cup \{3^{\varpi(\pi)}\}$. Then index all parameters based on $\Xi(\pi)$ (the set of all finite sets of integers can be indexed by integers; just take, for each integer n in the set, the nth prime number and multiply all those primes, you have a unique index due to the unicity of prime numbers decomposition) to obtain $\not{K}(\pi)$. Notice this process can be done locally by each peer only based on common information and its local parameters and will return the same result.

Algorithm 2: Decentralized multi-task learning

Data: num number of peers, mbn number of training rounds between averaging, M, the set of all models, T, the set of all neurons' index grouped by slice (index differences between peers are ignored for concision), N and D as defined before

```
loop
    each peer 0 ⩽ p < num does
        for 0 ⩽ i < mbn do
            N^p.TRAIN(D^p)
    foreach m ∈ M do
        AVERAGEMODEL({p ∈ ⟦0, num⟦ | p ⊩ m}, m)  // Does nothing if model m is local
function AVERAGEMODEL(P, m) is
    foreach i ∈ T do
        foreach k ∈ T_i | k ⊢ m do
            AVERAGE({N^p_{i,k}.q | p ∈ P})
            AVERAGERESTRICTED({N^p_{i,k}.w | p ∈ P}, {j | j ⊢ m ∨ j ⊢ m.deps})
        foreach k ∈ T_i | k ⊢ m.deps do
            AVERAGERESTRICTED({N^p_{i,k}.w | p ∈ P}, {j | j ⊢ m})
```

First, only models are transmitted, not the actual (training) data, thus, the only information transmitted is about the whole data-set and not individual data elements. While this does not completely prevent inference on training data from the trained model [NSH19], it is still a significant protection compared to transmitting the training data itself.

Also, only portions of each node's model are averaged. When the network converges to a stable state, all averaged slices will converge to the same value for all peers. All peer-specific information will end up in the non-averaged portion of the neural network, which is never transmitted, since this portion is the only thing that differentiate peers. During the first rounds, the information transmitted will be more peer specific, but since the models will not have converged to a stable state yet, this information will be very noisy.

In federated setups, since semi-locals models (see Section II.3) can be averaged on dedicated servers, this also reduces the amount of information that has to be transmitted to the central server (which averages the global model). Semi-local models may, for example, be private and limited to a specific organization's members, among all the organizations involved in the process. On a distributed setup, semi-local models are only transmitted to a limited set of peers.

Also, peers will never get to see any part of another peer's model with a centralized implementation.

When implementing the method using gossip averaging, a private gossip averaging protocol, like the one proposed in [BFT19], can also be used to further limit the ability of potential malicious peers to deduce information about other peers.

III Theoretical analysis

We now present a theoretical analysis of our distributed multi-task learning process. Due to the high level of generality of our system (not limited to one specific kind of neural network or learning rule), we provide an analysis of our system's behavior compared to local learning, which can be used to adapt existing convergence proofs to our model. More precisely, we show that in the case of loss-based learning (e.g. SGD), our decentralized system is essentially equivalent to a single model that comprises the whole set of partial models on a single system.

Algorithm 3: Decentralized multi-task learning, cross-model averaging extension

Data: num number of peers, mbn number of training rounds between averaging, M, the set of all models, T, the set of all neurons' index grouped by slice (index differences between peers are ignored for concision), N and D as defined before

```
loop
    each peer 0 ⩽ p < num does
        for 0 ⩽ i < mbn do
            N^p.TRAIN(D^p)
    foreach m ∈ M do
        AVERAGEMODEL({p ∈ ⟦0, num⟦ | p ⊩ m}, m)  // Does nothing if model m is local
    foreach (m1, m2) ∈ M × M | m1 ≠ m2 do                    // Cross-model part
        AVERAGECROSSMODELS({p ∈ ⟦0, num⟦ | p ⊩ m1 ∧ p ⊩ m2}, m1, m2)  /* Does nothing
         if no or only one peer implements both models */
function AVERAGECROSSMODELS(P, m1, m2) is
    foreach i ∈ T do
        foreach k ∈ T_i | k ⊢ m1 do
            AVERAGERESTRICTED({N^p_{i,k}.w | p ∈ P}, {j | j ⊢ m_2})
        foreach k ∈ T_i | k ⊢ m2 do
            AVERAGERESTRICTED({N^p_{i,k}.w | p ∈ P}, {j | j ⊢ m_1})
```

Each peer p has a parameter vector depending on time t (discrete, $\in \mathbb{N}$; t is incremented after each averaging step, which we assume to happen after each peers' models' updates; averaging steps synchronize all peers) $x_p(t) \in \mathbb{R}^{n_p}$[7] and a loss function $l_p \in \mathbb{R}^{n_p} \to \mathbb{R}$.

We start by defining a global loss function for our problem.

Since all peers' parameters are associated with a partial model (which can be local, for the parameters that are not averaged; note that coordinates not averaged must be considered part of the local model of a peer, also if the cross-model averaging extension is used, each pair of models have to be considered as one additional model for this analysis), we define a global parameter vector. To this end, we just take all partial models used (in the whole system), $m_0(t) \in \mathbb{R}^{r_0}, m_1(t) \in \mathbb{R}^{r_1}, ..., m_q(t) \in \mathbb{R}^{r_q}$ and concatenate them: $M(t) = m_0(t)m_1(t)...m_q(t) \in \mathbb{R}^u$ $(u = \sum_i r_i)$. If a peer p implements models 1, 3 and 5, then, after averaging, we have $x_p(t) = m_1(t)m_3(t)m_5(t)$.

To define our global loss function, $L \in \mathbb{R}^u \to \mathbb{R}$, we first define a set of functions $h_p \in \mathbb{R}^u \to \mathbb{R}$. $h_p(v)$ is simply equal to $l_p(v')$ where v' is the vector v restricted to the coordinates that corresponds to models implemented by peer p. For example, if $v = w_0w_1w_2w_3w_4$ and the peer p implements models 1, 2 and 4, then $h_p(v) = l_p(w_1w_2w_4)$. Since the values of parameters associated with models not implemented by peer p have no direct influence on the output of p's neural network, it is logical that they have no influence on p's loss.

Now we can simply define our global loss function as the sum of all h's: $L(v) = \sum_p h_p(v)$.

Let $v_{\llcorner}k_{\lrcorner}$ correspond to the coordinates of vector v associated with model k. We assume that each peer uses a learning rule satisfying the following property, which holds true for common variants of SGD:

$$\mathbb{E}[x_p(t+1) - x_p(t)] = -\lambda_p(t) \circ \nabla l_p(x_p(t))$$

Where $z \circ z'$ is the element-wise product of vectors z and z' and $\lambda_p(t)$ is the vector of the learning rates of peer p at time t for all parameters. Notice that we allow different parameters to have different

[7] we use x to contrast with the notation z used for parameter vectors in Section II.4 as this analysis focuses on the evolution of these vectors over time and uses different indexing for better readability in this context

learning rates (the learning rate is a vector, not a scalar); an idea first suggested in [Jac88] and on which some modern SGD-derived optimization algorithms, notably Adam [KB14], are based. Since we use model averaging, we need to distinguish the values of peers' models before and after averaging; we use $x_p(t)$ for the value before averaging and $x'_p(t)$ for the value after averaging ($x'_p(t)\llcorner k \lrcorner = m_k(t)$) and the version of the previous formula that we apply is:

$$\mathbb{E}[x_p(t+1) - x'_p(t)] = -\lambda_p(t) \circ \nabla l_p(x'_p(t))$$

Now, for each model m_k we have:

$$m_k(t) = \frac{\sum_{p|p\Vdash m_k} x_p(t)\llcorner k \lrcorner}{\#\{p \mid p \Vdash m_k\}}$$

$$m_k(t+1) - m_k(t) = \frac{\sum_{p|p\Vdash m_k} x_p(t+1)\llcorner k \lrcorner}{\#\{p \mid p \Vdash m_k\}} - m_k(t)$$

$$m_k(t+1) - m_k(t) = \frac{\sum_{p|p\Vdash m_k} x_p(t+1)\llcorner k \lrcorner}{\#\{p \mid p \Vdash m_k\}} - \frac{\sum_{p|p\Vdash m_k} x'_p(t)\llcorner k \lrcorner}{\#\{p \mid p \Vdash m_k\}}$$

$$m_k(t+1) - m_k(t) = \frac{\sum_{p|p\Vdash m_k} x_p(t+1)\llcorner k \lrcorner - \sum_{p|p\Vdash m_k} x'_p(t)\llcorner k \lrcorner}{\#\{p \mid p \Vdash m_k\}}$$

$$m_k(t+1) - m_k(t) = \frac{\sum_{p|p\Vdash m_k} x_p(t+1)\llcorner k \lrcorner - x'_p(t)\llcorner k \lrcorner}{\#\{p \mid p \Vdash m_k\}}$$

$$\mathbb{E}[m_k(t+1) - m_k(t)] = \frac{\sum_{p|p\Vdash m_k} \mathbb{E}[x_p(t+1)\llcorner k \lrcorner - x'_p(t)\llcorner k \lrcorner]}{\#\{p \mid p \Vdash m_k\}}$$

$$\mathbb{E}[m_k(t+1) - m_k(t)] = -\frac{\sum_{p|p\Vdash m_k} \lambda_p(t)\llcorner k \lrcorner \circ \nabla l_p(x'_p(t))\llcorner k \lrcorner}{\#\{p \mid p \Vdash m_k\}}$$

We now add the assertion that all peers implementing a model use the same learning rate for each parameter that is part of this model $\forall_{k,p,p'|p\Vdash m_k \wedge p'\Vdash m_k} \lambda_p(t)\llcorner k \lrcorner = \lambda_{p'}(t)\llcorner k \lrcorner = \Lambda_k(t)$. If the learning rate is supposed to be variable and not depending only on t, it is possible to compute a single learning rate for all peers implementing a model or to compute an optimal learning rate locally for each peer, average all value and use this averaged value. Now we can write $\lambda_k(t) = \frac{\Lambda_k(t)}{\#\{p|p\Vdash m_k\}}$ and $\Lambda(t) = \lambda_0(t)\lambda_1(t)...\lambda_q(t)$.

$$\mathbb{E}[m_k(t+1) - m_k(t)] = -\frac{\sum_{p|p\Vdash m_k} \Lambda_k(t) \circ \nabla l_p(x'_p(t))\llcorner k \lrcorner}{\#\{p \mid p \Vdash m_k\}}$$

$$\mathbb{E}[m_k(t+1) - m_k(t)] = -\frac{\Lambda_k(t)}{\#\{p \mid p \Vdash m_k\}} \circ \sum_{p|p\Vdash m_k} \nabla l_p(x'_p(t))\llcorner k \lrcorner$$

$$\mathbb{E}[m_k(t+1) - m_k(t)] = -\frac{\Lambda_k(t)}{\#\{p \mid p \Vdash m_k\}} \circ \nabla L(M(t))\llcorner k \lrcorner$$

$$\mathbb{E}[m_k(t+1) - m_k(t)] = -\lambda_k(t) \circ \nabla L(M(t))\llcorner k \lrcorner$$

Which, when we consider all models at the same time, gives us:

$$\mathbb{E}[M(t+1) - M(t)] = -\Lambda(t) \circ \nabla L(M(t)) \tag{3.1}$$

We are back to the same property for the global learning rule as for the local ones except that the loss function is the sum of the local losses and the learning rate of each parameter is divided by the number of peers having this parameter as part of their model.

This requires some discussion. Is having the learning rates divided by the number of peers implementing the corresponding model bad? Remember that the global loss function is a sum. If you consider the simple case where all peers have the same loss function, this mean that, compared to local learning, the (global) loss function will have its values, and thus its gradient, multiplied by the number of peers. This means that, in this case, if the learning rate was not divided by the number of peers, federated learning would be equivalent to learning with a higher learning rate, while it is reasonable to think that it should be equivalent to local learning with no modification of the learning rate, which is what we achieve with our reduced global learning rate. Put more succinctly, the reduced learning rate compensates the increased gradient.

We can summarize this analysis the following way: for a loss-based learning algorithm (typically SGD), our decentralized system is equivalent to training the whole set of partial models at once (on a single system) with a loss function that is the sum of peers local loss functions and a learning rate that is, for each partial model, divided by the number of peers implementing the model (which compensates the increased gradient induced by the sum).

In particular, we see with equation (3.1) that if the system is in a converged state, $\mathbb{E}[M(t+1) - M(t)] = \vec{0}$, then we have $-\mathcal{I}(t) \circ \nabla L(M(t)) = \vec{0}$ which means, assuming a > 0 learning rate for all parameters, that $\nabla L(M(t)) = \vec{0}$. This means that function $L(M(t))$ is at an extremum. In practice, except if the system started at a maximum, this extremum will be a (local) minimum (since the process is descending the gradient). If the function is convex, it will in all cases be the minimum of the function L, the sum of all peers loss functions, which is thus effectively the exact function optimized by our system.

IV Application to specific neural networks

Our method is general and, to be useful, needs to be implemented with some kind of neural network.

For our experiments, we focus on the Multi-Layer Perceptron (MLP), because it is a well-known, simple and efficient kind of neural network that is easy to train (with gradient descent). We also conduct additional experiments with a second kind of neural network, a custom neural network relying on associative learning, to prove that our method is not limited to MLP and SGD.

IV.1 Multi-Layer Perceptron

We worked with a basic MLP [GBC16], without convolutional layers [LB95]. The simple design of this kind of this well-known neural network being well suited for testing. For the same reason, we trained it using classical Stochastic Gradient Descent with Back Propagation [RHW86], without any additions (variable learning-rate, momentum). While such additions may give better performance, we wanted to stick to the most classical case possible to keep our results clean from potential side effects of more complex designs.

For this kind of neural networks the parts are, naturally, the layers. For each layer, we can associate a fixed number of neurons with each model we want to use.

Note that, in the case of Convolution Neural Networks (CNN) [LB95], unlike MLP, each convolution layer is not a single part, since neurons constituting it have different inputs. Those layers could easily be kept in a single slice but partial averaging is also possible, if the layers uses siblings cells: a few constitutional cells sharing the same inputs. In that case, a set of sibling cells is a part.

Figure 3.8 gives an example of an MLP with 4 models. Here, the red model depends on green and yellow, which themselves depends on blue. Grey lines are the ones that can not be associated with a unique model (they can be associated with the (unordered) pair $(green, yellow)$).

MLP will be our main test case for experiments. It will allow us to evaluate how our method works depending on various parameters.

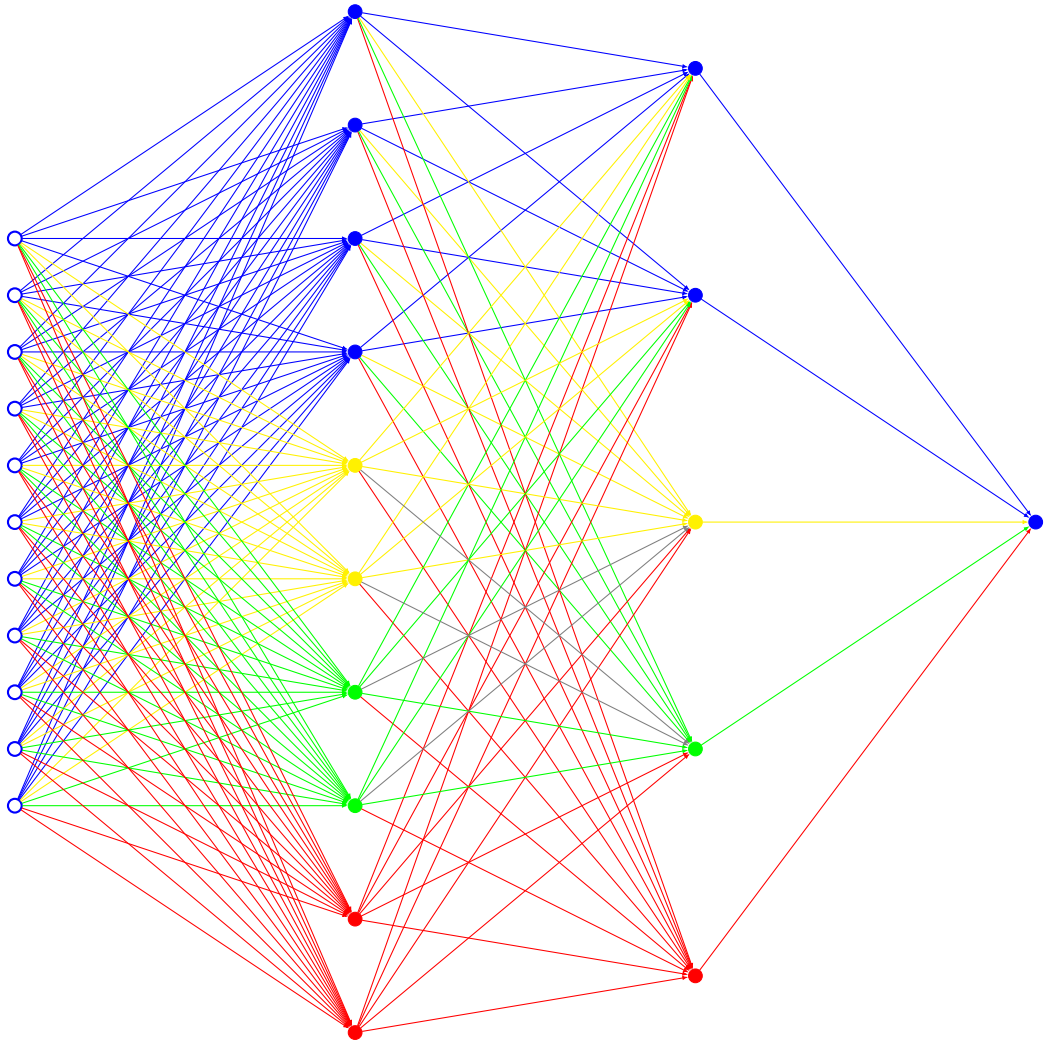


Figure 3.8: A multi-layer perceptron with 4 partial models (one per color)

IV.2 Associative neural network

To show how our method performs on a network that differs significantly from MLP and uses a learning algorithm different from gradient descent, we designed a custom kind of neural network using an associative learning rule (proof of concept).

A simple kind of task for an associative neural network is a completion task: given some partial input, return the complete item as output. The precise task we use is a sparse binary vector (i.e, binary vectors with few "1"s) completion task. The network must be able to complete (i.e., find missing "1"s) incomplete binary vectors after being trained on a set of (complete) vectors (the vectors the network have to be able to complete are exactly those of the training set). For example, some learned vector could be $[0, 0, 1, 0, 1, 1, 0, 0, 1]$ and the neural network should be able to return it if given the complete vector $[0, 0, 1, 0, 0, 1, 0, 0, 0]$ as input.

We took elements from Hopfield Networks [Hop82] for the neurons themselves and Restricted Boltzmann Machines [Smo86] for the structure of the network and the data is represented using Sparse Distributed Representation [AH15] (a concept of sparse binary vectors in the context of neural networks). Our network is divided into a visible and a hidden layer, forming a complete (directed) bipartite graph. The activation functions are fixed thresholds (output is either 1 or 0) and the network is used for completion tasks: a partial vector (with 0 instead of 1 for some of its coordinates) is given as input and the network must return the complete vector as output. A sample of this architecture is

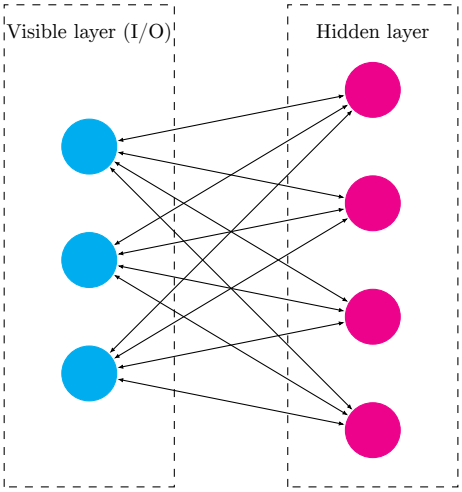


Figure 3.9: Our Associative Neural Network model

Algorithm 4: Computation

Data: *thld* threshold for neurons' activation

for $0 \leqslant i < hidden.size$ **do**
  if $visible.output \cdot hidden.weight[i] \geqslant thld$ **then**
    $hidden.output[i] \leftarrow 1$
  else
    $hidden.output[i] \leftarrow 0$
for $0 \leqslant i < visible.size$ **do**
  if $hidden.output \cdot visible.weight[i] \geqslant thld$ **then**
    $visible.output[i] \leftarrow 1$
  else
    $visible.output[i] \leftarrow 0$

presented in Figure 3.9.

The process is the following: the visible layer's neurons outputs are set to the values of the input vector, the hidden layers' neuron's output is computed from the values of the visible layer, finally, the visible neurons' values are updated, based on the hidden layer's values. The computation is simple: if the weighted sum of the inputs is $\geqslant$ some threshold (hyperparameter), the output is 1, 0 otherwise. See Algorithm 4 for pseudo-code.

For learning, the network is given a complete vector as input, then the hidden layer's values are computed, finally, both layers' neurons' weights are updated. The learning rule is the following: if two neurons are active together, the weight of the link between them is increased, if one is active and not the other, this weight is decreased. Increment values depend on a fixed learning rate (half of the distance from the extreme value times the learning rate) and weights are in the interval $[-1; 1]$. See Algorithm 5 for pseudo-code.

V Experiments

Our experiments aim to show the efficiency of our method on simple examples and evaluate how this efficiency is affected by various parameters. While real world applications of our method are likely to be more complex, these simple examples are an efficient way to provide detailed results about our

Algorithm 5: Learning

Data: *thld* threshold for neurons' activation, l the learning rate

for $0 \leqslant i < hidden.size$ **do**
  for $0 \leqslant j < visible.size$ **do**
    if $visible.output[j] == 1$ **then**
      if $hidden.output[i] == 1$ **then**
        $hidden.weight[i][j] += \frac{1-hidden.weight[i][j]}{2} l$
      else
        $hidden.weight[i][j] -= \frac{1+hidden.weight[i][j]}{2} l$

for $0 \leqslant i < visible.size$ **do**
  for $0 \leqslant j < hidden.size$ **do**
    if $hidden.output[j] == 1$ **then**
      if $visible.output[i] == 1$ **then**
        $visible.weight[i][j] += \frac{1-visible.weight[i][j]}{2} l$
      else
        $visible.weight[i][j] -= \frac{1+visible.weight[i][j]}{2} l$

method's behavior.[8]

As it is classically done in machine learning, our learning process relies on the notions of *batch* and *mini-batch*. A batch is a set of training data and a mini-batch is a subset of a batch (chosen randomly). During learning, neural networks process a determined number of mini-batches.

Our results from different tasks are summarized in Figure 3.10. We see that our approach systematically outperforms local training and global training (non-multi-task federated averaging). More details in the followings subsections.

V.1 Multi-Layer Perceptron - General considerations

This section summarizes the general considerations applicable to the two test series we performed on the Multi-Layer Perceptron.

In both cases, we used a classical Stochastic-Gradient-Descent algorithm with Back Propagation [RHW86] with a fixed learning rate of 0.1 and a sigmoid activation function $f(x) = \frac{1}{1+e^{-x}}$.

We use the following notation to describe our MLP layouts: n=m=p=q means that we have a four-layer MLP (input, output, and two hidden layers) with n neurons in the first (input) layer, m in the second, and so on. We use a similar notation to describe partial models: n-m-p-q means that the described model has n neurons for the first layer, m for the second, and so on. Since classical MLPs use full connected layers, all neurons of a layer are equivalent, thus models can be distinguished only by the number of neurons they include in each layer.

Unless explicitly stated otherwise, we use a 784=300=100=k configuration, were k is the number of classes for the relevant task, for our MLP (two hidden layers), unless explicitly stated otherwise, a fixed learning rate of 0.1 and a sigmoid activation function $f(x) = \frac{1}{1+e^{-x}}$.

The "Accuracy" metrics, since the tasks we used with MLP are symbol recognition tasks, is simply the proportion of recognized symbols. A symbol is considered recognized if the most active neuron of the last layer corresponds to this symbol.

The "averaging level" is the number of neurons in the global model (shared by all peers), others neurons being in local models, in the second hidden layers, which contains a total of 100 neurons. The

[8] Our code is available at https://gitlab.inria.fr/abouchra/distributed_neural_networks

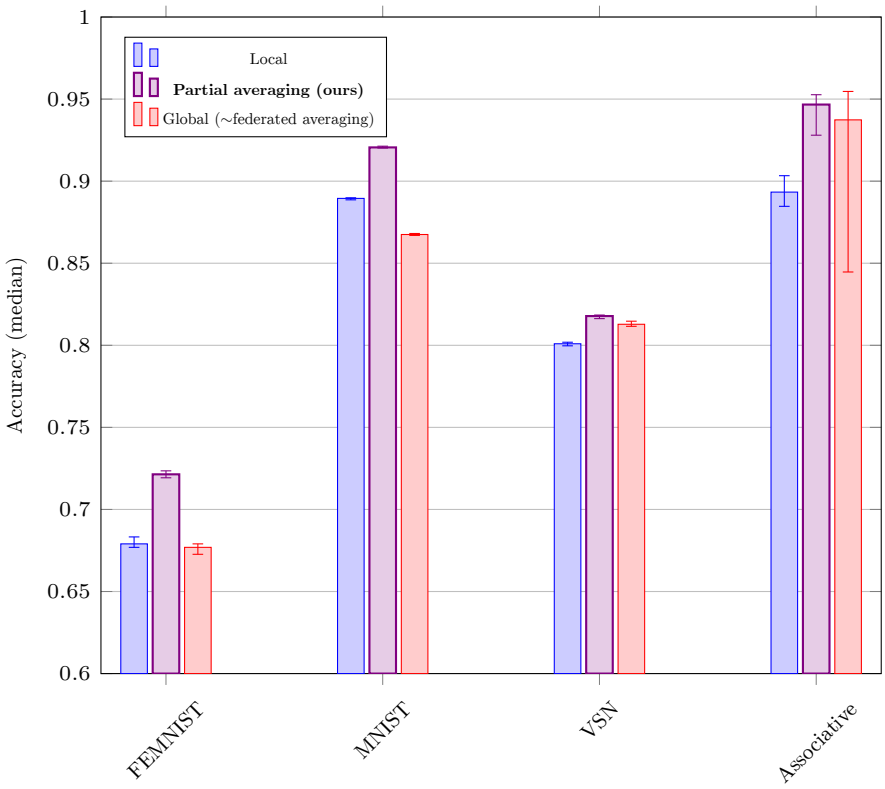


Figure 3.10: Results summary
Partial averaging is 80% for all but Associative, 70%

levels we commonly test are 0, 15, 30, 60, 80 and 100. Some neurons from the first hidden layer are also averaged, the exact numbers are, respectively, 0, 50, 100, 200, 250, 300 (proportional to the numbers in the first layer). The input and output layers are completely shared, except for an averaging level of 0. An averaging level of 0 is equivalent to local learning, 100 is equivalent to training a single global model (with averaging), similar to federated averaging.

In general, we present results as median values of a series of 10 independent runs with second and third quintiles in error bars. Due to the variety of setups tested, there are not many common parameters statically shared in different tests; we also have a wide variety of tests. For this reason, most test parameters are presented in each test's specific section.

V.2 Multi-Layer Perceptron - FEMNIST

The FEMNIST dataset [Cal+19] targets the recognition of 62 classes of handwritten characters (uppercase and lowercase letters, plus digits). FEMNIST is derived from the NIST Special Database 19 [Nat16], as is the well-known MNIST [LeC+98] dataset. Compared to MNIST, FEMNIST adds non-digit characters and groups characters by hand-writer. This grouping, originally intended for non-multi-task federated learning, can also be used for multi-task federated/decentralized learning since different writers do not have the same handwriting, creating the similar but different tasks we need to evaluate our system.

To adapt FEMNIST to the multi-task setup, we preprocess the dataset as follows. First, after partitioning the data by writer, we sort the writers by descending numbers of samples. This allows us to select the writers with the largest samples in each experiment. We also had to perform a random shuffle on the samples for each writer, since the natural order of the samples has obvious bias (big clusters of digits notably).

Our experiments simulate a peer-to-peer/federated setup. We assign exactly one writer to each

peer and randomly shuffle its samples. Then we limit all peers' (writers') sets of samples to the same size, truncating the largest sets. This prevents a "super peer" bias, where one peer with a very large dataset could, alone, allow the learning process to gain greater general accuracy than all other peers. This bias would artificially increase the average accuracy, masking the interest of collaboration between multiple smaller peers.

While this anti-bias rule is a great constraint that poses limitations on both, the number of peers we can include in our tests and the number of samples we can use, effectively reducing the maximum possible accuracy, we believe that it is very important to try to eliminate, has much as possible, such bias in the dataset to really prove the efficiency of distributed multi-task learning itself, without any side effects. While some could argue that dataset size disparity is present in real-world datasets, we can answer that, if the only interest of federated learning is transferring information from "rich" peers to "poor" peers, then, nothing would force those rich peers to share with poor, preferring to keep their large dataset for themselves or sell them or sell the models learned from them, making federated/distributed learning effectively useless. Thus, we will perform our analysis on peers with same size dataset, to prove the efficiency our system, even when it can not benefit from this wealth redistribution effect.

Each peers data is divided in a training and a testing section.

Each test is done on three variants of the FEMNIST task. FEMNIST-A, the classical FEMNIST task, with all characters (62 classes). FEMNIST-M, were some similar-looking characters are merged in a single class as proposed in the NIST Special Database 19 documentation [Nat16] (47 classes). FEMNIST-D, limited to digits (10 classes).

We performe a total of 3 independent tests with MLP on FEMNIST, each designed to evaluate one specific aspect of our system: partial averaging efficiency (Section V.2.a), different averaging schemes (Section V.2.b), effect of the number of peers (Section V.2.c). Results are median values of 10 runs.

V.2.a Averaging level

This first experiment aims to show that performing a partial averaging is better, in the considered multi-task situation, than a complete averaging or no averaging and to determine which level of averaging is the best.

For this experiment we use 16 peers, each with a training set of 360 characters for the A and M variants, 100 for the D variant. The test set is 60 characters long for the A and M variants and 30 characters long for the D variant. The mini-batch size is 180 for the A and M variants and 50 for the D variant.

For recall, the "averaging level" is the number of neurons in the global model (shared by all peers), others neurons being in local models, in the second hidden layers, which contains a total of 100 neurons. Some neurons from the first hidden layer are also averaged, in equivalent proportion.

Each curve corresponds to a different number of mini-batch (from 30 to 1000). The averaging process is done after every mini-batch. The results are presented in Figures 3.11, 3.12 and 3.13 (in order, A, M and D variants).

From those results, we can conclude that, whatever the mini-batch number is, the best performance is always achieved using partial averaging. In general, the best performance is achieved with an averaging level of 80. This validates our partial averaging concept.

V.2.b Averaging scheme

In this experiment, we compare our approach of partial averaging of each hidden layer (in its best performing variant, 784-250-80-62, with other partial averaging schemes based on complete averaging of specific layers. Notably, this includes a configuration similar to what is commonly used in local multi-task learning approaches design as "transfer learning": sharing everything but the last layer (784-300-100-0).

For this experiment we use 16 peers, each with a training set of 360 characters for the A and M variants, 100 for the D variant. The test set is 60 characters long for the A and M variants and 30

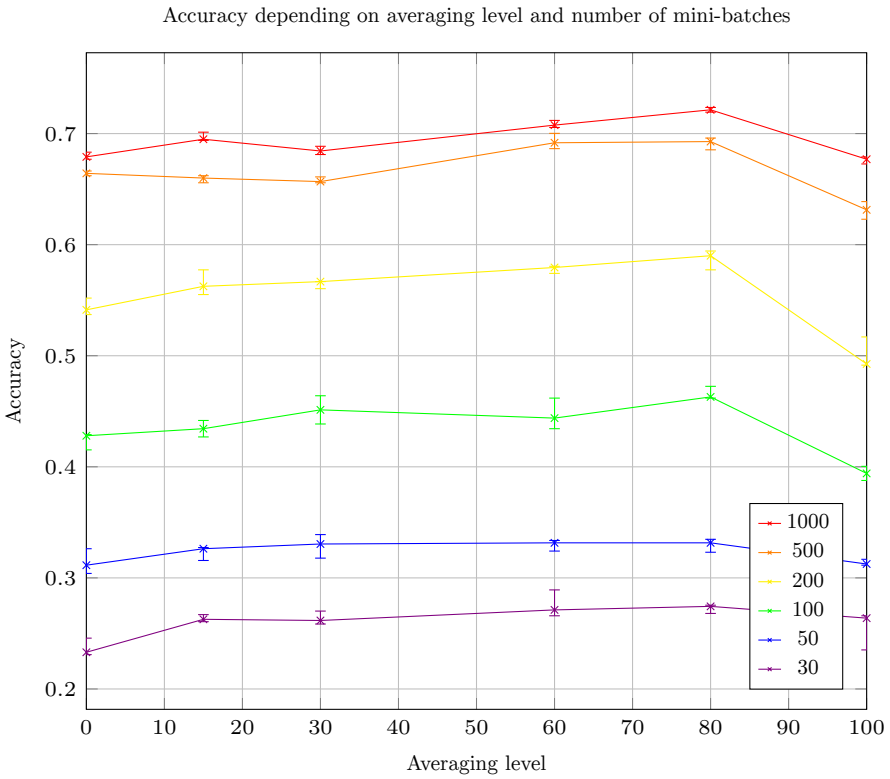


Figure 3.11: Averaging level on FEMNIST-A - Accuracy for various averaging levels and numbers of mini-batches

characters long for the D variant. The mini-batch size is 180 for the A and M variants and 50 for the D variant.

Each curve corresponds to a different averaging scheme. 200 mini-batches were used for training. The averaging process is done after every mini-batch. The results are presented in Figures 3.14, 3.15 and 3.16 (in order, A, M and D variants).

This experiment shows that, in the FEMNIST case, whatever the mini-batch number is, partially averaging hidden layers is superior to all tested schemes based on averaging specific layers, including 784-300-100-0. This proves the interest of having a flexible system allow partial averaging of layers rather than specific layers averaging.

V.2.c Number of peers

In this experiment, we evaluate how our system's performance is affected by the number of peers.

Due to the nature of the FEMNIST dataset, with hand-writers that can in no way be considered "uniform" either themselves or in their respective differences, this test will present the following particularities. First, for all "numbers of peers" tested, we actually tested 64 peers and the "number of peers" value is the number of peers sharing the same "global model". For example, the value for 16 peers is obtained by taking 4 groups of 16 peers (64 total) and performing the averaging process separately for each group. This eliminates any bias that could be exhibited if, for example, the first peers in our dataset were actually "easier". Secondly, we need to admit that another bias can still exist in our results. Even with our grouping method, some groups can be harder than others, notably, it is likely that multi-task learning is easier if you only have 4 different tasks rather than 16. Due to the nature of the FEMNIST dataset, each peer has its own, specific, task, thus, more peers mean more tasks. As a consequence, this tests is not only testing the effect of the number of peers but also the number of different tasks. While the number of peers should increase accuracy, the number of tasks

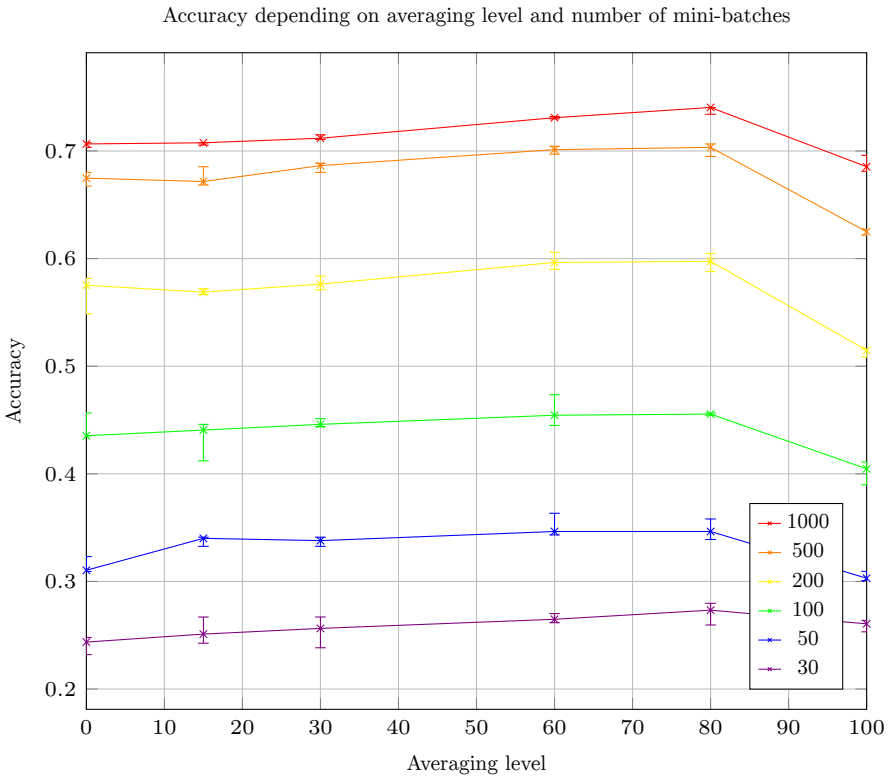


Figure 3.12: Averaging level on FEMNIST-M - Accuracy for various averaging levels and numbers of mini-batches

should decrease it. We have to keep this in mind when interpreting our results. Also, in addition to the number of peers, the different sets of peers may have unknown and unpredictable biases, so, we have to be careful with our conclusions.

For this experiment, each peer has a training set of 350 characters for the A and M variants, 100 for the D variant. The test set is 60 characters long for the A and M variants and 30 characters long for the D variant. The mini-batch size is 175 for the A and M variants and 50 for the D variant.

Each curve corresponds to a different averaging level. 200 mini-batches were used for training. The averaging process is done after every mini-batch. The results are presented in Figures 3.17, 3.18 and 3.19 (in order, A, M and D variants).

Obviously, the line for a 0 averaging level is flat, since the number of peers has no influence in this case. It clearly appears that full averaging (100) rapidly loses efficiency when the number of peers increases, specifically at the beginning. This is due to the increased number of different tasks; an averaging level of 100 being unable to perform multi-task learning. For an averaging level of 80, we observe, however, that our multi-task learning system seems to allow effective gains.

V.3 Multi-Layer Perceptron - modified MNIST

We continue our experiments with the MLP on the well-known MNIST dataset [LeC+98]. The task is simple: recognizing handwritten digits. To generate different but related learning tasks for each peer, we permute digits (e.g., a [9] will be labeled as “8” and vice versa). Compared to FEMNIST, this more artificial testing allows us to perform a more precise evaluation, by providing better control on the respective tasks of peers.

To obtain different training sets for different peers, we divide the MNIST training set in successive sequences. This is important because practical implementations will be likely to have peers with completely independent data; as keeping data local being one of the key features of our method. The

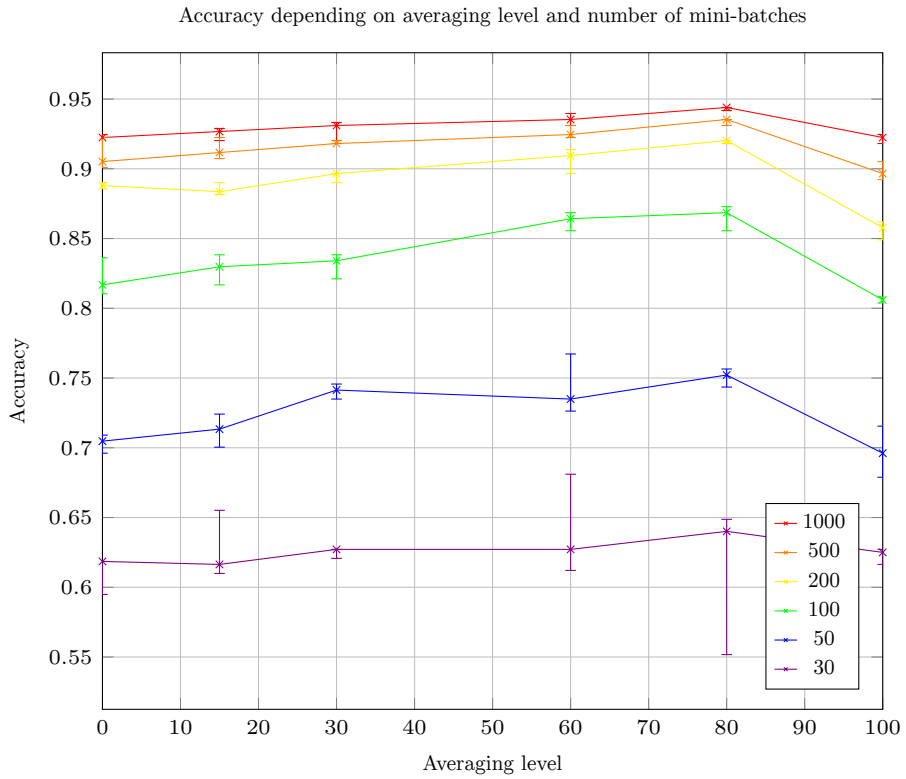


Figure 3.13: Averaging level on FEMNIST-D - Accuracy for various averaging levels and numbers of mini-batches

test set is the 1000 first digits of the MNIST test set.

All the training sets are different, not overlapping, parts of the whole MNIST training set (60000 digits). These choices are a compromise to have a sufficient number of peers while not reducing the training set size too much and maintaining independent training sets, within the limitation imposed by the total size of the MNIST training set.

We perform a total of 9 independent tests with MLP on MNIST, each designed to evaluate one specific aspect of our system: partial averaging efficiency (Section V.3.a), convergence (Section V.3.b), effect of the number of peers (Section V.3.c), semi-local models (Section V.3.d), semi-local+local models (Section V.3.e), effect of the level of difference between tasks (Section V.3.f), effect on the number of peers with modified tasks (Section V.3.g), effect of the training set size (Section V.3.h), use of a more problem-specific layout (Section V.3.i).

Similarly to previous experiments, results are median values of 10 runs, except for semi-local averaging, which has 20 runs (since results were closer, we wanted more samples). The error bars corresponds to the second and third quintiles.

V.3.a Averaging level

This first experiment aims to show that performing a partial averaging is better, in the considered multi-task situation, than a complete averaging or no averaging and to determine which level of averaging is the best.

For this experiment we use 16 peers, each with a training set of 3500 digits. All the training sets are different, not overlapping, parts of the whole MNIST training set (60000 digits). These choices are a compromise to have a sufficient number of peers while not reducing the training set size too much and maintaining independent training sets, with MNIST training set size limitation.

For recall, the “averaging level” is the number of neurons in the global model (shared by all peers),

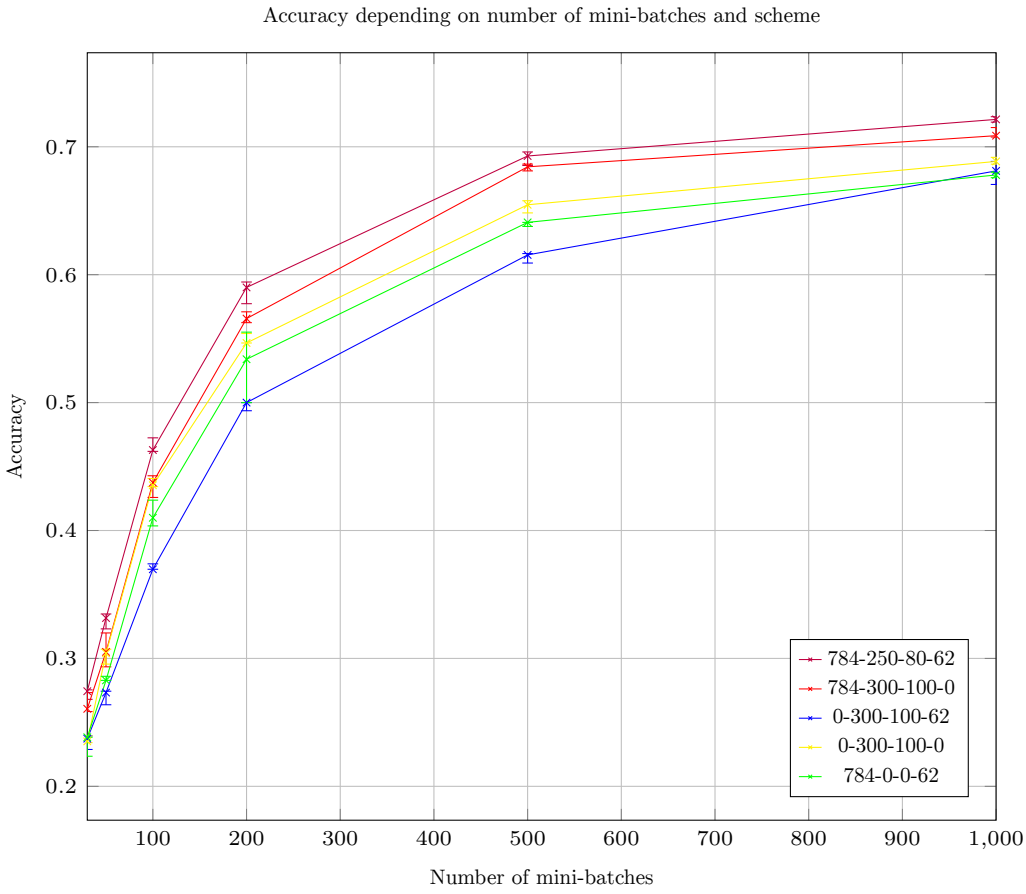


Figure 3.14: Averaging scheme on FEMNIST-A - Accuracy for various numbers of mini-batches and schemes

others neurons being in local models, in the second hidden layers, which contains a total of 100 neurons. Some neurons from the first hidden layer are also averaged, in equivalent proportion.

The averaging process is done after every mini-batch. 30 mini-batches are used for training. Differences between peers are induced by permuting digits 8 and 9 for 7 of the 16 peers, giving us a close to but not even split. The results are presented in Figure 3.20.

From those results, we can conclude that, whatever the mini-batch size is, the best performance is always achieved using partial averaging. For higher mini-batch sizes, the best performance is achieved with an averaging level of 80.

The significant drop of performance observed with an averaging level of 100 is due to fact that a unique model can not address the permutation of 8 and 9 for certain peers. This validates our partial averaging concept.

V.3.b Convergence

The point of this test is to see how the accuracy evolves during the training process.

For this experiment we tested different averaging levels, with 16 peers, a fixed mini-batch size of 100. Training sets are independent and of size 3500. The smaller mini-batch size allows finer grain in accuracy sampling (since the curve is drawn from accuracy evaluations on the test set taken after every mini-batch).

The “averaging level” is the number of neurons of the global model (shared by all peers ; others neurons being in local models) in the second hidden layer, which contains a total of 100 neurons. The levels we tested are 0, 80 and 100. Some neurons from the first hidden layer are also averaged, the

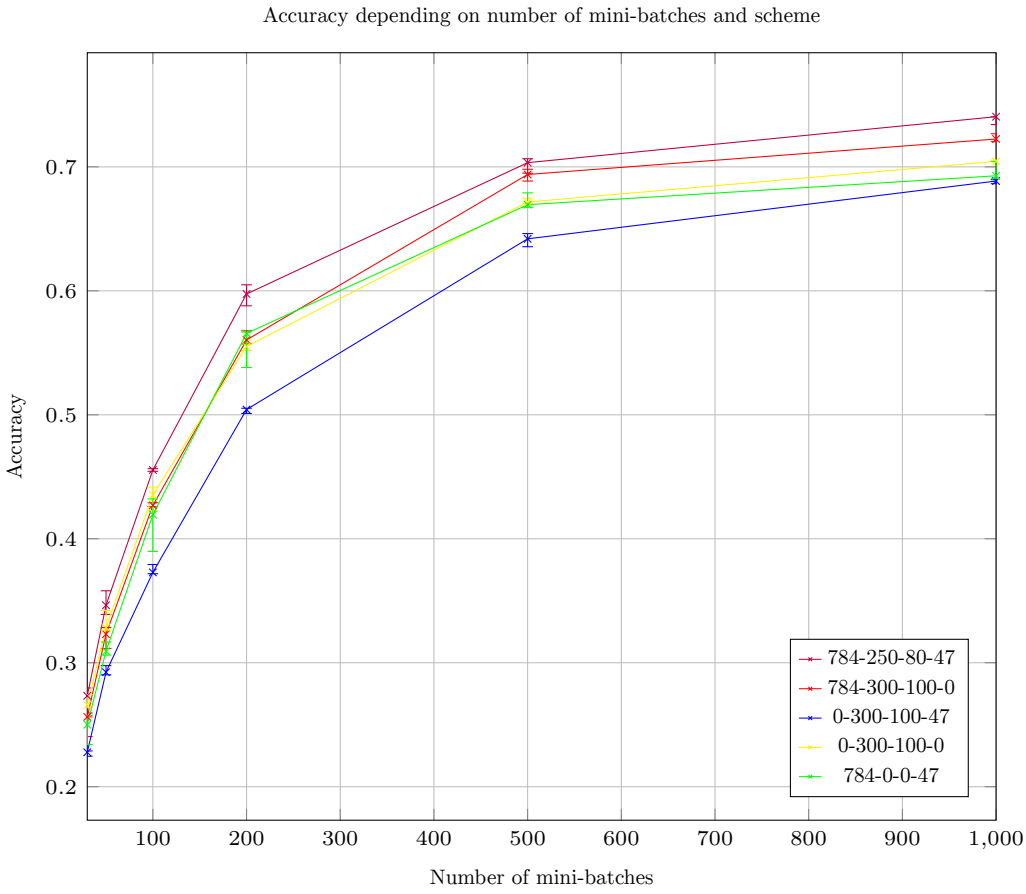


Figure 3.15: Averaging scheme on FEMNIST-M - Accuracy for various numbers of mini-batches and schemes

exact numbers are, respectively, 0, 250, 300. Each curve corresponds to a different averaging level. The averaging process was done every 10 mini-batch. 30 mini-batches were used for training. Differences between peers are induced by permuting digits 8 and 9 for 7 of the 16 peers.

Results are median values of 10 runs. The results are presented in Figure 3.21.

At the beginning of the training, no averaging is leading and 80 averaging is last but when approaching maximum accuracy, 80 becomes first and 100 last. The three curves have similar shapes and remain very close.

V.3.c Number of peers

In this experiment, we want to evaluate how different averaging levels' performances are affected by the number of peers.

For this experiment we tested different numbers of peers, from 4 to 16, and different averaging levels with a fixed mini-batch size of 500. Training sets are independent and of size 3500.

The "averaging level" is the number of neurons in the global model (shared by all peers), others neurons being in local models, in the second hidden layers, which contains a total of 100 neurons. The levels we tested are 0, 15, 30, 60, 80 and 100. Some neurons from the first hidden layer are also averaged, the exact numbers are, respectively, 0, 50, 100, 200, 250, 300. Each curve corresponds to a different averaging level. The averaging process was done after every mini-batch. 30 mini-batches were used for training. Differences between peers are induced by permuting digits 8 and 9 for one fourth of peers (this ratio dividing evenly all tested numbers of peers). The results are presented in Figure 3.22.

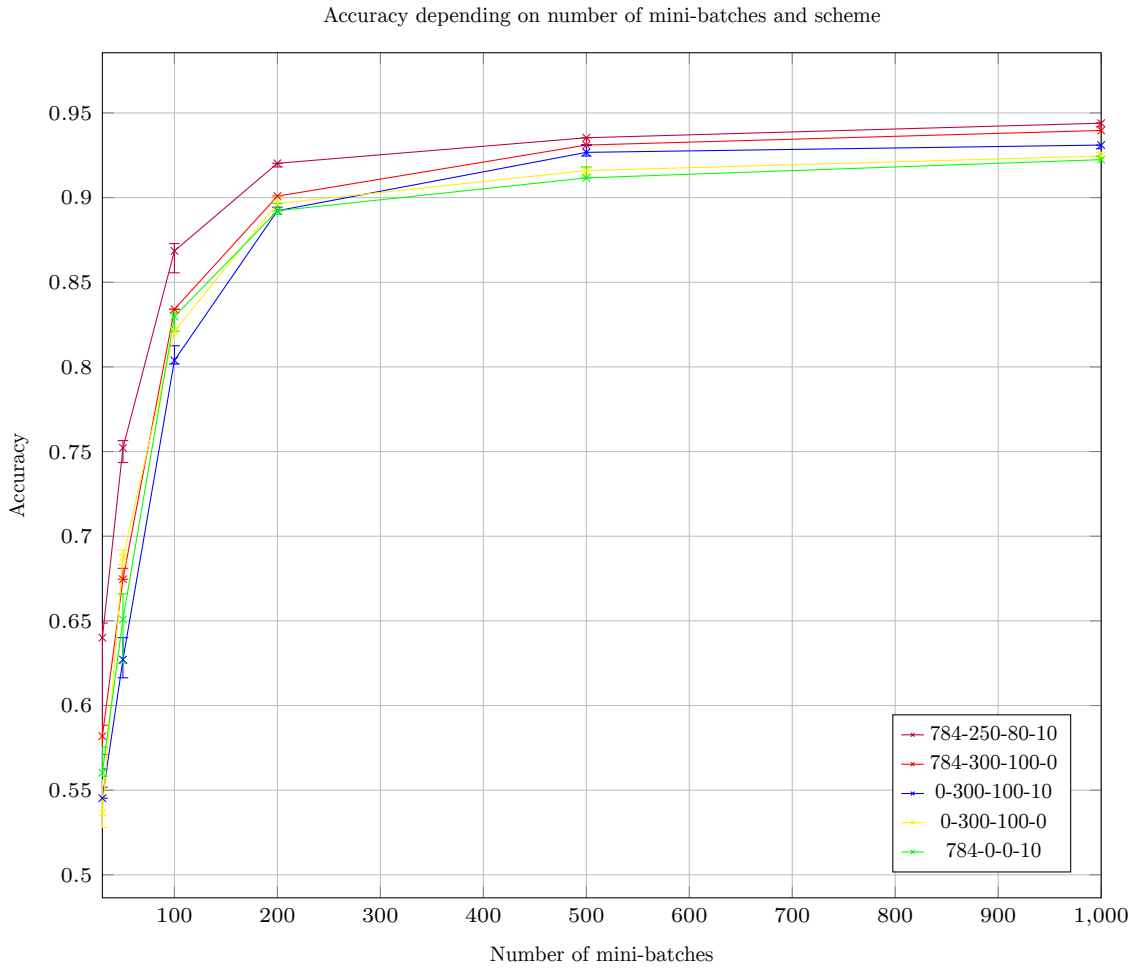


Figure 3.16: Averaging scheme on FEMNIST-D - Accuracy for various numbers of mini-batches and schemes

We see here that, obviously, the number of peers has no effect when averaging is not used. It also has a small effect with a limited averaging level but, with high level partial averaging (60-80), we get a significant performance gain when increasing the number of peers. The performance of full averaging, despite gaining from peer number increase, remains low. This indicates that partial averaging allows for performance gain when connecting more peers, giving access to more training examples.

V.3.d Semi-local averaging

Here, we want to see how semi-local models (i.e. partial models shared by several but no all peers) can improve performance, compared to just using global and local models.

For this experiment we tested different averaging schemes with different mini-batch sizes (from 25 to 150). Smaller mini-batches allowing more frequent averaging for this more difficult context. Training sets are independent and of size 200, so that the smaller mini-batches would still cover a significant portion of the training sets. We used 12 peers; a highly divisible number allowing more complex permutation patterns.

Differences between peers are induced by applying all elements of $\mathfrak{S}_3$ (permutations on a set of 3 elements) to the last 3 digits. The schemes tested are the following: no averaging, complete averaging of all peers' neural networks, complete averaging limited to peers with identical permutation, averaging with an 80 level of all peers' neural networks, averaging with an 80 level (784-250-80-10) of all peers' neural networks + averaging with a 20 level (0-50-20-0) of peers with identical permutation. For the

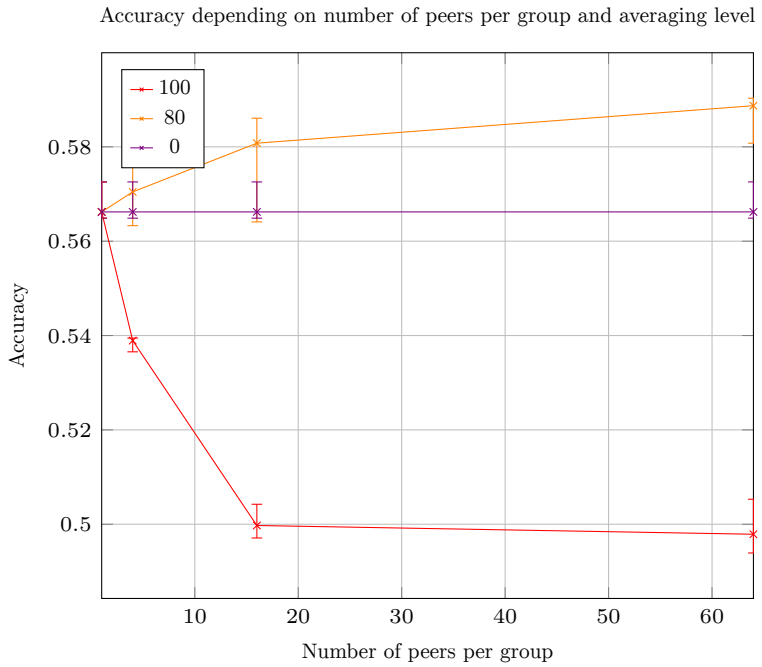


Figure 3.17: Number of peers on FEMNIST-A - Accuracy for various numbers of peers per group and averaging levels

last scheme, we tested two variants: in the first case, no model was declared dependent of another one, which meant that no inter-model weight was averaged, in the second case, the semi-local models were declared dependent on the global model, allowing inter-model weights between global and semi-local to be averaged with the semi-local models' internal weights. Each curve corresponds to a different averaging scheme. The averaging process was done after every mini-batch. 100 mini-batches were used for training. Each permutation is used for 2 peers (12 peers total).

Results are median values of 20 runs (more precise than 10, since results were close). The error bars corresponds to the second and third quintiles. The results are presented in Figure 3.23.

On this test, we observe that no averaging is the method with the worst performance. Full averaging is better but still less efficient than averaging per class or 80 averaging. Having a 80 global model + a 20 semi-local model ended-up being the best solution here and declaring the dependency allowed greater gains and declaring the dependency allowed greater gains, very significant for higher mini-bash sizes.

V.3.e Semi-local averaging with local models

In this test, we evaluate how a combination of global, semi-local and locals models performs.

For this experiment we tested different averaging schemes with different mini-batch sizes (from 100 to 800), allowing a medium averaging frequency, adapted to the difficulty of the task (complex permutation set, but with few peers). Training sets are independent and of size 1000. We used 4 peers, fewer peers implying that some would have unique permutations, the natural use-case for local models.

Differences between peers are induced by permuting digits 8 and 9 for peer 2 and 3 + digits 6 and 7 for peer 3. The schemes tested are the following: no averaging, complete averaging of all peers' neural networks, complete averaging limited to peers with identical permutation, averaging with an 80 level of all peers' neural networks, averaging with a 70 level (784-220-70-10) of all peers' neural networks + averaging with a 30 level (0-80-30-0) for peers 0 and 1 (id class, no intra-class difference) and an averaging with a 15 level (0-40-15-0) for peers 2 and 3 (semi-id class, some intra-class difference). For the last scheme, we tested two variants: in the first case, no model was declared dependent of

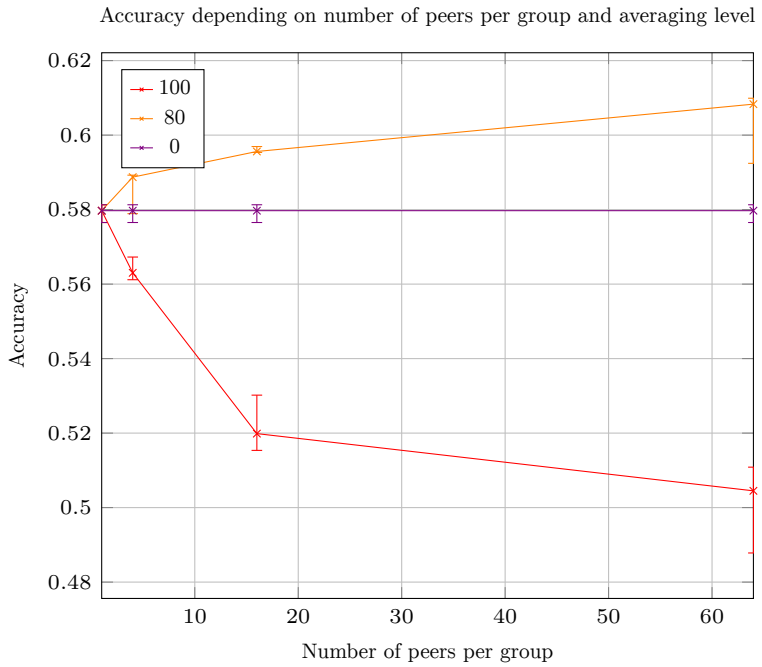


Figure 3.18: Number of peers on FEMNIST-M - Accuracy for various numbers of peers per group and averaging levels

another one, which meant that no inter-model weight was averaged, in the second case, the semi-local models were declared dependent on the global model, allowing inter-model weights between global and semi-local to be averaged with the semi-local models' internal weights. Each curve corresponds to a different averaging scheme. The averaging process was done after every mini-batch. 50 mini-batches were used for training.

Results are median values of 20 runs (more precise than 10, since results were close). The error bars corresponds to the second and third quintiles. The results are presented in Figure 3.24.

On this test, due to the important differences between peers, full averaging was clearly inferior to all other schemes, including no averaging. 80 averaging was better than no averaging, but still inferior to 100 averaging per class (pairs of peers with closer tasks, $0, 1$ and $2, 3$) with big enough mini-batches. $70 + 30$ for identical peers and 15 for peers with only one common inversion was the best and declaring the dependency allowed greater gains.

V.3.f Difference rate

Here we want to see how the efficiency of different averaging levels is affected by the level of difference between peers' tasks.

We tested different averaging levels for different difference rates. The difference rate being the number of digits affected by a different permutation between peers, ranging from 0 to 10. Training sets are independent and of size 3500; mini-batch size is 500. We used 16 peers. Each curve corresponds to a different averaging level. The averaging process was done after every mini-batch. 30 mini-batches were used for training. Differences between peers are induced by applying the cycle $(10-r...9)$ where r is the difference rate (or no permutation if $r = 0$, $r = 1$ being impossible with this system) to 7 of the 16 peers output. The results are presented in Figure 3.25.

We observe that, while 100 averaging seems to be the best when there is no difference (the gap with 80 and 60 is too low to officially conclude), it is the only scheme severely affected by the difference rate, loosing more than 50% of accuracy. For other schemes, 80 and 60 are the best for low difference rate but the gap is significantly reduced when the difference rate increases. For 10, 80 seems to be inferior to other partial averaging scheme, with a level of accuracy similar to no averaging.

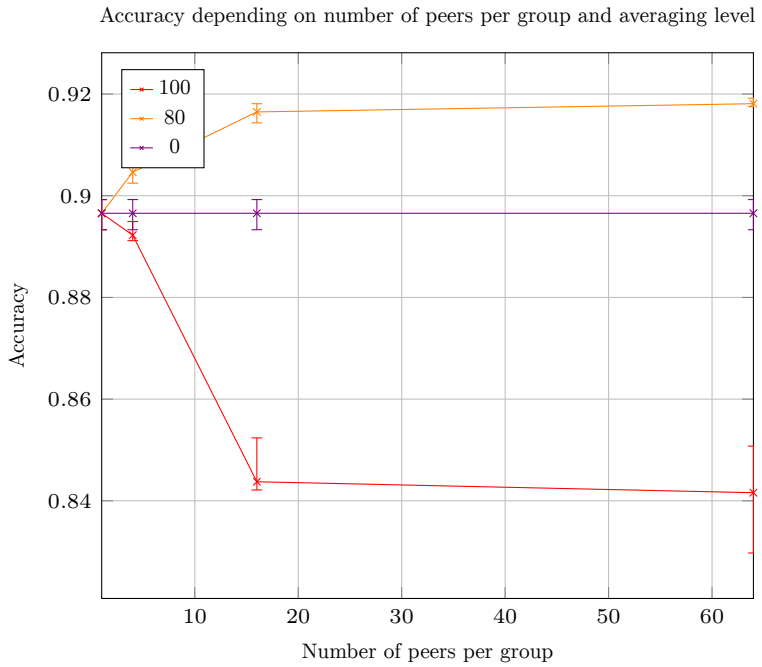


Figure 3.19: Number of peers on FEMNIST-D - Accuracy for various numbers of peers per group and averaging levels

V.3.g Number of peers with permutation

Here we want to see how the efficiency of different averaging levels is affected by the proportion of peers with a non-trivial (id) permutation.

We tested different averaging levels for number of peers with permutation. Training sets are independent and of size 3500; mini-batch size is 500. We used 16 peers. Each curve corresponds to a different averaging level. The averaging process was done after every mini-batch. 30 mini-batches were used for training. The peers with a non-trivial permutation have 8 and 9 permuted.

Results are median values of 20 runs (more precise than 10, since results were close). The error bars corresponds to the second and third quintiles. The results are presented in Figure 3.26.

We observe that full averaging's accuracy significantly drops when the number of permuted peers increases. Full averaging is the most accurate with 0 peers with permutations (as it should be) but is only third with 2 and last for 4 to 8. As one could expect, no averaging is not affected by the number of peers with permutation. More interestingly, partial averaging schemes do not seem to be much affected by the number of peers with permutation; only a limited drop is observed for 30, 60 and 80.

V.3.h Training set size

Here we want to see how the efficiency of different averaging levels is affected by the size of each peer's training set.

We tested different averaging levels for different training set sizes (ranging from 250 to 3500). The mini-batch size is fixed at 200. We used 16 peers. Each curve corresponds to a different averaging level. The averaging process was done after every mini-batch. 50 mini-batches were used for training. Differences between peers are induced by permuting digits 8 and 9 for 7 of the 16 peers. The results are presented in Figure 3.27.

We see that the lower the averaging level is, the more the training set size is important. 100 averaging is the best for 250 set size, third behind 80 and 60 for 500, beats only 0 for 1000 and last after that. 80 is the best averaging level, except for very small training sets (size 250); 60 being second.

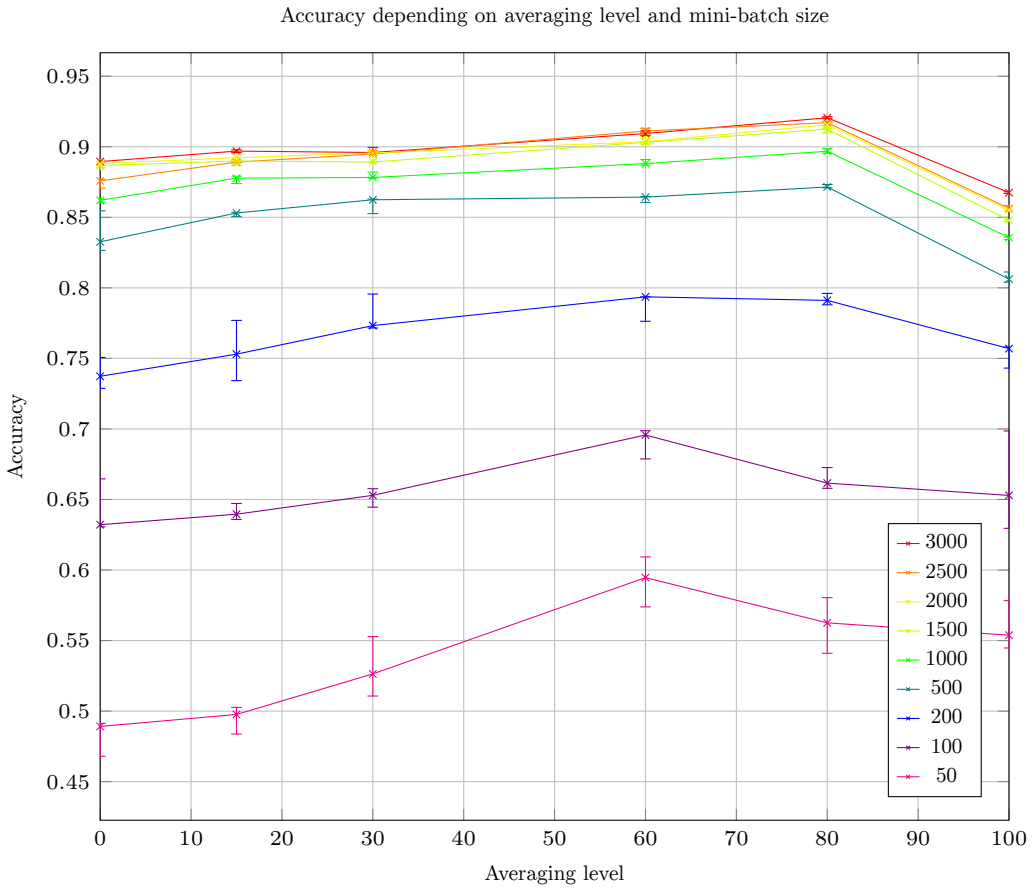


Figure 3.20: Averaging level on modified MNIST - Accuracy for various averaging levels and mini-batch sizes

V.3.i Different layouts

Due to the way we induce differences between peers, a simple post-treatment consisting in a permutation inversion would allow 100 averaging to outperform any scheme. We did not try to use this fact before to improve our averaging schemes to remain general. In this test, we evaluate how a partial averaging scheme crafted more specifically for this problem will perform.

Here we test different averaging schemes and layouts for different difference rates. The first four schemes are used with a 784=300=100=10 network layout (4 layers), while the last 4 have a 784=300=100=10=10 (5 layers) layout. Each curve corresponds to a different averaging scheme. The key difference between these two layouts is the addition of a fifth layer, which would (intuitively) learn the permutation. Training sets are independent and of size 3500; mini-batch size is 500. We used 16 peers. The averaging process was done after every mini-batch. 50 mini-batches were used for training (longer training, since some layouts are deeper than usual). Differences between peers are induced by applying the cycle $(10 - r...9)$ where r is the difference rate (or no permutation if $r = 0$, $r = 1$ being impossible with this system) to 7 of the 16 peers output. The results are presented in Figure 3.28.

Complete averaging schemes see their accuracy drop severely when the difference rate increases. The 80 variant used on 784=300=100=10=10 has an accuracy close to no averaging on the same layout, lower for high difference rates. 784=300=100=10=10 layout seems less efficient than 784=300=100=10 in general, but with a 784-300-100-10-0 averaging scheme, it outperforms no averaging on 784=300=100=10 and even (but not much significantly) 784-250-80-10 for high difference rates. 784-300-100-0 appears to be the best scheme in general, beating any other scheme significantly, except in the no difference case and being less affected than 784-250-80-10 by difference rate.

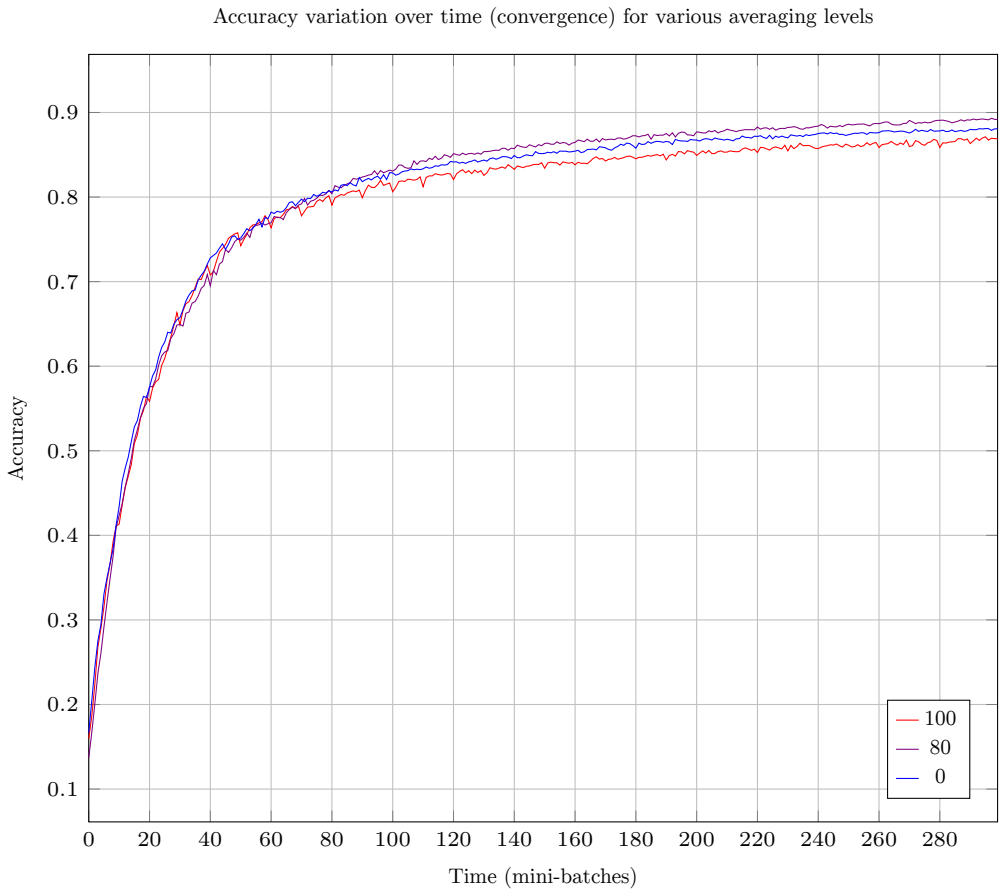


Figure 3.21: Convergence on modified MNIST - Accuracy over time (convergence) for various averaging levels

It is important to remember that the 784-300-100-0 was the best here due to the particular nature of the differences between peers, a permutation. This scheme was crafted specifically for this problem, which is a luxury, we could not have done this if the exact nature of the differences between peers was not precisely known. Moreover, this kind of layout is not easy to generalize to semi-local models.

V.4 Multi-Layer Perceptron - VSN

We finally test MLP on a significantly different task: vehicle recognition from sensors. We use a dataset from [DH04]. This dataset contains data from sensors (notably seismic and acoustic sensors) produced while some (military) vehicle passes near the sensors (grouped in nodes, which will be our peers).

The task proposed is to recognize the class of vehicle (*assault amphibious* or *dragon wagon*) from each sensor's data (binary classification). To allow an MLP to perform this task, the data from each set of sensors (node) is first transformed into 50 seismic and 50 acoustic features (100 total inputs). This process (based on Fast Fourier Transform) is described in the original paper ([DH04]).

For our distributed multi-task setup, we consider each node as a peer. The tasks are similar, since all peers want to classify the same kinds of vehicles from the same kind of data, but different due to the location of sensors influencing their output.

For this experiment we use 16 peers, each with a training set of 50 samples. The test set contains 200 samples. The mini-batch size is 25. We use an MLP with a 100=50=20=1 layout.

The "averaging level" is the number of neurons of the global model (shared by all peers), others neurons being part of local models, in the first hidden layer, which contains a total of 50 neurons.

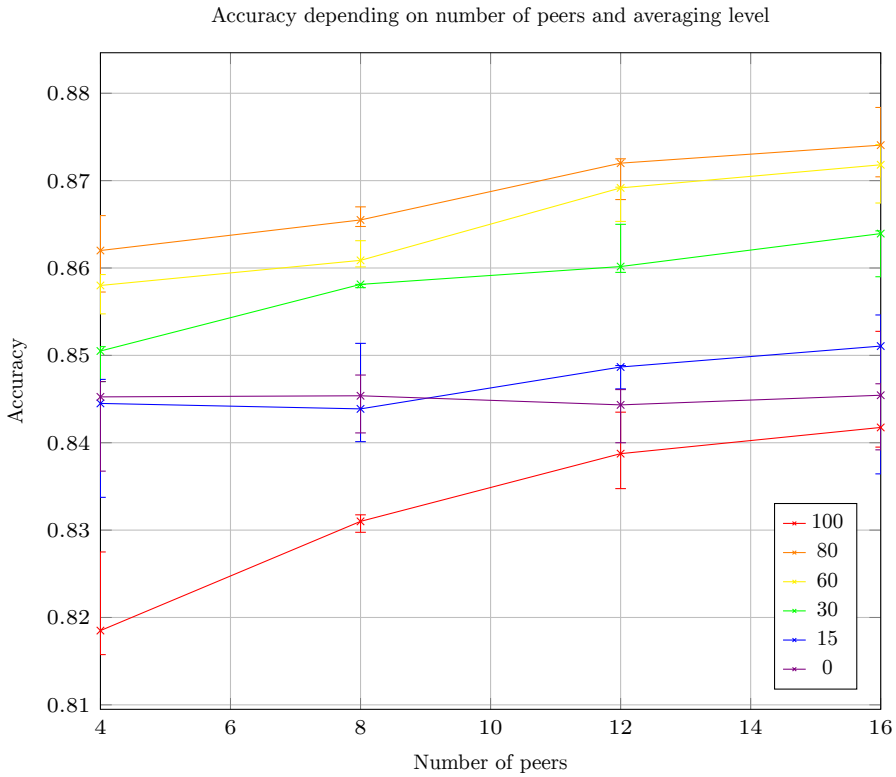


Figure 3.22: Number of peers on modified MNIST - Accuracy for various numbers of peers and averaging levels

The levels we test are 0, 7, 15, 30, 40 and 50. Some neurons from the second hidden layer are also averaged, the exact numbers are, respectively, 0, 3, 6, 12, 16, 20. Each curve corresponds to a different mini-batch number (from 100 to 400). The averaging process was done after every mini-batch.

Similarly to previous experiments, results are median values of 10 runs. The error bars corresponds to the second and third quintiles. The results are presented in Figure 3.29.

This test's results are very similar to our other MLP tests. We get an optimum around 80% averaging.

V.5 Associative neural network

All our previous experiments used MLPs with SGD; now we want to see how our system performs on a different kind of neural network, with a different training method. We test associative neural networks on the following task.

We generate binary vectors of size n ($\in 0, 1^n$) with $2m$ 1s (and $n-2m$ 0s). The task of the neural network is to be able to reconstruct each learned vector from an input with only m 1s. For example, the learned vector could be $[0, 0, 1, 0, 1, 1, 0, 0, 1]$, the neural network will be given $[0, 0, 1, 0, 0, 1, 0, 0, 0]$ as input and should be able to return the learned vector.

For training, we give the network the full vectors as input; for testing, we give the network a partial input with only m 1's and observe the output. To complicate the task, we add noise to the vectors: a fixed number z of random bits are flipped (1 becomes 0 and 0 becomes 1) in all vectors (training and testing) before feeding them to the network.

For accuracy evaluation, the task is considered successfully accomplished if the number of matching 1's between the expected output and the actual output is greater than or equal to the number of non-matching 1's. More formally ($\mathbf{c}$ is expected output, $\mathbf{a}$ is actual output), a test is considered successful if and only if $\mathbf{c} \wedge \mathbf{a}$ contains at least as many 1's as $\mathbf{c} \oplus \mathbf{a}$.

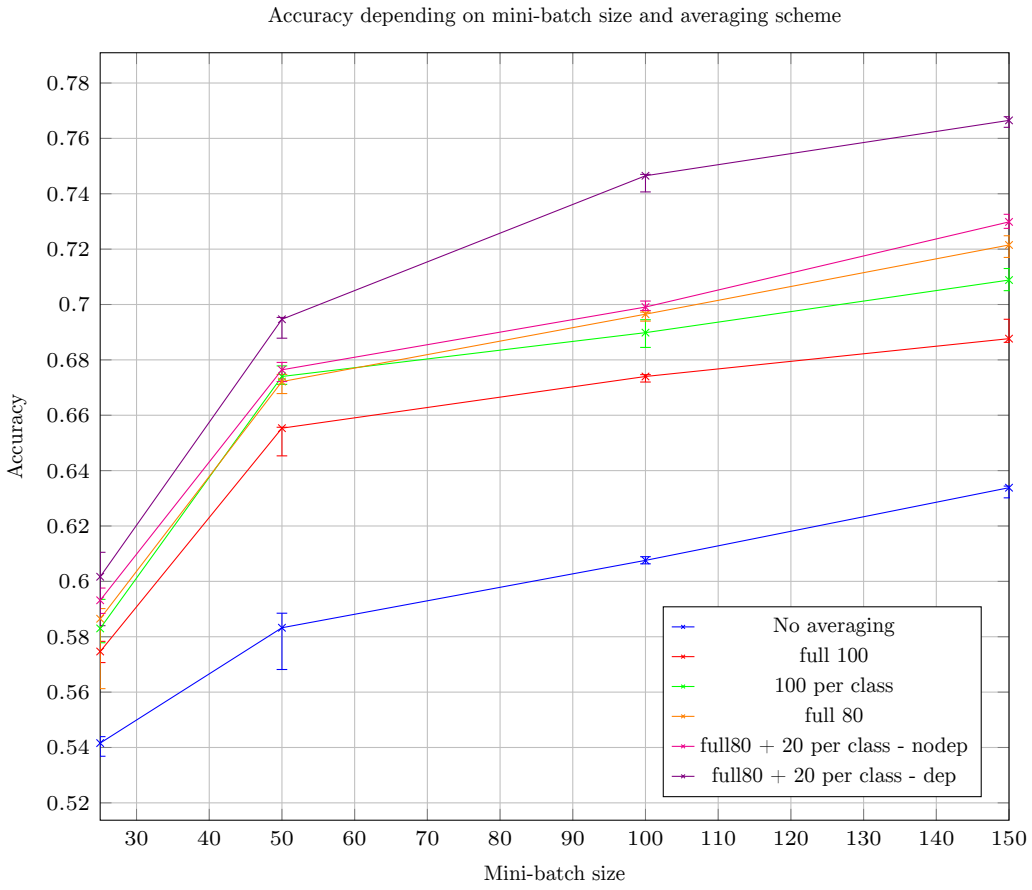


Figure 3.23: Semi-local averaging on modified MNIST - Accuracy for various mini-batch sizes and averaging schemes

To generate differences between peers, we simply changed, for some peers, half of the 1's that are not in the testing input. For example, from $[0,0,1,0,0,1,0,0,0]$, some peers will have to find $[0,0,1,0,1,1,0,0,1]$ and others $[1,0,1,0,0,1,1,0,0]$.

Our vector generator works in the following way: it takes as input an integer $k \in \mathbb{N}$ and has a periodicity parameter s. The 1s of the vector that will be given as input for both training and testing corresponds to the coordinates $3m(k[s]) + i$ (.[.] is the modulo operation: $k[s]$ is the remainder of the Euclidean division of k by s) with i ranging from 0 to $m-1$. The 1s of the vector that will be given as input only for training (that should be guessed for testing) corresponds to the coordinates $3m(k[s])+i$ with i ranging from m to $2m-1$ in the general case and, in the modified case (when we want to make a peer different), with i ranging from $2m$ to $3m-1$. This allows use to general pseudo-random sparse binary vectors which are similar but not identical among all peers. To add noise, we simply flip bits with coordinates $((k/s)+i(n/z))[n]$ (/ representing integer division) with i ranging from 0 to $z-1$.

For this test, we use 15 peers, 2 of them having 1 out of 3 vectors modified (i ranging from $2m$ to $3m-1$ rather than m to $2m-1$). Each vector is 50 bits long, with $m = 8$ (non-zero bits) and $z = 2$ (noise). The periodicity of the generators is 3, the batch size is 9 ($k \in [\![0, 9[\![$) and the mini-batch size is 3. The tests consider 100 consecutive values of k (starting at 1000). The learning rate is 0.1 and the activation threshold for neurons is 0.5. The visible layer has 50 neurons (the length of an input/output vector), while the hidden layer comprises 400 neurons.

We compare no averaging at all, complete averaging of both layers (visible and hidden) and several partial averaging schemes. For partial averaging schemes, the visible layer was completely averaged and 100, 200, 300 or 350 neurons of the hidden layer were averaged. Each curve corresponds to a

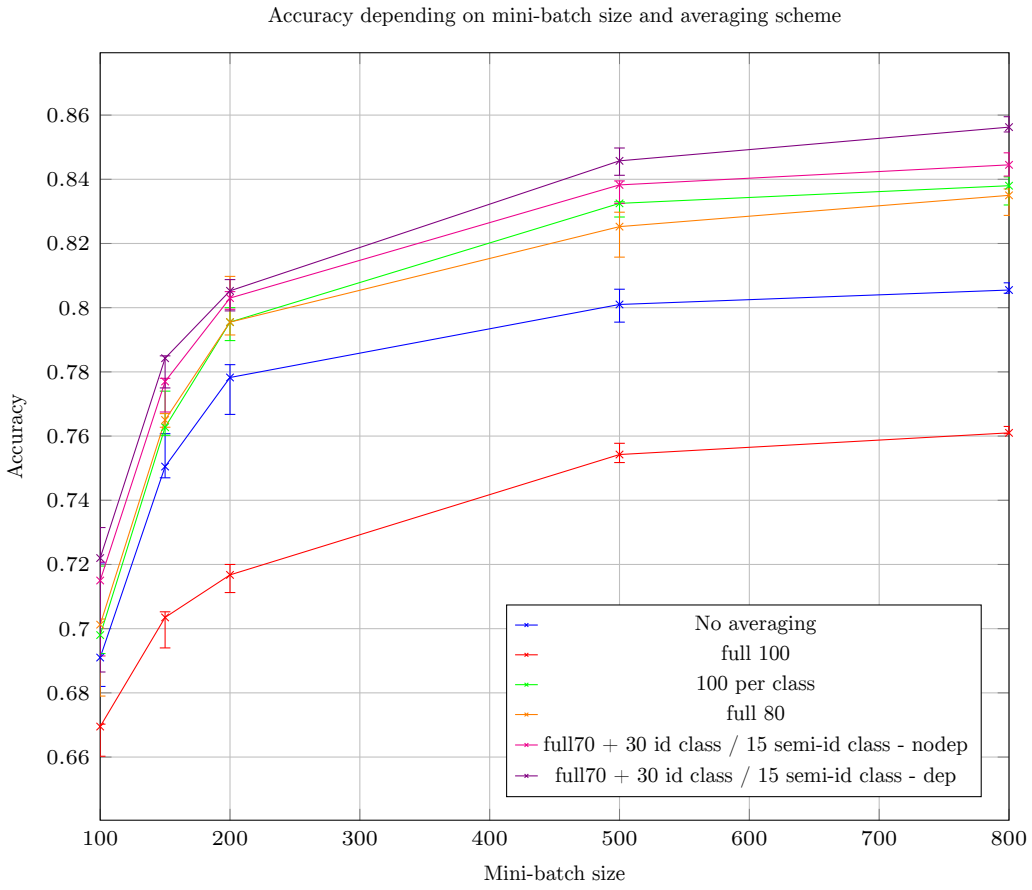


Figure 3.24: Semi-local averaging with local models on modified MNIST - Accuracy for various mini-batch sizes and averaging schemes

particular number of mini-batches used: 10, 20, 30 or 40. Averaging was done after each mini-batch.

Results are median values of 100 runs. The error bars corresponds to the second and third quintiles. The results are presented in Figure 3.30.

We observe here that an averaging level of 300 (over 400) seems optimal for lower numbers of mini-batches (10,20), while 350 is for higher numbers (30,40). Except for 40 mini-batches, partial averaging schemes are always the best. Complete averaging was the worst for all tests but the 40 mini-batches one. For 40 mini-batches, complete averaging, while still worse than partial 350, is better than most partial averaging schemes. For all numbers of mini-batches, the optimal averaging level remained around 300-350. These results with associative neural network are consistent with those obtained with MLP, which indicates that our system works similarly on these different kinds of neural networks.

V.6 General analysis of results

The various experiments we made, and their parameters, are summarized in Table 3.1. Our experiments show that our method effectively enables distributed multi-task learning with neural networks. From the presented results, we can make the following detailed statements.

Partial model averaging yields better results than no averaging or complete averaging on different tasks, synthetic and real, with a variety of parameters, different kinds of neural networks and learning algorithms.

Among the considered tasks, a high level of averaging (60-80% of hidden layers) usually turns out to be the best solution and the level of difference between peers' tasks only has a low influence on the

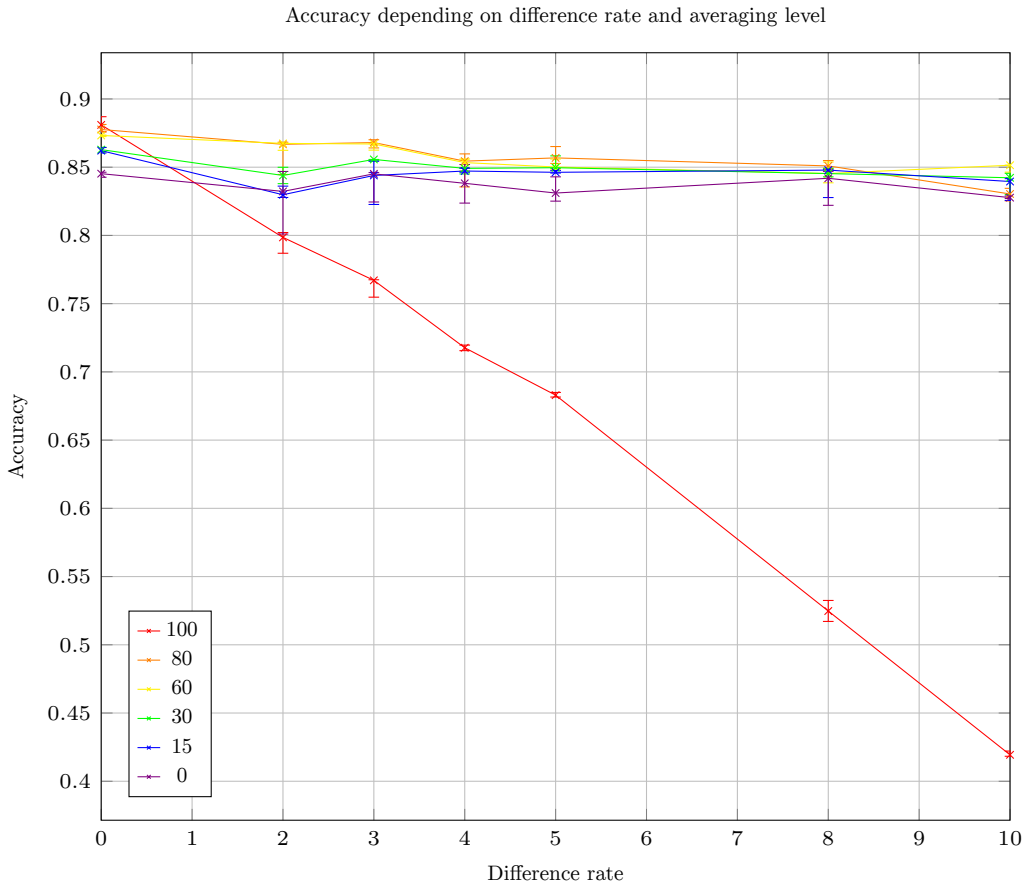


Figure 3.25: Difference rate on modified MNIST - Accuracy for various difference rates and averaging levels

optimal averaging level.

Semi-local averaging and the associated dependency system further improve performance when groups of peers share more similar functions than the whole system.

Multi-task learning with MLP benefits in FEMNIST from the possibility to share portions of layers rather than just complete layers, while more task-specific layouts, averaging limited to specific layers, have proved to be more efficient in modified MNIST but require more knowledge about the differences between peers. A mix of both kinds of schemes could be interesting in certain cases.

While those tests are done on simple cases, they allowed us to examine numerous parameters changes and their effect on the efficiency of our method. This makes this chapter a good base for applying our work to more complex real use cases.

VI Conclusion

In this chapter, we have introduced an effective solution for distributed multi-task learning with neural networks, applicable to different kinds of learning algorithms and not requiring mandatory prior knowledge about the nature, nor magnitude, of the differences between tasks, while still being able to benefit from such knowledge when available. The simplicity, range of application, and flexibility of our method significantly distinguishes it from existing works.

This work also opens several new directions in different fields. First, in the context of machine learning, we plan to design an algorithm that allows peers to be grouped automatically to share semi-local models. Second, it would be interesting to reduce communication overhead similarly to

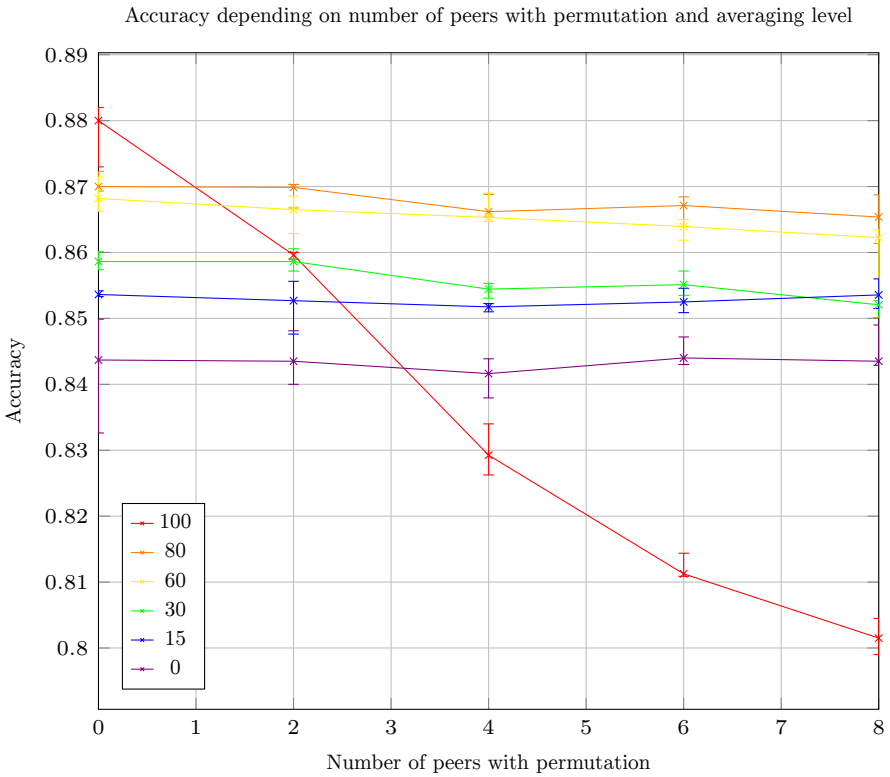


Figure 3.26: Number of peers with permutation on modified MNIST - Accuracy for various numbers of peers with permutation and averaging levels

[Kam+18]. Third, it would be interesting to apply our method to other kinds of neural networks, like LSTM [HS97; Ger99; GSC00]. One may also want to extend our experiments to other datasets and ranges of parameters and compare to more competitors, like the configurations presented in [Mar+21], including several cases where federated averaging exhibits great performance compared to our experiments; it would be interesting to see how our system performs is such configurations.

In the next chapter of this work, we explore the first direction: automatically assign models to peers, using clustering-based approach.

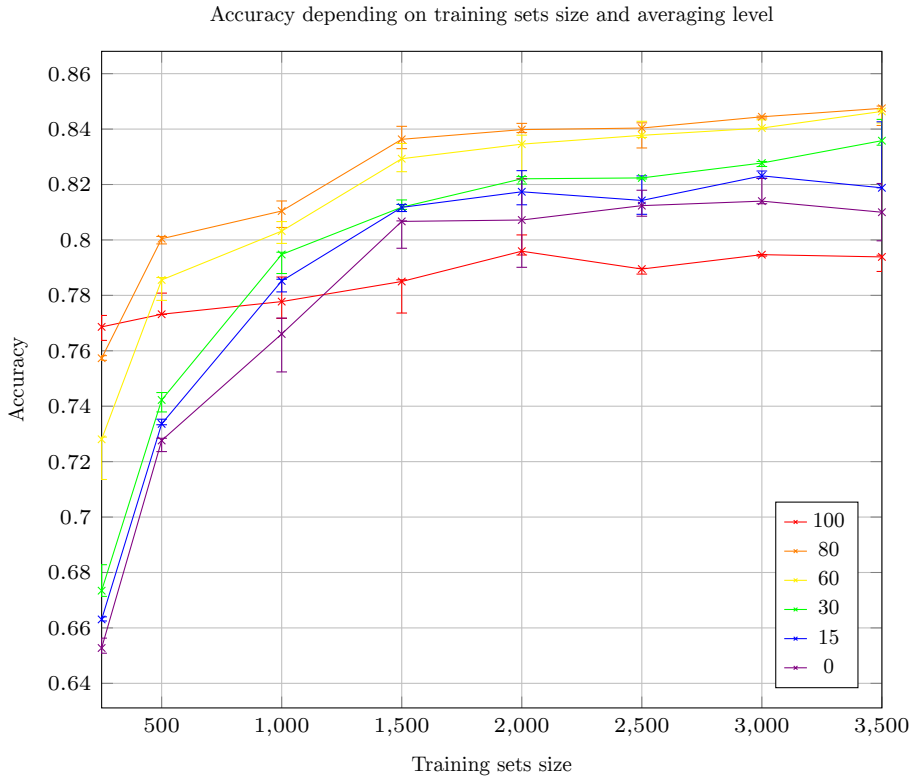


Figure 3.27: Training set size on modified MNIST - Accuracy for various training sets sizes and averaging levels

Tests recapitulation ("Classic" is 784=300=100=#classes)								
Task [NN] (figure)	Peers	Batch size	MB size	#MB	Avg every	Layout	Averaging	Difference [#peers]
FEMNIST-A [MLP] (3.11)	16	360	180	Varies	1	Classic	Varies	Natural
FEMNIST-M [MLP] (3.12)	16	360	180	Varies	1	Classic	Varies	Natural
FEMNIST-D [MLP] (3.13)	16	100	50	Varies	1	Classic	Varies	Natural
FEMNIST-A [MLP] (3.14)	16	360	180	Varies	1	Classic	Varies	Natural
FEMNIST-M [MLP] (3.15)	16	360	180	Varies	1	Classic	Varies	Natural
FEMNIST-D [MLP] (3.16)	16	100	50	Varies	1	Classic	Varies	Natural
FEMNIST-A [MLP] (3.17)	Varies	350	175	200	1	Classic	Varies	Natural
FEMNIST-M [MLP] (3.18)	Varies	350	175	200	1	Classic	Varies	Natural
FEMNIST-D [MLP] (3.19)	Varies	100	50	200	1	Classic	Varies	Natural
MNIST [MLP] (3.20)	16	3500	Varies	30	1	Classic	Varies	(89) [7]
MNIST [MLP] (3.21)	16	3500	100	30	10	Classic	Varies	(89) [7]
MNIST [MLP] (3.22)	Varies	3500	500	30	1	Classic	Varies	(89) [1/4]
MNIST [MLP] (3.23)	12	200	Varies	100	1	Classic	Varies	$\mathfrak{S}_3$ [2 each]
MNIST [MLP] (3.24)	4	1000	Varies	50	1	Classic	Varies	(89) [1] + (67)(89) [1]
MNIST [MLP] (3.25)	16	3500	500	30	1	Classic	Varies	Varies [7]
MNIST [MLP] (3.26)	16	3500	500	30	1	Classic	Varies	(89) [Varies]
MNIST [MLP] (3.27)	16	Varies	200	50	1	Classic	Varies	(89) [7]
MNIST [MLP] (3.28)	16	3500	500	50	1	Varies	Varies	Varies [7]
VSN [MLP] (3.29)	16	50	25	Varies	1	100=50=20=1	Varies	Natural
SBV compl [Assoc] (3.30)	15	9	3	Varies	1	50<>400	Varies	Half 1s [2]

Table 3.1: Summary of experiments

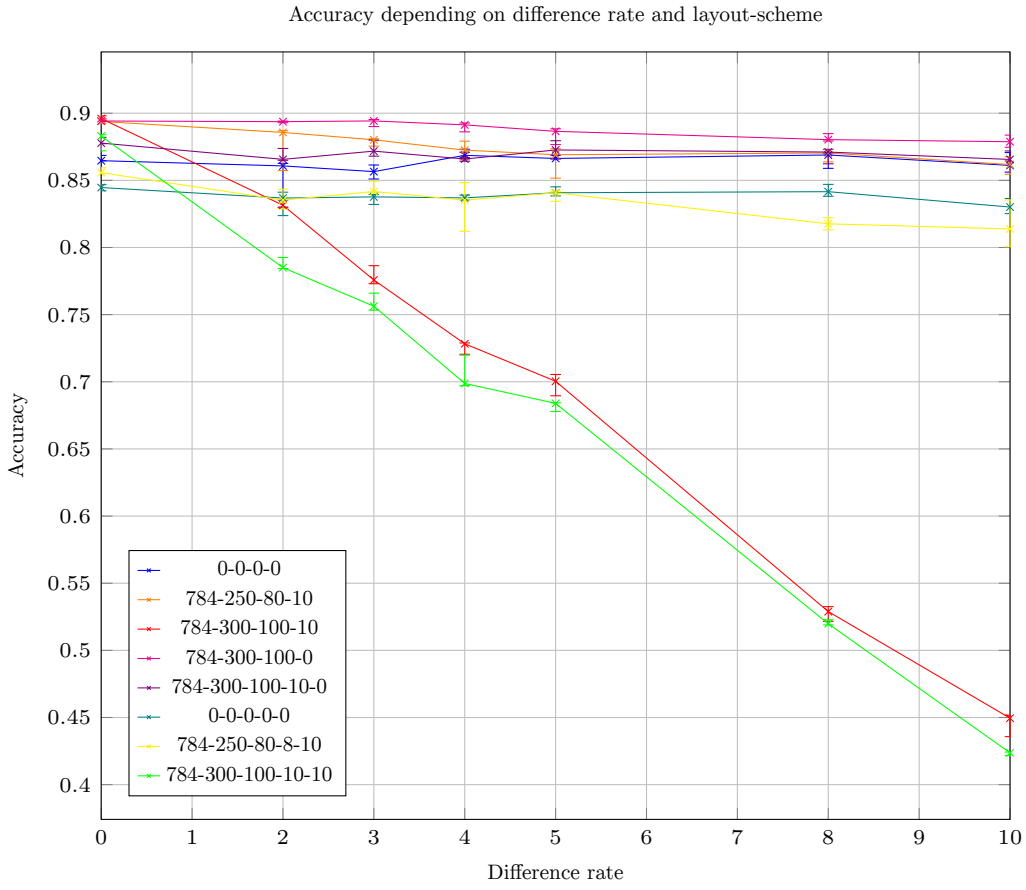


Figure 3.28: Different layouts on modified MNIST - Accuracy for various difference rates and layouts-schemes

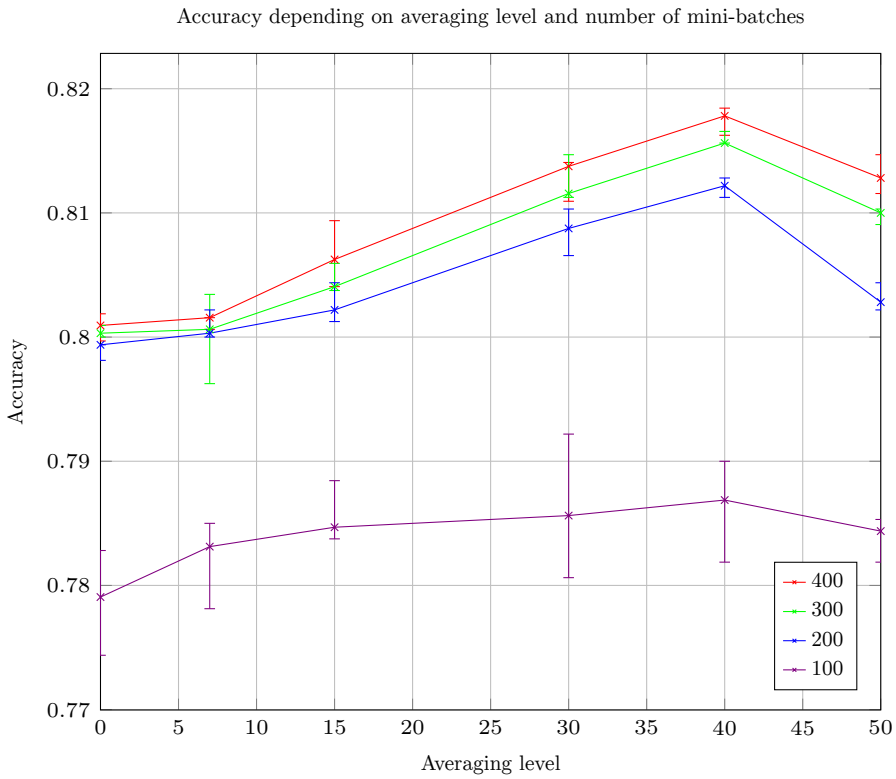


Figure 3.29: Averaging on VSN - Accuracy for various averaging levels and numbers of mini-batches

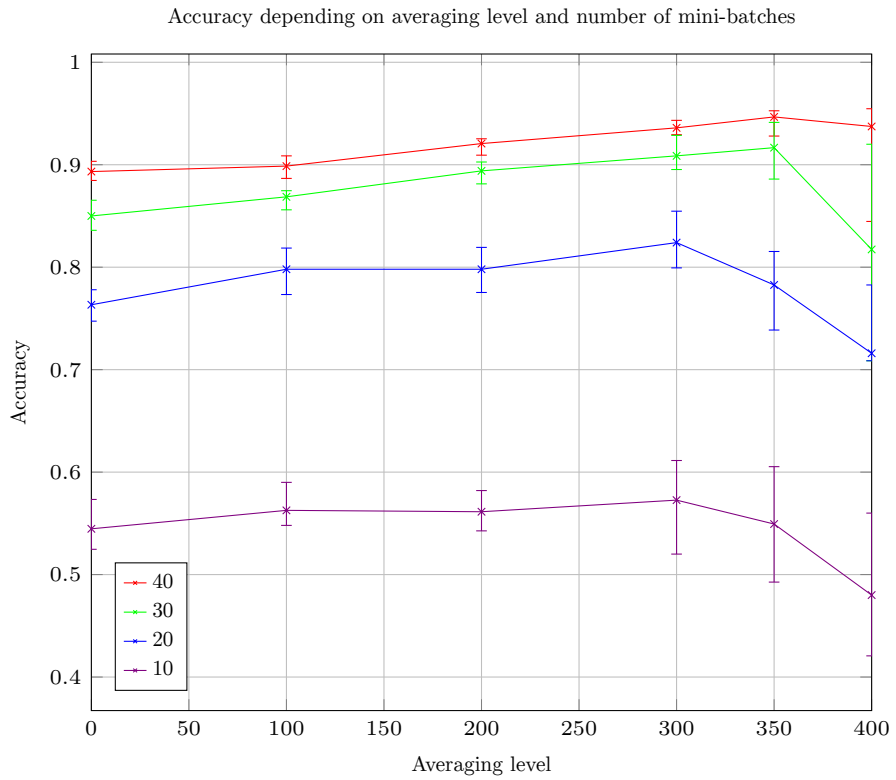


Figure 3.30: Associative neural network - Accuracy for various averaging levels and number of mini-batches

Chapter 4

AUCCCR: Agent Utility Centered Clustering for Cooperation Recommendation

I Introduction

In this chapter, we examine the problem of automatically assigning models to peers for our distributed multi-task machine learning system. To maximize the impact of this work, we decided not to focus on machine learning but to consider a more general problem. Model assignment for distributed multi-task machine learning is only an instance of a problem we will refer to as *cooperation recommendation*.

We define cooperation recommendation as follows. We have a set of agents (people, organizations...) who want to perform similar tasks that can be done collaboratively (machine learning, grouped orders...). These agents may benefit from collaboration but only if other agents have sufficiently similar expectations (function to learn, goods to buy) and are sufficiently numerous (larger groups compensate divergences). Agents would like to know who they could collaborate with and think in terms of individual interest; they can and will refuse to collaborate if the proposed recommendation does not benefit them individually. Given data on agents preferences, we want to provide cooperation recommendations that agents will accept.

This definition leads us to a formalism based on economics and prescriptive decision theory [vM44][1]. The utility an agent obtains from belonging to a group is positively correlated to the group's size (larger is better), but negatively correlated with its distance from the group's barycenter (closer is better). Such a model can capture various practical problems: in collaborative machine learning, for instance, learners with similar but different tasks may or may not collaborate with each other, depending on the effect of this collaboration on the efficiency of their learning process; in grouped purchases, potential buyers will search for other people with similar buying habits to save money by placing grouped orders, but will not benefit if the products bought deviates too much from their preferred options; when planing organized vacations, most people want to save money with group rates, but not at the cost of visiting too many places they are not interested in. If agents are too unsatisfied by the recommendation (worse than being alone), they may reject it. We also consider the case where people want to be in groups that are large enough but not too large.

An algorithm that solves this problem should provide collaboration recommendations (e.g. with whom each agent should share parts of a machine learning model or perform a grouped order) that agents find acceptable and from which they do not deviate. Since, in this context, the relative utility of different options for an agent depends on the choices of other agents, it is crucial to prevent situations

[1] While descriptive decision theory has shifted to the more exact Prospect theory [KT79], this work is about automated decision support and, thus, based on expected utility [Ran+07].

where a few agents reject the recommendation, as this could lead to a complete collapse of the solution. The departure of a dissatisfied agent can decrease the group's value for other agents, which may in turn leave the group, etc.

In economic terms, this problem can be modeled as a *hedonic game* [DG80], but this formalization tends unfortunately to be too general to allow for effective algorithmic solutions. Existing algorithms that solve generic hedonic games, summarized in [AS16], need to constrain them by requiring the existence of some kind of equilibrium (Nash equilibrium or a similar definition), which is not guaranteed by the generic definition of hedonic games and may not exist in practice. In our work, we instead consider a more restricted class of games, where agents' utility is given by a certain formula, and we provide an heuristic that also works in cases where no kind of equilibrium exists (though the solution is necessarily imperfect).

Hedonic games designed for specific problems have been proposed in the past, for example in [Saa+10], but the practical problems they consider do not apply to our context. The cited example presents the case of roadside units in intelligent transportation systems. In particular, these works tend to adopt a formalization that ensures the existence of some kind of equilibrium, while we want solutions for cases where no equilibrium exists.

Our insight is that the hedonic games corresponding to our coalition-formation problem can be interpreted as a clustering problem. We want to identify groups of close and numerous agents, which is essentially a clustering task. However, commonly used clustering algorithms, while technically applicable, are not adapted to this particular task, as they can not work with the kind of utility formula we want to use. To address this gap, this chapter proposes a novel clustering algorithm, that is specifically designed to address the task we want to solve.

We begin this chapter with a formal definition of the class of hedonic games we use to model collaboration. We then propose AUCCCR (pronounced "okr", IPA: [okʁ]), a clustering algorithm able to provide solutions to the considered problem. We present a theoretical and experimental evaluation of our algorithm on both synthetic and real data. From this evaluation, we conclude that our algorithm respects individual agents interests better than other clustering algorithms.

II Our approach

II.1 Problem statement and formalization

We consider a set of agents $a_0, ..., a_m \in A$ that seek to form groups so that every agent in a group effectively benefits from belonging to this group. We represent agents' preferences using an ℓ-dimensional real vector given by a projector function, $p : A \to \mathbb{R}^\ell$, which allows us to measure similarity between agents as a distance. In our model, the benefit from being in a group depends on two factors: the size of the group (the larger the better), and the similarity between an agent and the (average of) the group (the more similar the better). In practice, the chosen projector function is likely to associate each agent with a vector of real values taken from a database; for example (what is used in our experiments) statistics on review for holiday destinations or annual spending in different types of goods.

More concretely, we define a *utility* function $U_p(a, g)$ that expresses the interest of an agent, a, in a group, g, using a projector, p, as the product of two factors: *(i)* the value of the group (function v), which grows with the group's size; and *(ii)* a decreasing function ($n(...)$) of the distance between an agent and the group's barycenter (as measured by a distance function $d(...)$). This is formally captured by the following formula:

$$U_p(a, g) : A \times \mathcal{P}(A) \to \mathbb{R}^+ = n\Big(d(p(a), bary_p(g))\Big) \times v(\#g) \tag{4.1}$$

Where $\mathcal{P}(A)$ is the power set of A, $\#g$ is the size of group g, $bary_p(g)$ is the barycenter of group g with projector p, and the functions d, v, and n are defined as follows: $d(x, y) : \mathbb{R}^l \times \mathbb{R}^l \to \mathbb{R}$ is the

Ident	Description
$d(x, y)$	distance between x and y
$p(a)$	projector function
$bary_p(g)$	barycenter of group g with projector p
A	set of agents

Table 4.1: Symbols used in formulas and algorithms

distance between x and y; $v(n) : \mathbb{N} \to [1, +\infty[$ is the *value* of a group of size n (increasing, $v(1) = 1$); and $n(d) : \mathbb{R}^+ \to [0, 1]$ is the *normalizer* for an agent at distance d from its group's barycenter (decreasing, $n(0) = 1$). In the following we will also use $g(a)$ to denote the group of agent a. These definitions associate a utility value of 1 with an agent that remains alone (since the interest of joining a group with a barycenter at distance 0 and consisting of 1 element (itself) is $n(0) \times v(1) = 1 \times 1$).

Given A, $p()$, $d()$, $n()$ and $v()$, the group formation problem we seek to solve consists in finding a partition of A (representing the groups), that maximizes the sum of every agent's utility, while minimizing (or even eliminating) the benefit individual agents could gain by either changing group or remaining alone (to ensure the solution's stability). Table 4.1 summarizes these notations.

More formally, our objective is:
Given A (agents), $p()$ (projector function), $d()$ (distance metric), $n()$ (normalizer) and $v()$ (generic group value for given size).
With $U_p(a, g)$ defined as above ($U_p(a, g) : A \times \mathcal{P}(A) \to \mathbb{R}^+ = n\Big(d(p(a), bary_p(g))\Big) \times v(\#g)$).
Find some group affectation $g()$ ($g(a)$ may be "no group").
That maximize $\sum_{a \in A} U_p(a, g(a))$.
Subject to $\forall_{a \in A} \neg\exists_{g'} U_p(a, g(a)) < U_p(a, g')$ (g' may be "no group").
Or (if the above condition can not be meet) minimizing $\sum_{a \in A} \max_{g'}(U_p(a, g') - U_p(a, g(a)))$.

II.2 Algorithm

In this section, we present our algorithms and define them precisely. In the following pseudocode, the member function $*.invertkv()$ inverts keys and values of an associative container (data structure) with unique values, the member function $*.group(x)$ returns the set of all elements in the container associated with the same key as x, singleton x if x is not in the container. Table 4.2 summarizes these and defines the variables and parameters appearing in algorithms.

Algorithm 6 presents the control function (loop) of our group recommendation algorithm. It loops over the parameter k (number of groups), increasing it until no significant global utility (sum of all agents utility) gains have been achieved. To reduce the randomness of the process (our clustering algorithm being randomized) each k value is tried several times (parameter prc) and the loop will only terminate if no gain happen for several k increments (momentum, parameter mmt).

Algorithm 7 is based on k-means[Llo82]. The first step is the initialization of groups, which is a k-means++[AV07] initialization (each group is given one single random member sequentially, with probability increasing with the distance to existing groups). After initialization, the process consists of two nested loops (inner and outer). Since, unlike k-means, we take group size into account, we need two loops, one for distance changes (as k-means) and one for group size changes (the inner loop). In the inner loop, at initialization, each group, has a fixed barycenter and is given a potential size of $\#A$. At each turn, agents compute their utility for each group with the potential sizes and choose their group accordingly. After that, the potential size of each group is set to the number of agents choosing this group, the barycenters are not updated. The process is repeated until no change occur in one turn. Working on the basis of potential size (i.e. maximum number of agents potentially interested in joining a group) rather than actual size resolves the "chicken or the egg" problem caused by the fact that, to appeal to potential members in terms of utility, a group needs to be already large enough. In the outer loop, at each turn, the inner loop is executed and the barycenters are updated. The loop

Ident	Description
k	number of clusters
$[\![a, b]\!]$	the range of integers from a to b (inclusive)
$\#A$	size of set A
$d.invertkv()$	key-value inverted version of dictionary d
$d.group(x)$	set of values associated with the same key as value x
$rand(A)$	Random (uniform) element of A
$rand(A, d)$	Random element of A with distribution d
$dmin2$	k-means++ distribution
grp	groups
$ngrp$	new groups (in building)
$size$	groups' sizes
$nsize$	new groups' sizes (in building)
$bgrp$	estimated best groups for each agent (inner loop)
val	affectation's value (sum of utilities)
$nval$	new affectation's (just built) value (sum of utilities)
$\perp$	no cluster

Table 4.2: Symbols used in algorithms

terminates if no change occurred in one turn.

Compared with k-means, the interesting part of our algorithm is the inner loop. Since, unlike k-means, our algorithm take the size of groups into account, we need this inner loop. It allows us to compute agents' utility with bigger groups. For agents to join groups, these groups have to be big enough and for groups to be big, agents have to join them. Our inner loop, with its over-evaluated groups' initial potential sizes, breaks this circle. The main limitation of this algorithm is that we do not have a theoretical termination proof for it. It could potentially, in very rare cases, get stuck in an infinite loop. So, we propose variation with guaranteed termination in Algorithm 8.

Algorithm 8 terminates the inner loop if the total number of agents member of any group does not change in one turn and the outer loop if the global utility of the affectation does not increase in one turn. This way, the algorithm is guaranteed to converge.

Both algorithms include a somewhat complex formula to compute agent's interest. For large instances, with very big clusters, individual agents influence on a cluster can be considered negligible. We call this situation "agents' atomicity" (agents are "atoms", too small relatively to the whole system to have significant influence individually) and use the adjective *atomic* to qualify things liked to this property[2]. Thus, we can make an agents' atomicity hypothesis to simplify the formulas. $n\Big(d(p(a), bary(grp[i] \cup a))\Big) \times v(size[i] + \mathbb{1}_{a \notin grp[i]})$ can be replaced by $n\Big(d(p(a), bary(grp[i]))\Big) \times v(size[i])$. This possibility may be used for a production implementation.

III Theoretical analysis

Here we prove the essential property of the clustering algorithm presented in Algorithm 7: equilibrium. After which, we consider the one of Algorithm 8: convergence.

Property (Nash equilibrium)**.** *If we consider a game where each agent's (player) possible choices are k different groups to join, or not choosing any group, and each agent's, a, utility (payoff) for choosing group g is given by $U_p(a, g)$ as defined before and 1 for no choosing any group, the affectation produced by Algorithm 7, if it exists, is a (weak) Nash equilibrium for this game.*

[2]The term "atomic" here is not to be understood as commonly used by computer scientists in the field of concurrency; this terminology comes from economics where it is used with a similar meaning as in this work, to describe competitive markets

Algorithm 6: Recommendation algorithm

Data: A set of agents, p projector, prc is the number of (random) initial values to try, mmt momentum effect
Result: A pair of a group affectations of agents and its value

1 $val \leftarrow \#A$
2 $k \leftarrow 1$
3 $pmm \leftarrow mmt$
4 **repeat**
5 $pval \leftarrow val$
6 $val \leftarrow \#A$
7 **repeat**
8 $afect \leftarrow \text{CLUSTER}(A, p, k)$
9 $nval \leftarrow \sum_{a \in A} n\Big(d(p(a), bary_p(afect.group(a)))\Big) \times v(\#afect.group(a))$
10 **if** $nval > val$ **then**
11 **if** $nval > pval$ **then**
12 $sol = afect$
13 $val \leftarrow nval$
14 **until** prc *times*
15 $k++$
16 **if** $val > pval$ **then**
17 $pmm \leftarrow mmt$
18 **else**
19 $pmm--$
20 **until** $pmm = 0$
21 **return** $(sol, pval)$

Proof (Nash equilibrium)**.** We assume that our algorithm as converged, we consider the last iteration of the loop guarded by $ngrp = grp$ (line 11) and, in this iteration, the last iteration of the loop guarded by $nsize = size$ (line 15).
Let's assume that the output of the function, $ngrp = grp$, is not a (weak) Nash equilibrium. Then $\exists a \in A, g \in \perp \cup [\![0, k-1]\!](U_p(a, g) > U_p(a, grp.group(a)))$.
Since at the end of the loop iteration $nsize = size$ and $nsize$ is, by design, the vector of the sizes of the groups given by $ngrp$, we can affirm that, during the considered loop iteration (assimilating i with a group), $U_p(a, grp[i]) = n\Big(d(p(a), bary(grp[i] \cup a))\Big) \times v(size[i] + \mathbb{1}_{a \notin grp[i]})$.
Since, by design, our algorithm selects i to maximize $n\Big(d(p(a), bary(grp[i] \cup a))\Big) \times v(size[i] + \mathbb{1}_{a \notin grp[i]})$, we can affirm that, in this loop iteration, a different group would have been selected for at least one a. Now, either that changes the size of at least one group, implying $nsize \neq size$ at the end of the loop, which is a contradiction. Or, if the sizes remains identical (exchange of members), then, the loop guarded by $nsize = size$ exists, and we get $ngrp \neq grp$, which also is a contradiction.□

Since the existence of a Nash equilibrium for this game is not guaranteed, it is possible that Algorithm 7 does not terminate. Now, we prove that Algorithm 8, for which we do not have an equilibrium proof, will always terminate.

Property (Convergence)**.** *Algorithm 8 always terminates.*

Proof (Convergence)**.** Among the control structures used in Algorithm 8, the two repeat...until loops are the only that do not trivially terminate (for loops terminate trivially). For the inner loop, we

Algorithm 7: Clustering algorithm

function CLUSTER(A, p, k) **is**

  /* k-means++ initialization */

  $ra \leftarrow rand(A)$

  $ngrp[0] \leftarrow \{ra\}$

  foreach $a \in A$ **do**

    $dmin2[a] \leftarrow d(p(a), p(ra))^2$

  for $1 \leqslant i < k$ **do**

    $na \leftarrow rand(A, dmin2)$

      /* random element from A, relatives probabilities given by $dmin2$ */

    $ngrp[i] \leftarrow \{na\}$ // appends $\{na\}$ to ngrp

    foreach $a \in A$ **do**

      $dmin2[a] \leftarrow \min(d(p(a), p(na))^2, dmin2[a])$

  /* main loop */

  repeat

    $grp \leftarrow ngrp$

    $ngrp.clear()$

    $nsize \leftarrow [\#A...\#A]$ // vector length = k

    repeat

      $size \leftarrow nsize$

      $nsize \leftarrow [0...0]$

      foreach $a \in A$ **do**

        $bgrp[a] \leftarrow \text{argmax}_{i \in \bot \cup [\![0,k-1]\!]}(n\Big(d(p(a), bary_p(grp[i] \cup a))\Big) \times v(size[i] + \mathbb{1}_{a \notin grp[i]}))$

        /* assuming $grp[\bot] = \emptyset$ and $size[\bot] = 0$ */

        $nsize[bgrp[a]] + +$ // does nothing if $bgrp[a] = \bot$

    until $nsize = size$

    $ngrp \leftarrow bgrp.invertkv()$ // key-value inversion

  until $ngrp = grp$

  return $ngrp$

replaced $nsize = size$ by $nsize.sum() \geqslant size.sum()$ as termination condition. This implies that, during the loop, $size.sum()$'s values are a strictly decreasing natural ($\mathbb{N}$) sequence. Such a sequence can not be infinite, thus, the inner loop terminates. For the outer loop, we replaced $ngrp = grp$ by $nval \leqslant val$ as termination condition. This implies that, during the loop, val's values are a strictly increasing real sequence. Moreover, those values are given by a formula taking as parameter an affectation of a finite number of agents in a finite number of groups. The possible inputs for this formula for a given execution of the algorithm (fixed parameters) are a finite set. This implies that the possible values for val (output of this formula) are also a finite set (for a given execution of the algorithm). Thus, val's values are a strictly increasing sequence of elements of a finite set (the order is given by the classical order for real number). Such a sequence can not be infinite, thus, the outer loop terminates. We can now conclude that our algorithm will always terminate.□

We note that Algorithm 8 is not guaranteed to give a Nash equilibrium. Such an equilibrium may simply not exist, but even if it exists, the algorithm is not guaranteed to find it. Due to the end condition of its outer loop, Algorithm 8, while still based on agents' self-interest, is more centered on attaining general optimality (maximizing the sum of all agents utilities) than Algorithm 7.

An important thing that differentiates our algorithm from generic approaches to solve hedonic

games presented in [AS16] is the inner loop of our algorithm, which allows groups to grow to a point that is better for every agent even in situations where individual rational decisions could not.

Let's examine this simple example : We have four agents, A, B, C and D. $n(x) = \frac{1}{1+x}$, $v(x) = x$, p's values in $\mathbb{R}^2$, $p(A) = [0, 0]$, $p(B) = [0, 2.5]$, $p(C) = [2.5, 0]$ and $p(D) = [2.5, 2.5]$ (a square).

In this example, having all agents in a single group, $\mathcal{G}$, is the best solution. $\forall_{a\in\{A,B,C,D\}} U_p(a, \mathcal{G}) = \frac{4}{1+2.5\sqrt{2}/2} > 1$. But this situation could not be reached from groups consisting of a single agent by individual rational decisions, since the best interest one single agent could get from grouping with another agent would be $\frac{2}{2.25} < 1$, so no agent would want to group with any other. With our algorithm, if we start with a group $\mathcal{G} = \{A\}$, due to making computations based on the potential maximum size of the group rather than its actual size, individual interest of B and C for join the group would be estimated at $\frac{4}{3.5} > 1$ with an atomic variant (even more with a non-atomic variant) and the group $\mathcal{G}$ will be updated $\mathcal{G} = \{A, B, C\}$. On the second run of the inner loop, with the bary center of $\mathcal{G}$ being at $[2.5/3, 2.5/3]$, the atomic-estimated interest for D to join $\mathcal{G}$ would be $\frac{4}{1+5/3\sqrt{2}} > 1$. We end up with $\mathcal{G} = \{A, B, C, D\}$.

IV Experimentation

In this section, we evaluate Algorithm 8, in normal and atomic agents variants and see how much, in practice, it deviates from Algorithm 7's Nash equilibrium property. We compare our algorithm with two reference clustering algorithms: OPTICS [Ank+99] and k-means [Llo82] with k-means++ [AV07] initialization. All algorithms evaluated have guaranteed termination. We perform this evaluation on four kinds of datasets: synthetic data and two real data applications: leisure travel and wholesale purchases.[3]

IV.1 Metrics

We rely on the notion of *loss* of each agent for a given cluster affectation. Intuitively, the loss of an agent in a given configuration is the difference between the utility of the agent in this configuration and the maximum utility the agent could have if the agent deviates from this configuration (leave and/or join clusters) without any other agent moving. Said otherwise, the *loss* of an agent a for a given cluster affectation $\mathcal{C}$ is defined as the difference ($\geqslant 0$) between the utility the agent gets in this affectation and the maximum utility the agent could get in another affectation that only differs from $\mathcal{C}$ by the position of a in the affectation (a can leave its group and potentially join another one but no other agent can move). For example, if in a situation, an agent a is affected to the cluster $\{a, b, c\}$ and gets a utility of 3 but could get a maximum (over all configurations possible without changing the position of any agent other than a) utility of 4 if in cluster $\{a, d, e\}$, then its loss is 1. The targeted value is 0.

We use the following three metrics to measure the performance of each algorithm:

- The losses of agents (summed among all agents).
- The share of agents having losses (loss > 0).
- The global utility of agents (sum of all utilities).

A good group formation algorithm (according to our definition of this problem) should deliver a close-to-maximum global utility, along with a low sum of agents' losses, and a low rate of losing agents (in the ideal case these last two metrics should have value 0).

In addition to the above metrics, we also display the results of example runs (in the case of random synthetic data, using the same data for all algorithms). In these, a color identifies a cluster, gray points

[3] Our code is available at https://gitlab.inria.fr/abouchra/distributed_neural_networks

represent isolated agents (or agents in a cluster of size 1). Results are averaged over 10 runs; error bars indicate the standard deviation.

IV.1.a Clusters visualization

While our synthetic data is bidimensional and thus easily representable in a figure, the real data we use have a larger dimension (6 for both datasets), thus, we provide views in reduced dimensions, 2 and 3. We propose two different kinds of dimension reduction methods. The first simply presents the dimensions were the original dataset has higher standard deviation. See Figure 4.20 for 2D and Figure 4.21 for 3D. The second uses the *scikit-learn*[Ped+11] Python library, which implements various algorithms that can perform non-linear dimension reduction (Manifold). The specific algorithm we used is MDS (Multi-Dimensional Scaling)[BG97].

IV.2 Synthetic Data

We first evaluate our algorithm on $\mathbb{R}^2$ data generated using a combination of Gaussian distributions. We choose Gaussians because these distributions are usually considered good models of characteristics distribution in real populations. A fixed number of points is generated, the distribution used to generate each vector is chosen randomly according to a predetermined ratio. Parameters of these laws will be detailed in each specific tests. We used the usual Euclidean distance for d, $n(x) = \frac{1}{1+x}$ and $v(x) = \sqrt{x}$[4]. $prc = 20$ (number of trials (execution of the clustering algorithm itself) with the same parameters), $mmt = 5$ (momentum). k-means' parameter k is searched for in the same way as for AUCCCR. For OPTICS, which is not randomized and, as such, does not suffer from the same instability (as k-means and our algorithm) in its output, the procedure used to select the proper minimum reachability distance (points closer than this distance are considered close enough to be part of the same cluster) is equivalent but with $prc = mmt = 1$. For OPTICS-specifics parameters, we use $\epsilon = \infty$ (maximum distance to consider, mostly a speedup parameter so ∞ is an acceptable (official) default), $MinPts = 1$ (number of points of a cluster that should be close enough for a certain point to be considered part of this cluster, we considered 1 to be an acceptable default).

Results are average values over 10 runs (execution of the whole recommendation algorithm), error bars indicate standard deviation. Example results, based on a single run (same data for all algorithms), are also given. In these, a color identify a cluster, gray points are alone (or in a cluster of size 1). "AUCCCR-C" is Algorithm 8 in its normal variant, "AUCCCR-CA" is the atomic variant.

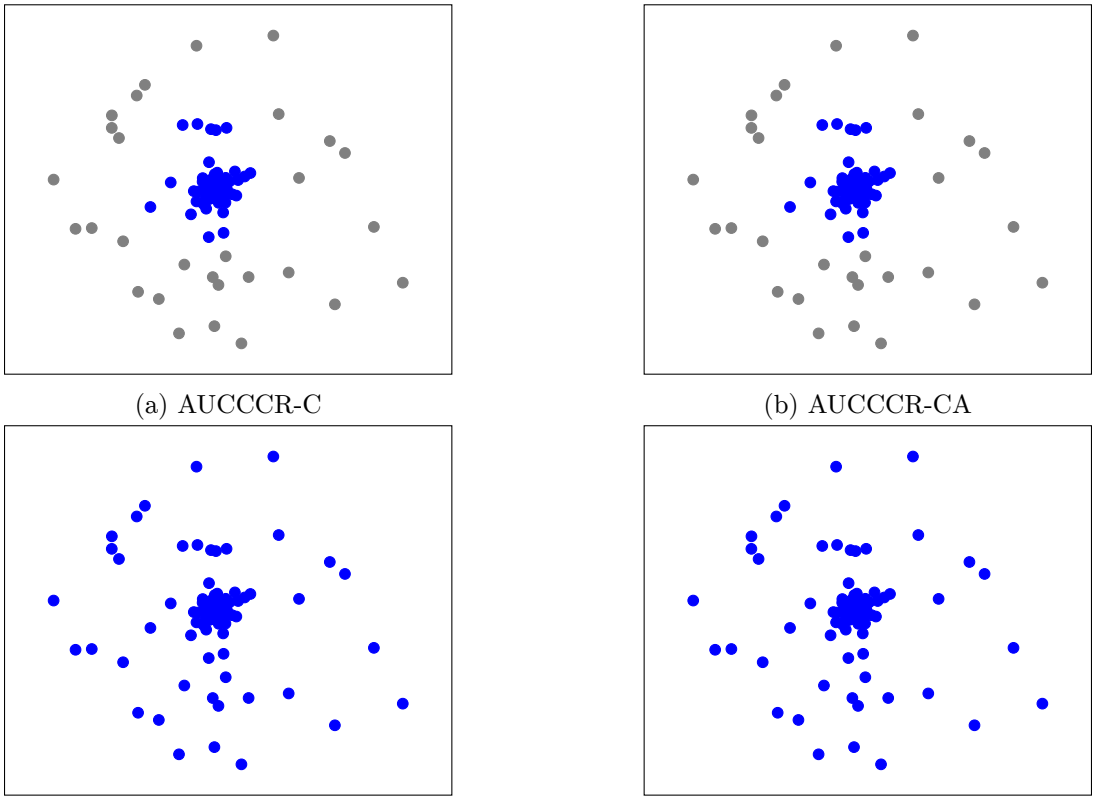
IV.2.a Bi-Gaussian

For this test, we used two $(0, 0)$-centered Gaussian distributions with 1 and 8 as standard deviations and a $0.5 - 0.5$ ratio (same probability for each distribution). The number of point is 100.

Figure 4.1 presents a sample output of the four algorithms. In these, a color identify a cluster, gray points are alone (or in a cluster of size 1). Figure 4.2c shows the global utility, as a sum of all agents' utility, for each algorithm. Figure 4.2a shows the sum of the losses of agents, which is the difference between the utility an agent gets from the present cluster affectation and the maximum utility the agent could get if the agent chooses its cluster (or being alone) following its own interest (given by Formula 4.1). Figure 4.2b shows the share of agents experiencing losses.

In this test, we see that AUCCCR-C(A) yields a lower global utility but also that its output is very close to a Nash equilibrium (near 0 losses), while k-means and OPTICS have more than 25% of agents experiencing losses. Looking at individual runs, we see that k-mean and OPTICS just produce a single giant cluster, maximizing its value for all agent but causing losses for agents that lie far away from the barycenter, while our algorithm, because it allows agent to stay on their own, took more care of their individual interest.

[4]We chose these functions because they are simple functions that have the right shape; we did not optimize this choice.



(a) AUCCCR-C (b) AUCCCR-CA

(c) k-means (d) OPTICS

Figure 4.1: Clusters found in Bi-Gaussian (colors are clusters, grey points are alone)

IV.2.b 3 Gaussians

For this test, we used three Gaussian distributions, $(-6, 0)$, $(0, 0)$ and $(0, 6)$-centered with 1.5 as standard deviations and a $0.33-0.33-0.33$ ratio (same probability for each distribution). The number of point is 200.

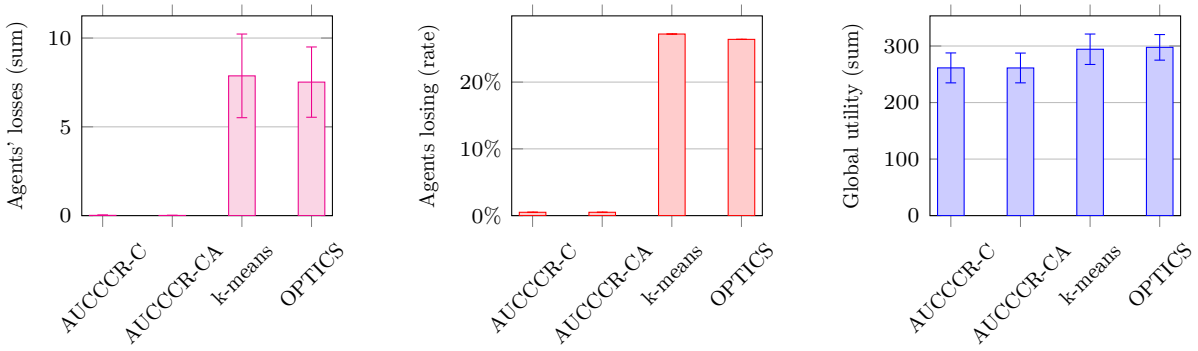
Figure 4.3 presents a sample output of the four algorithms. In these, a color identify a cluster, gray points are alone (or in a cluster of size 1). Figure 4.4c shows the global utility, as a sum of all agents' utility, for each algorithm. Figure 4.4a shows the sum of the losses of agents, which is the difference between the utility an agent gets from the present cluster affectation and the maximum utility the agent could get if the agent chooses its cluster (or being alone) following its own interest. Figure 4.4b shows the share of agents experiencing losses.

In this test, we see that AUCCCR-C(A) and k-means both perform well it terms of global utility, better than OPTICS, likely due to OPTICS density-centered design being ineffective for this kind of pattern ; OPTICS has difficulties in distinguishing close but clearly distinct distributions. Losses were limited but AUCCCR clearly outperformed k-mean and, even more, OPTICS.

IV.2.c Gaussians Star

For this test, we used five Gaussian distributions, $(0, 0)$, $(-10, -10)$, $(-10, 10)$, $(10, -10)$ and $(10, 10)$-centered with 5 for the $(0, 0)$ and 2 for others as standard deviations and a $0.5 - 0.125 - 0.125 - 0.125 - 0.125$ ratio. The number of point is 300.

Figure 4.5 presents a sample output of the four algorithms. In these, a color identify a cluster, gray points are alone (or in a cluster of size 1). Figure 4.6c shows the global utility, as a sum of all agents' utility, for each algorithm. Figure 4.6a shows the sum of the losses of agents, which is the difference between the utility an agent gets from the present cluster affectation and the maximum utility the agent could get if the agent chose its cluster (or being alone) following its own interest. Figure 4.6b



(a) Agents' losses (b) Share of agents having losses (c) Global utility achieved

Figure 4.2: Metrics in Bi-Gaussian for all algorithms

shows the share of agents experiencing losses.

In this test, all algorithms gave similar results in terms of global utility (slightly lower for AUCCCR on average). The gap much higher on losses, with AUCCCR clearly leading. Looking at samples, it seems that AUCCCR performed pretty well for identifying the five Gaussians mixed in the input data compared to k-means and OPTICS.

IV.2.d Effect of the number of agents

In this test we reuse our Gaussians Star setup, but we test different numbers of agents.

Figure 4.7c shows the global utility, as an average of all agents' utility, for each algorithm. Figure 4.7a shows the average of the losses of agents, which is the difference between the utility an agent gets from the present cluster affectation and the maximum utility the agent could get if the agent chooses its cluster (or being alone) following its own interest. Figure 4.7b shows the share of agents experiencing losses.

We see here that all algorithms have similar performances in terms of global utility; AUCCCR being slightly better for a high number of agents. In terms of losses, however, OPTICS and k-means perform significantly worse for low numbers of agents.

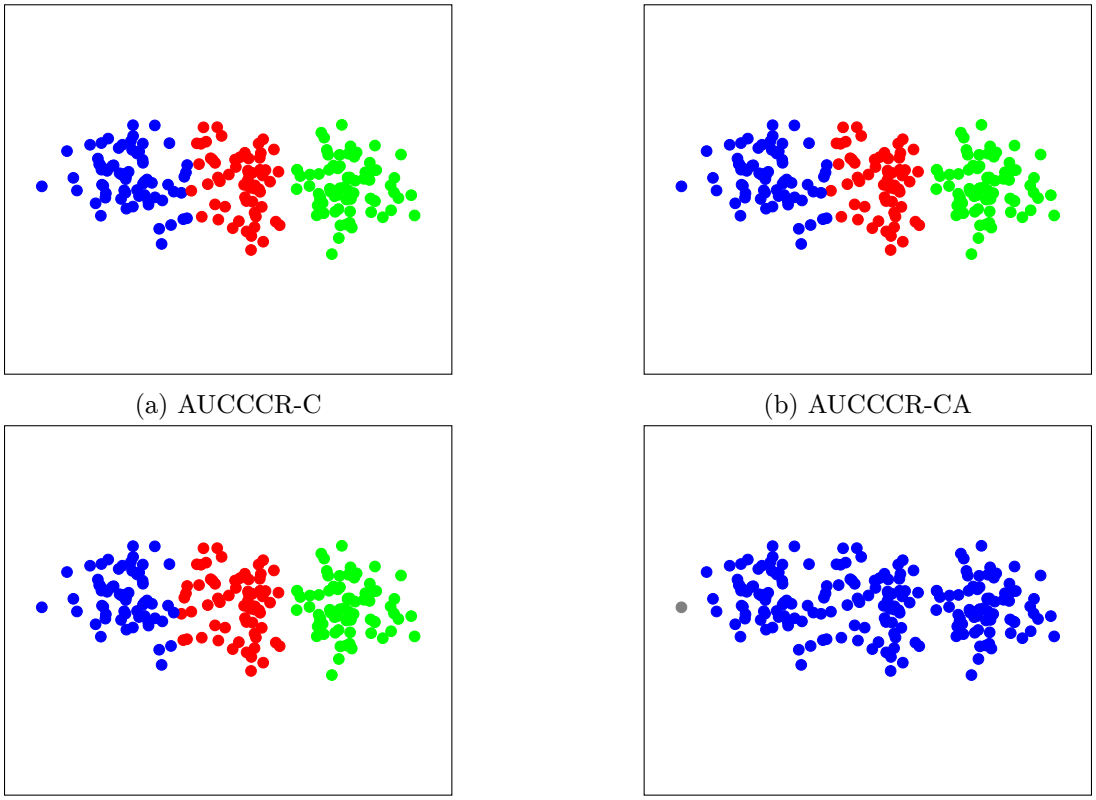
IV.2.e Effect of the standard deviations

In this test we reused our Gaussians Star setup, but we tested different standard deviations.

Figure 4.8c shows the global utility, as an average of all agents' utility, for each algorithm. Figure 4.8a shows the average of the losses of agents, which is the difference between the utility an agent gets from the present cluster affectation and the maximum utility the agent could get if the agent chooses its cluster (or being alone) following its own interest. Figure 4.8b shows the share of agents experiencing losses.

For recall, the Gaussian Star is a mixture of 5 distribution, a $(0,0)$-centered distribution with a certain standard deviation and 4 others, with their centers at a certain distance from $(0,0)$ and a lower (same for the 4) standard deviation. The value used in the "Standard deviations" axis is the $(0,0)$-centered distribution's standard deviations. Others' standard deviations are modified by the same factor, as well as the distances between distributions' centers.

With a high standard deviation, the optimal solution becomes essentially to have nearly as many clusters as agents. k-means performs much worse than other algorithms (nearly all agents have losses, while nearly none have for other algorithms) because it is not designed to find this kind of affectation. AUCCCR and OPTICS have similar performance overall, except for a standard deviation of 10. In losses, AUCCCR is the only algorithm to always remain at very low values. For the value 10, while AUCCCR was slightly lower for global utility, it is by far the best for losses.



(a) AUCCCR-C (b) AUCCCR-CA

(c) k-means (d) OPTICS

Figure 4.3: Clusters found in 3 Gaussians (colors are clusters, grey points are alone)

IV.2.f Effect of scale

In this test we reused our Gaussians Star setup and combined the two previous tests, modifying at the same time, standard deviations, number of agents and mean values separations by the same scale factor (for geometrical reasons, this implies that the distances were increased by the square root of this factor).

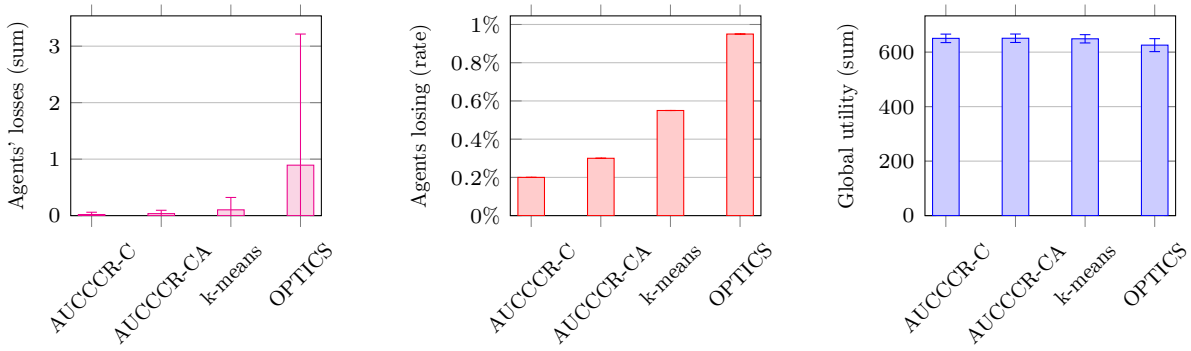
Figure 4.9c shows the global utility, as an average of all agents' utility, for each algorithm. Figure 4.9a shows the average of the losses of agents, which is the difference between the utility an agent gets from the present cluster affectation and the maximum utility the agent could get if the agent chooses its cluster (or being alone) following its own interest. Figure 4.9b shows the share of agents experiencing losses.

Here we observe that AUCCCR can have difficulties with certain scale values, around 2 or 4, in terms of global utility, while being more competitive for a higher value of 8. This is probably due to such scales offering more opportunity for AUCCCR to deviate from the global optimum in favor of reducing individual losses. In terms of losses, however, AUCCCR remains extremely low for all value, while other algorithms get much worse results.

IV.2.g Size constraint

While our algorithm is originally designed for cases where larger clusters are always the best (in terms of agents utility), we show here how it can be modified to accommodate cases where, passed a certain threshold, larger group become not more desirable or even less desirable. To integrate this in our algorithm, we simply need to modify the $v(x)$ (effect of the group size on utility) from a strictly increasing function to a function that reaches a maximum, or even starts decreasing, at some point.

To evaluate the ability of our algorithm to manage cases where very large clusters are not desirable (agents want to be in a sufficiently large group but not too large) we take a simple case, with a single



(a) Agents' losses (b) Share of agents having losses (c) Global utility achieved

Figure 4.4: Metrics in 3 Gaussians for all algorithms

Gaussian, in which our usual $v(x) = \sqrt{x}$ function would make a single cluster (with all agents in it) optimal. We then change $v(x)$ so that too large clusters are not optimal.

We test two different functions, both $= \sqrt{x}$ for $x \leqslant 20$, but for $x > 20$ the first function is constant $\sqrt{20}$ while the second is decreasing $\sqrt{\frac{20}{1+\frac{|20-x|}{20}}}$ (both functions are continuous). The number 20 is an arbitrary choice and can be considered as a target size for clusters. See Figure 4.10 for a plot.

Figure 4.11 presents the clusters obtained with the different algorithms and $v(x)$ functions. You can see the precise size of each cluster in Figures 4.12 ($\sqrt{x}$), 4.13 ($\sqrt{20}$) and 4.14. ($\sqrt{\frac{20}{1+\frac{|20-x|}{20}}}$). Metric are shown in Figures 4.15 ($\sqrt{x}$), 4.16 ($\sqrt{20}$) and 4.17. ($\sqrt{\frac{20}{1+\frac{|20-x|}{20}}}$).

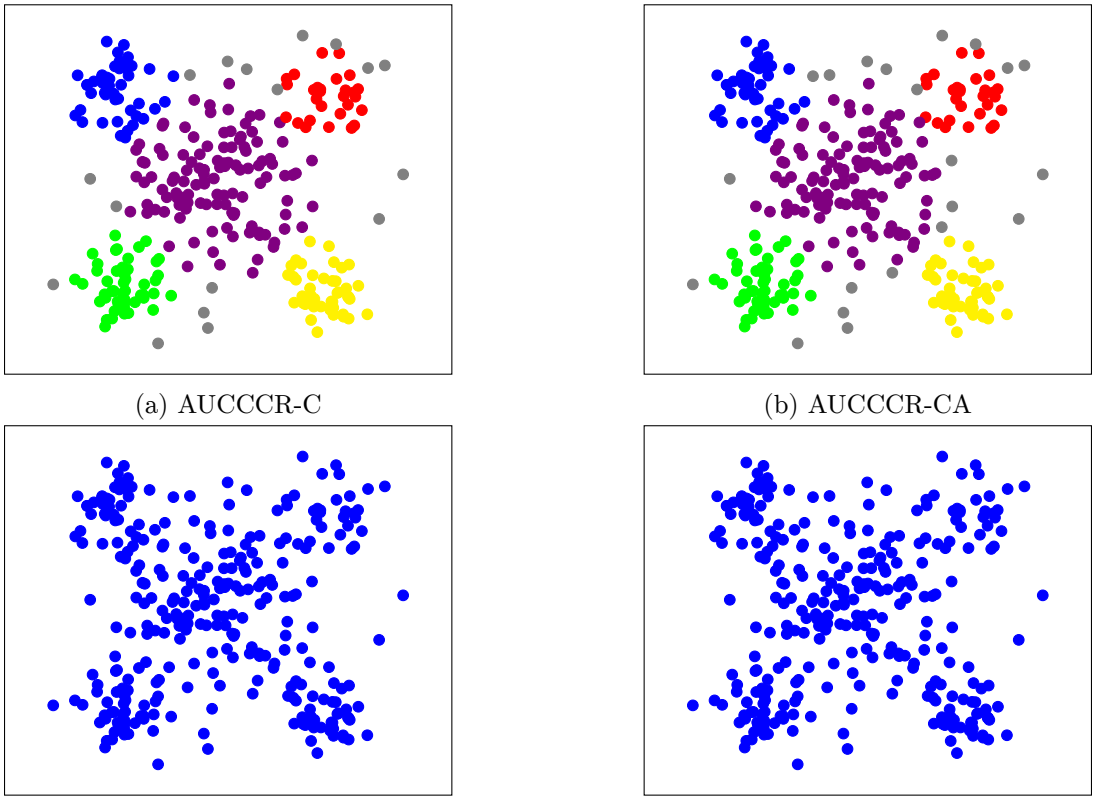
We see that changing the $v(x)$ function is a good way to adapt AUCCCR to size constraints. $\sqrt{20}$ is sufficient as the effect of distance reduces interest for larger (necessarily sparser) groups, even through $\sqrt{\frac{20}{1+\frac{|20-x|}{20}}}$ has a stronger effect. A downside of $\sqrt{\frac{20}{1+\frac{|20-x|}{20}}}$ is that AUCCCR is not designed for decreasing $v(x)$ functions (k-means and OPTICS are even worse, as they are not designed for utility functions at all) and we see that we get (too) small clusters in that case, but this does not prevent the algorithm from working correctly. Still, AUCCCR performs quite well in terms of size of cluster (close to the target size) in general, while k-means and OPTICS produce many smaller clusters with $\sqrt{\frac{20}{1+\frac{|20-x|}{20}}}$. With $\sqrt{20}$, AUCCCR clusters tends to be large, sometimes a bit larger than wanted, compared to k-means and, even more, OPTICS, which produce too small clusters. With the metrics, we see that our algorithm continues to outperform others in the same way as with other experiments with $\sqrt{20}$, with $\sqrt{\frac{20}{1+\frac{|20-x|}{20}}}$ however, k-means' results are very close to AUCCCR's.

IV.3 The BuddyMove dataset

The BuddyMove [RA14] dataset consists of statistics from users of a travel review website. For each user, the dataset provides the number of reviews written for each of 6 classes of destination (e.g. religious sites, parks, etc). From these data, we derived for each user the share of reviews written for each destination class. This can be interpreted as the relative interest of a user for each kind of destination. This could allow providing recommendation to users on whom they should go with for group travels, based on the similarity of their preferences.

IV.3.a Hyperparameters

Our clustering algorithms used Euclidean distance for d, $n(x) = \frac{1}{1+x}$ and $v(x) = \sqrt{x}$. $prc = 20$, $mmt = 5$. Results are average over 10 runs, error bars indicates standard deviation. Sample examples, based on a single run, are also given. In these, a color identify a cluster, gray points are alone (or in a cluster of size 1). "AUCCCR-C" is Algorithm 8 in its normal variant, "AUCCCR-CA" is the atomic



(a) AUCCCR-C (b) AUCCCR-CA

(c) k-means (d) OPTICS

Figure 4.5: Clusters found in Gaussians Star (colors are clusters, grey points are alone)

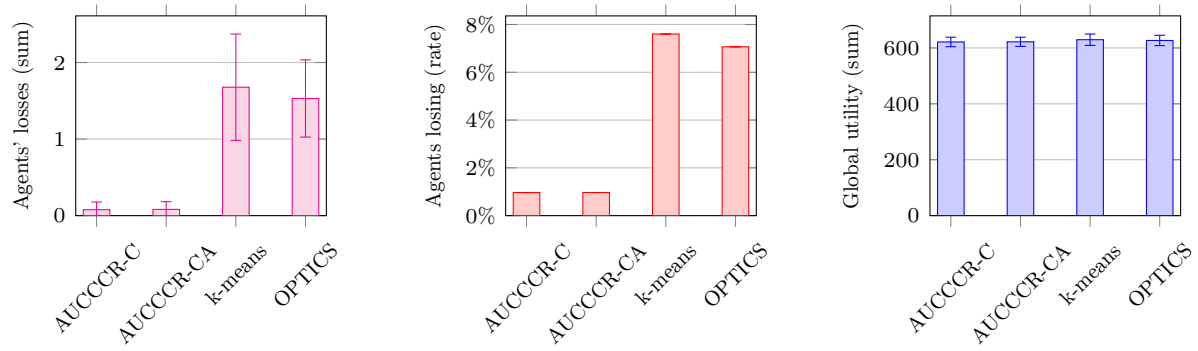
variant.

We introduce an additional hyperparameter when working with real datasets: *scale*. The scale describes how "far" a given distance is considered by the algorithm and can be seen as parameter of the projector function p we presented earlier. The distances naturally present in the dataset must be given an absolute cardinal signification for the algorithm, the right choice is up to the user, we variate it to see how this choice affects the results. For example, if the scale is 10, a distance of 1 will be computed as 10 for computation. As the scale grows, the distance between points will be considered longer by the algorithm.

IV.3.b Metrics

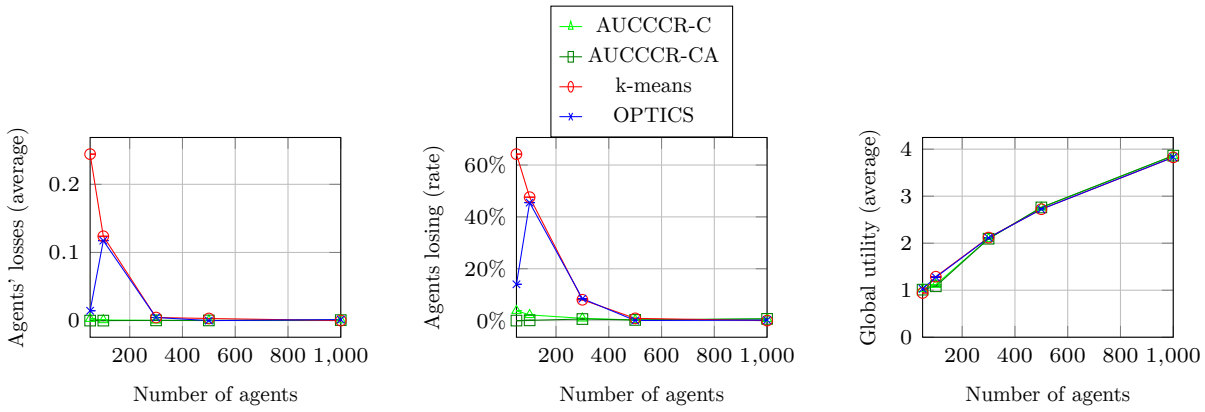
The graphs in Figure 4.18 presents the 3 following metrics : the average utility of agents (Figure 4.18c), the average losses of agents (Figure 4.18a) and the share of agents having losses (Figure 4.18b) for a range of scale values. The range of scale is selected to cover all the range of cases from a situation where everyone alone is optimal to a situation were a universal cluster is optimal.

We note that, is those graphs, the lines for the two variants of AUCCCR are largely superposed, and k-means and OPTICS are identical. For a low scale, users are considered close and all algorithms will detect a single cluster but, as the scale increases, the results become very different. AUCCCR, contrary to the reference algorithms, will prevent individual losses ; as we can see, losses are very low for AUCCCR, especially for large scales, while, for other algorithms, losses increase dramatically. For a scale of 130, AUCCCR has nearly 0 agents losses while k-mean and OPTICS have 30%. This comes at the cost of a reduced global utility, which slowly drops until it nearly hits 1 (equivalent to everyone alone) at 130. For midrange values, we see that AUCCCR has nearly no losses with a still high global utility (1.5, 1.7 for reference algorithms; this lower value being due to the small size of clusters possible with minimal individual losses) while reference algorithms have 15% to 20% of losses.



(a) Agents' losses caused (b) Share of agents having losses (c) Global utility achieved

Figure 4.6: Metrics in Gaussians Star for all algorithms



(a) Agents' losses caused (b) Share of agents having losses (c) Global utility achieved

Figure 4.7: Metrics in Gaussians Star for different numbers of agents by different algorithms

IV.3.c Clusters size

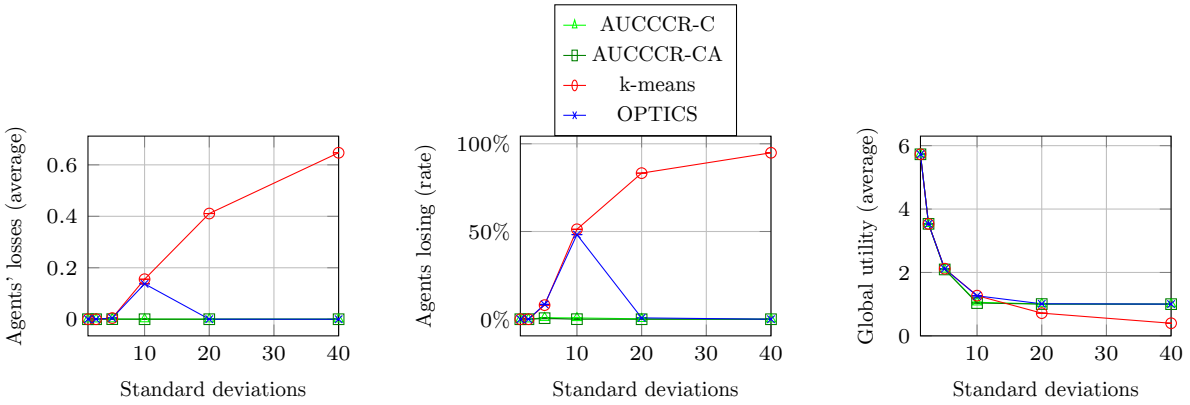
The graphs in Figure 4.19 gives the obtained clusters sizes for the different algorithms with a scale of 115. We see that, for this midrange scale, k-means and OPTICS identified only a universal cluster, while both AUCCCR variants identified 3 and let numerous users alone.

IV.3.d Clusters visualization

We propose visualizations of clusters in reduced dimension (2D and 3D) obtained as described in Section IV.1.a. See Figure 4.20 for 2D and Figure 4.21 for 3D with classical (standard deviation-based) dimension reduction. See Figure 4.22 for 2D and Figure 4.23 for 3D with Manifold MDS visualization. We see that AUCCCR is able to identify clusters in the dataset, unlike k-means and OPTICS.

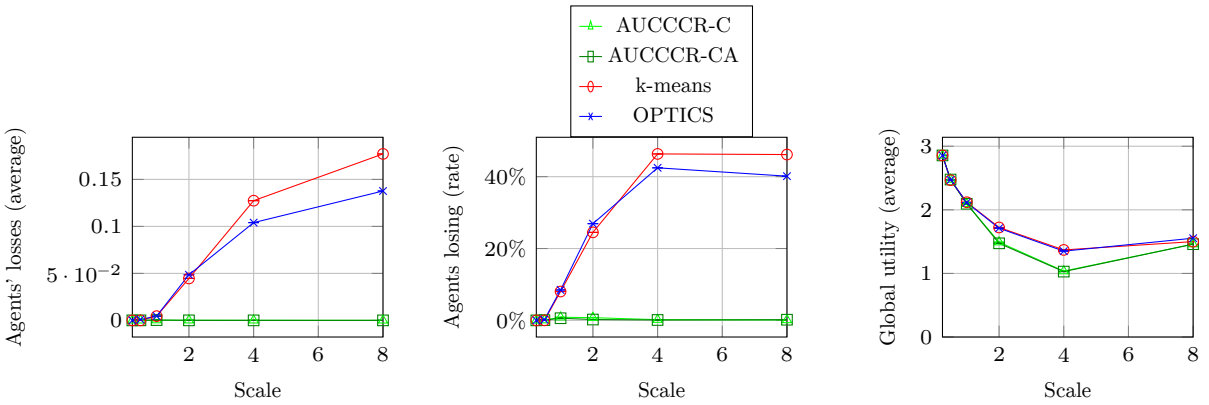
IV.4 The Wholesale dataset

The Wholesale [Fer11] dataset consists of statistics from customers of a wholesale vendor. For each customer, the dataset indicates the annual spending for each of 6 classes of products (e.g. fresh, frozen, etc). From these data, we derive for each customer the share of reviews written for each product class. This result could for instance be used to provide recommendations to customers on whom they should collaborate with to make grouped orders more directly, removing the wholesale distributor from the circuit (short circuit distribution), based on which kind of products they usually order.



(a) Agents' losses caused (b) Share of agents having losses (c) Global utility achieved

Figure 4.8: Metrics in Gaussians Star for different standard deviations by different algorithms



(a) Agents' losses caused (b) Share of agents having losses (c) Global utility achieved

Figure 4.9: Metrics in Gaussians Star for different scales by different algorithms

IV.4.a Hyperparameters

We configure our clustering algorithms the same way as for the previous experiment (most significant elements recalled here, for the rest see Section IV.3.a): Euclidean distance for d, $n(x) = \frac{1}{1+x}$ and $v(x) = \sqrt{x}$. $prc = 20$, $mmt = 5$. "AUCCCR-C" is Algorithm 8 in its normal variant, 'AUCCCR-CA" is the atomic variant.

IV.4.b Metrics

The graphs in Figure 4.24 presents the 3 following metrics : the average utility of agents (Figure 4.24c), the average losses of agents (Figure 4.24a) and the share of agents having losses (Figure 4.24b) for a range of scale values. The range of scale is selected to cover all the range of cases from a situation where everyone alone is optimal to a situation were a universal cluster is optimal.

We note that, is those graphs, the lines for the two variants of AUCCCR are largely superposed. For a low scale, users are considered close and all algorithms will detect a single cluster but, as the scale increases, the results become very different. AUCCCR, contrary to the reference algorithms, will prevent individual losses ; as we can see, losses are close to 0 for AUCCCR, especially for large scales, while, for other algorithms, losses increase dramatically. For a scale of 60, AUCCCR has nearly 0 agents losses while k-mean has $> 30\%$ and OPTICS has $> 40\%$. The global utility remains similar to k-means and greater than OPTICS for the lower half of scales, only higher values have a utility lower than OPTICS but still very close from both, OPTICS and k-means. For midrange values, we see that AUCCCR has nearly no losses with a global utility close to k-means and higher than OPTICS, while

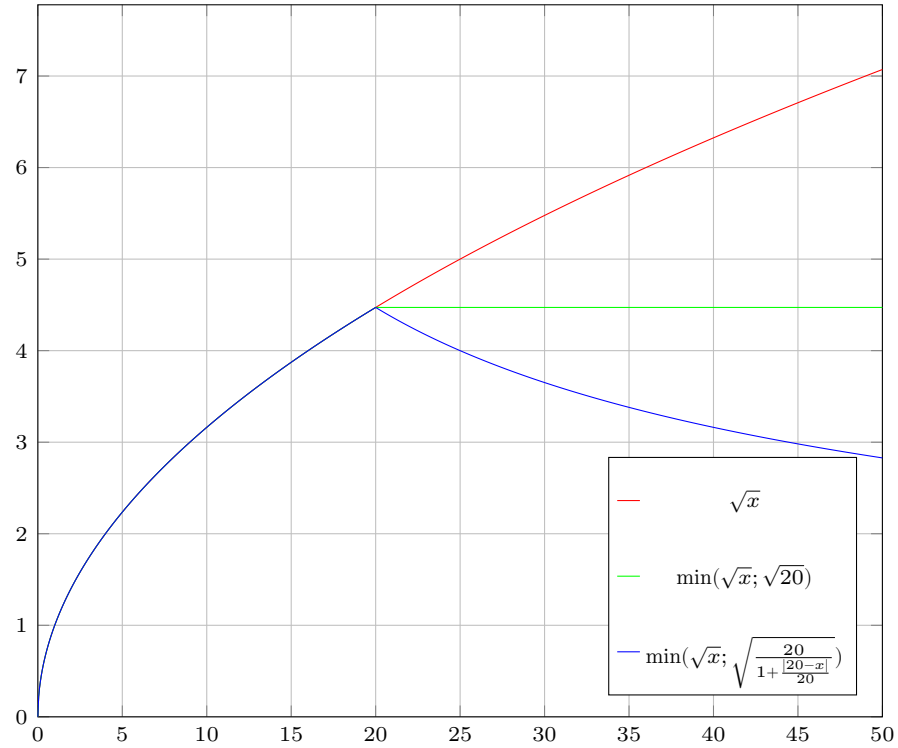


Figure 4.10: Group value functions

reference algorithms have around 10% of losses.

IV.4.c Clusters size

The graphs in Figure 4.25 gives the obtained clusters sizes for the different algorithms with a scale of 45. We see that, for this midrange scale, k-means and OPTICS identified only a universal cluster, while both AUCCCR variants identified 3 and let numerous users alone.

IV.4.d Clusters visualization

We propose visualizations of clusters in reduced dimension (2D and 3D) obtained as described in Section IV.1.a. See Figure 4.26 for 2D and Figure 4.27 for 3D with classical (standard deviation-based) dimension reduction. See Figure 4.28 for 2D and Figure 4.29 for 3D with Manifold MDS visualization. We see that AUCCCR is able to identify clusters in the dataset, unlike OPTICS, and also to leave to some points alone, unlike both, k-means and OPTICS.

Since the dataset has a large dimension (6), we provide views in reduced dimensions, 2 and 3. We propose two different kinds of dimension reduction methods. The first simply presents the dimensions were the original dataset has higher standard deviation. See Figure 4.26 for 2D and Figure 4.27 for 3D. The second uses the *scikit-learn*[Ped+11] Python library, which implements various algorithms that can perform non-linear dimension reduction. The specific algorithm we used is MDS (Multi-Dimensional Scaling)[BG97]. See Figure 4.28 for 2D and Figure 4.29 for 3D.

IV.5 General analysis of results

On our synthetic data tests, we showed that our algorithm is more efficient for cluster identification than classical clustering algorithms. It also appeared that, as intended, our algorithm causes much lower individual losses to agent than other algorithms. While it is not possible to completely remove losses in the absence of a Nash (or similar) equilibrium for mathematical reasons, our algorithm manages to keep such losses low, reducing the risk of agent unsatisfaction and collapse.

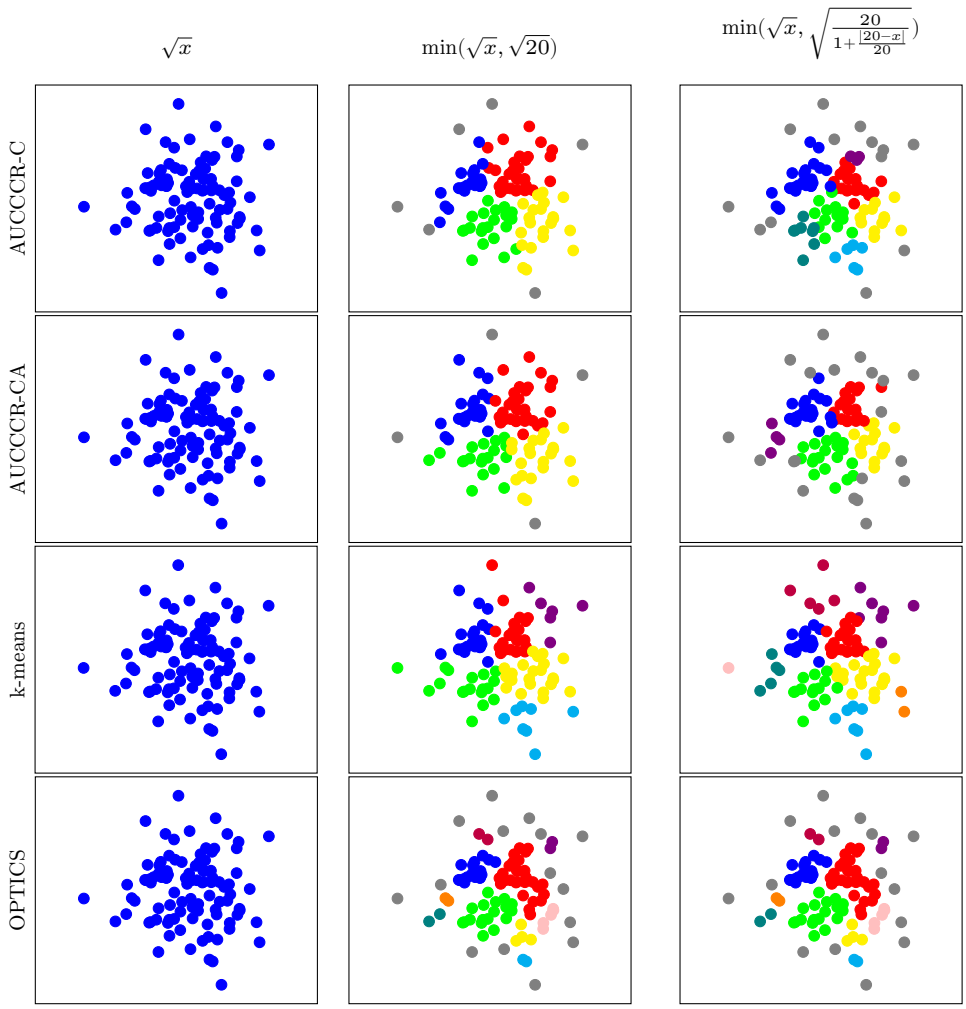


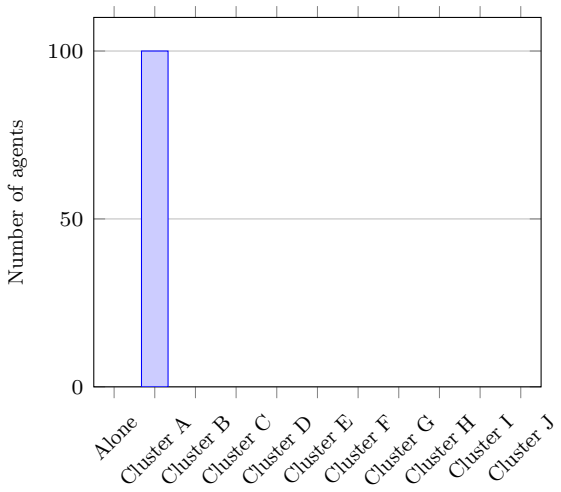
Figure 4.11: Clusters with various algorithms and different $v(x)$

Our algorithm also exhibited good performance on the two real datasets, BuddyMove and Wholesale, proving its applicability to real use cases.

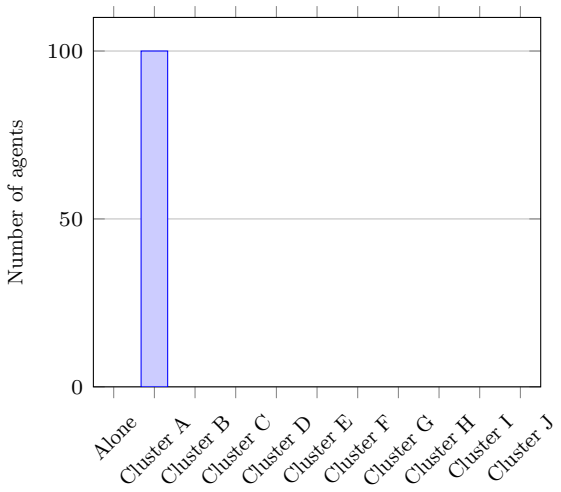
V Conclusion

In this chapter, we defined a new class of hedonic games on which we based a algorithm for providing cooperation recommendation. We compared our algorithm to classical clustering algorithm that could also have been used for the same recommendation problem. We proved that our algorithm yields results that exhibit close to no losses and are thus similar to a Nash equilibrium (which does not exist in the general case) than other algorithms.

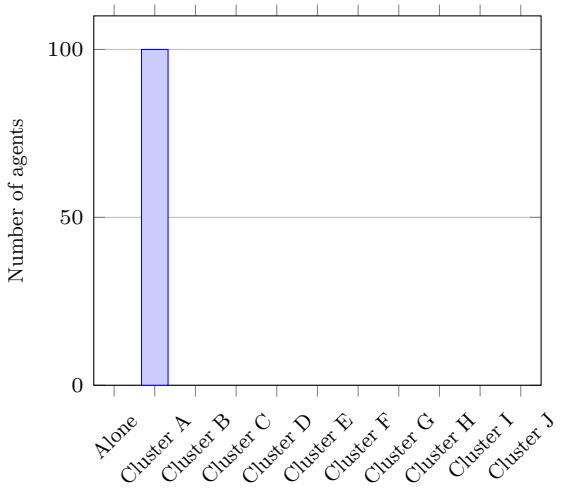
A possible way this work could be extended would be to adapt it to hierarchical clustering [Rok05]. Hierarchical clustering, unlike classical clustering, can produce a complete hierarchy of clusters, subclusters, etc.



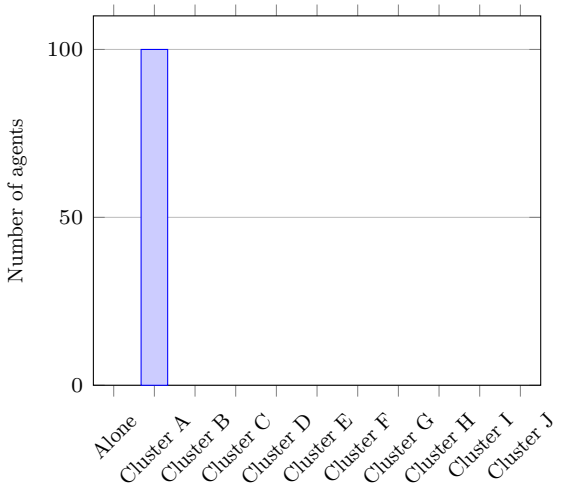
(a) AUCCCR-C



(b) AUCCCR-C



(c) k-means



(d) OPTICS

Figure 4.12: Clusters size with $v(x) = \sqrt{x}$ by different algorithms

Algorithm 8: Clustering algorithm with guaranteed convergence

function CLUSTERCONV(A, p, k) **is**

`/* k-means++ initialization */`

$ra \leftarrow rand(A)$

$ngrp[0] \leftarrow \{ra\}$

foreach $a \in A$ **do**

$dmin2[a] \leftarrow d(p(a), p(ra))^2$

for $1 \leqslant i < k$ **do**

$na \leftarrow rand(A, dmin2)$

`/* random element from` A`, relatives probabilities given by` $dmin2$ `*/`

$ngrp[i] \leftarrow \{na\}$ `// appends` $\{na\}$ `to ngrp`

foreach $a \in A$ **do**

$dmin2[a] \leftarrow \min(d(p(a), p(na))^2, dmin2[a])$

$nval \leftarrow \#A$

`/* main loop */`

repeat

$val \leftarrow nval$

$grp \leftarrow ngrp$

$ngrp.clear()$

$nsize \leftarrow [\#A...\#A]$ `// vector length =` k

repeat

$size \leftarrow nsize$

$nsize \leftarrow [0...0]$

foreach $a \in A$ **do**

$bgrp[a] \leftarrow \text{argmax}_{i \in \bot \cup [\![0,k-1]\!]}(n\Big(d(p(a), bary_p(grp[i] \cup a))\Big) \times v(size[i] + \mathbb{1}_{a \notin grp[i]}))$

`/* assuming` $grp[\bot] = \emptyset$ `and` $size[\bot] = 0$ `*/`

$nsize[bgrp[a]] + +$ `// does nothing if` $bgrp[a] = \bot$

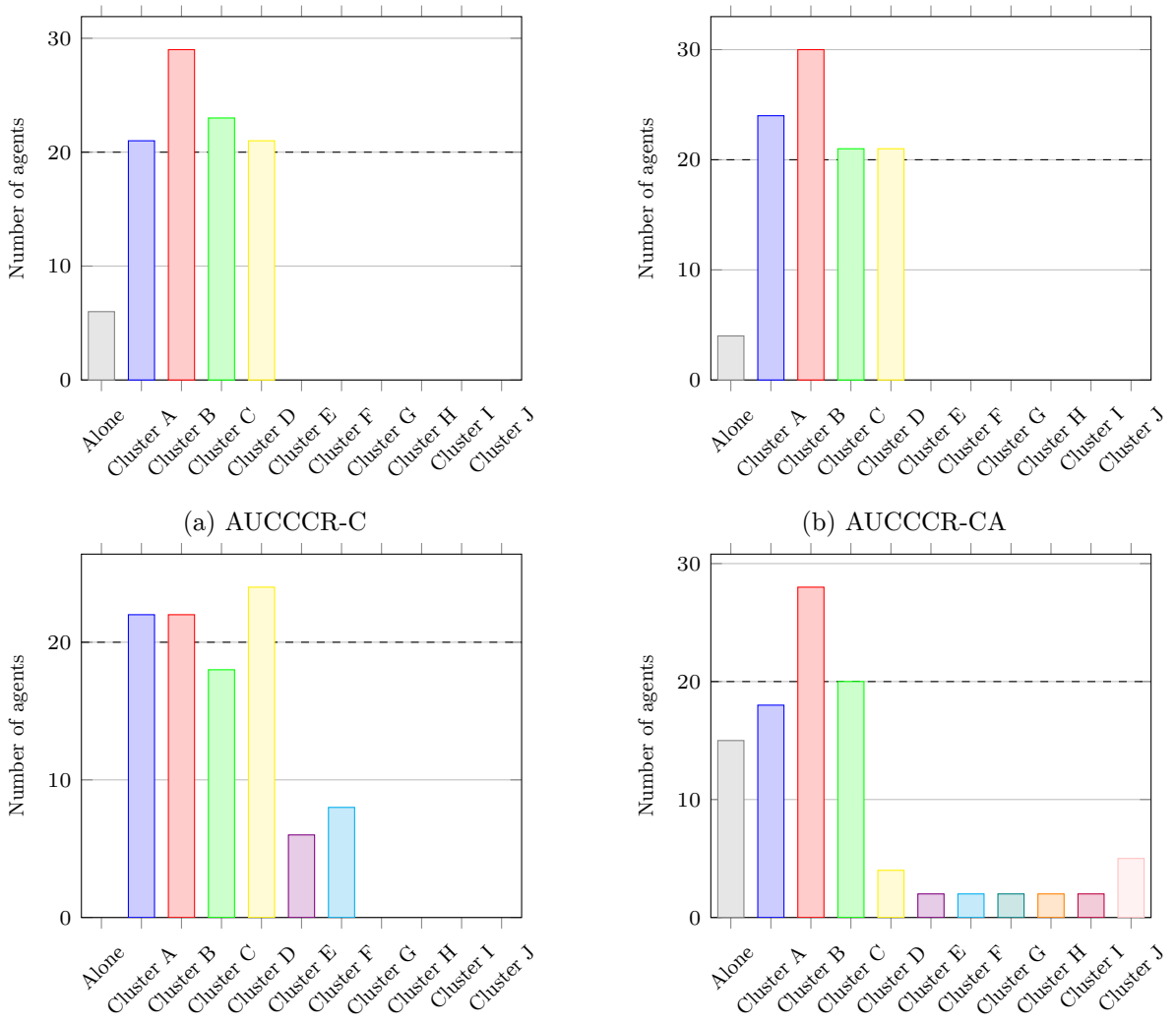
until $\sum nsize \geqslant \sum size$

$ngrp \leftarrow bgrp.invertkv()$ `// key-value inversion`

$nval \leftarrow \sum_{a \in A} n\Big(d(p(a), bary_p(ngrp.group(a)))\Big) \times v(\#ngrp.group(a))$

until $nval \leqslant val$

return grp



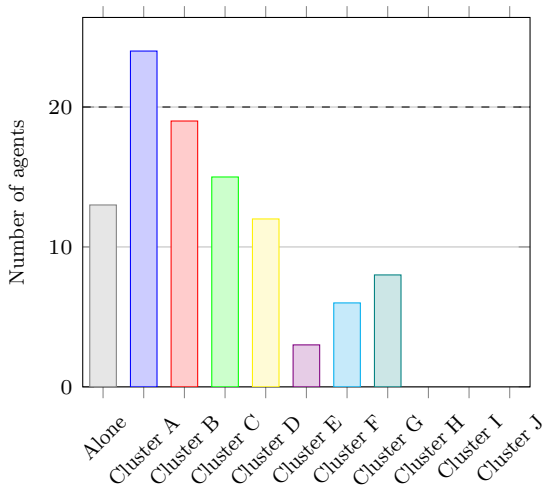
(a) AUCCCR-C

(b) AUCCCR-CA

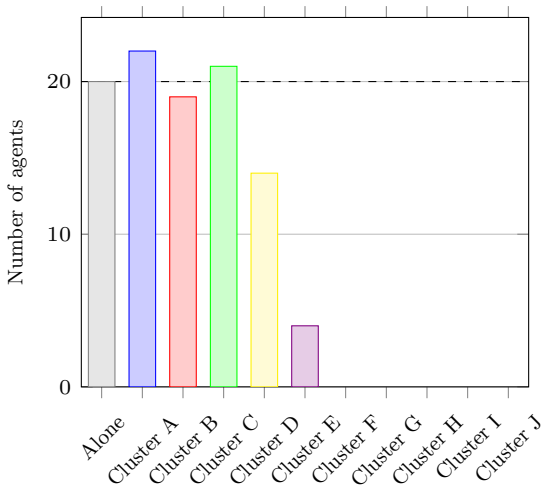
(c) k-means

(d) OPTICS

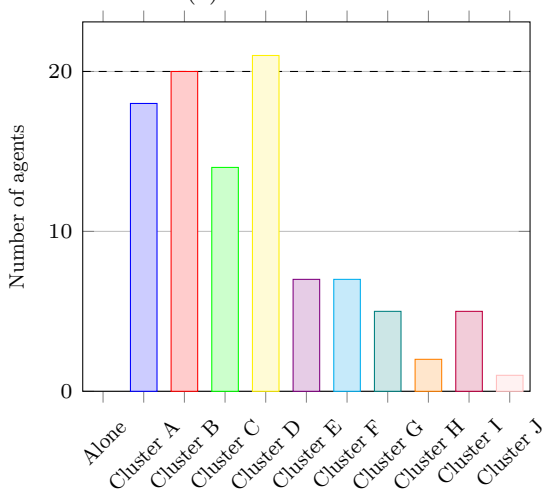
Figure 4.13: Clusters size with $v(x) = \min(\sqrt{x}, \sqrt{20})$ by different algorithms



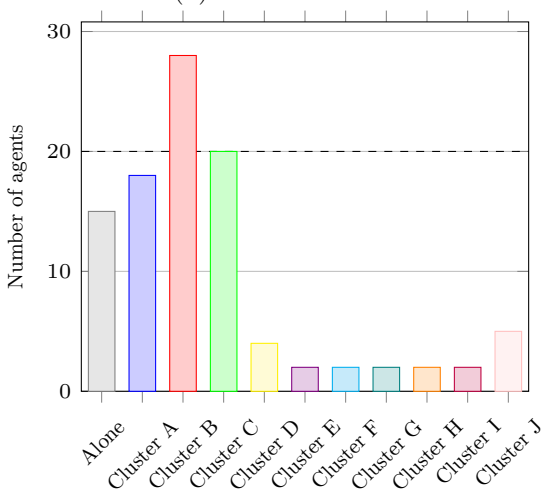
(a) AUCCCR-C



(b) AUCCCR-CA

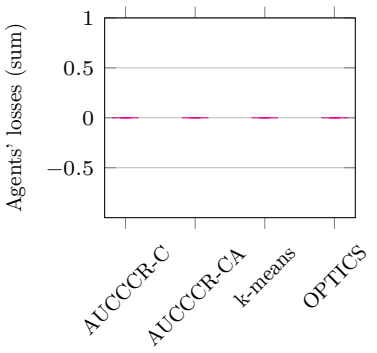


(c) k-means

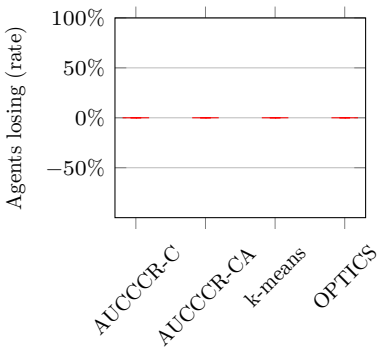


(d) OPTICS

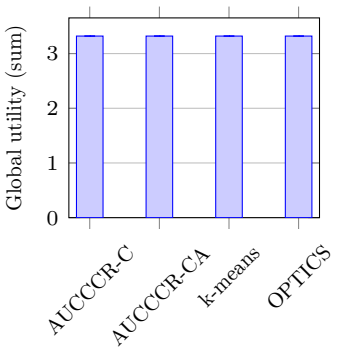
Figure 4.14: Clusters size with $v(x) = \min(\sqrt{x}, \sqrt{\frac{20}{1+\frac{|20-x|}{20}}})$ by different algorithms



(a) Agents' losses

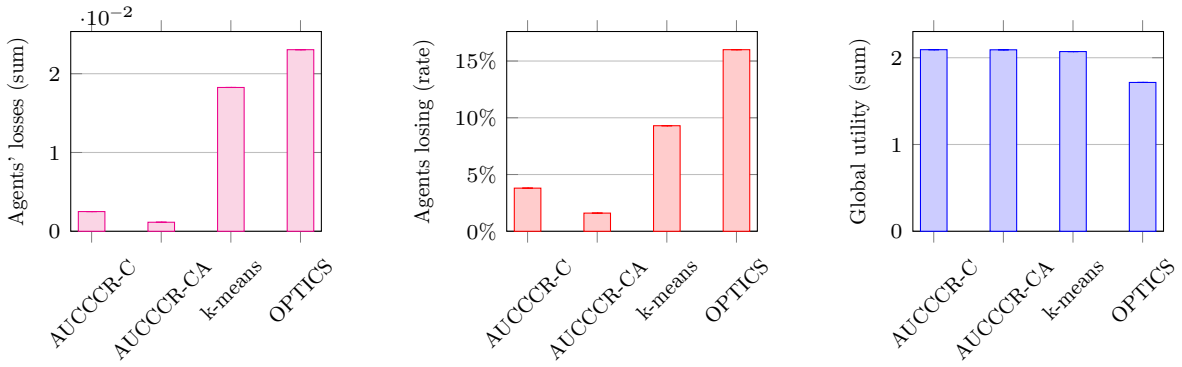


(b) Share of agents having losses



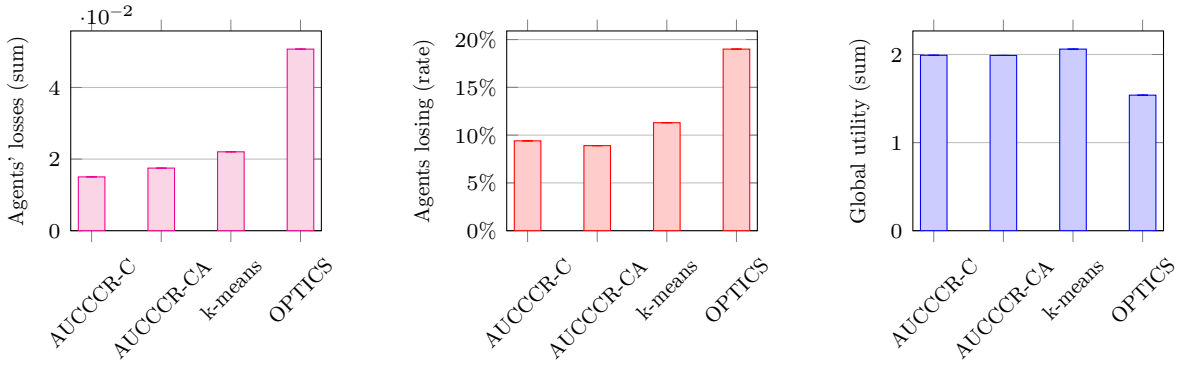
(c) Global utility achieved

Figure 4.15: Metrics with $v(x) = \sqrt{x}$ for all algorithms



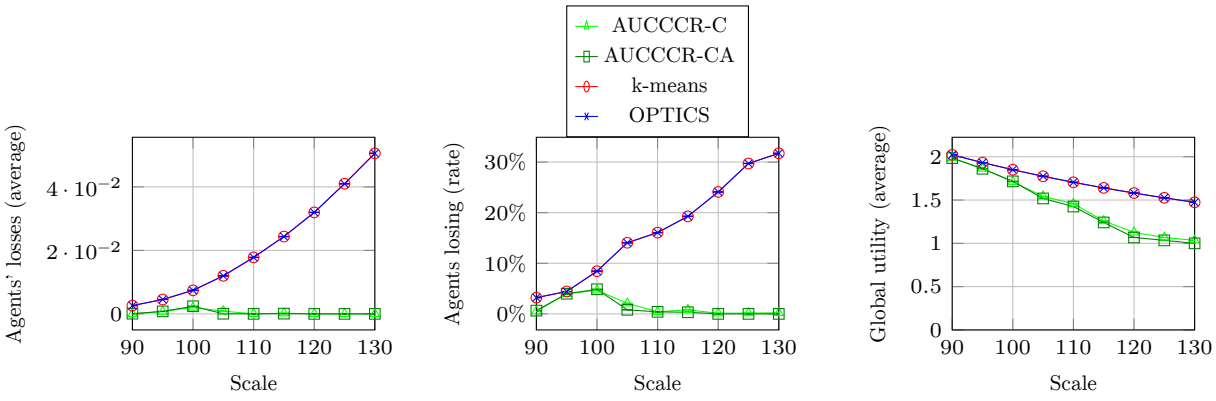
(a) Agents' losses (b) Share of agents having losses (c) Global utility achieved

Figure 4.16: Metrics with $v(x) = \min(\sqrt{x}, \sqrt{20})$ for all algorithms



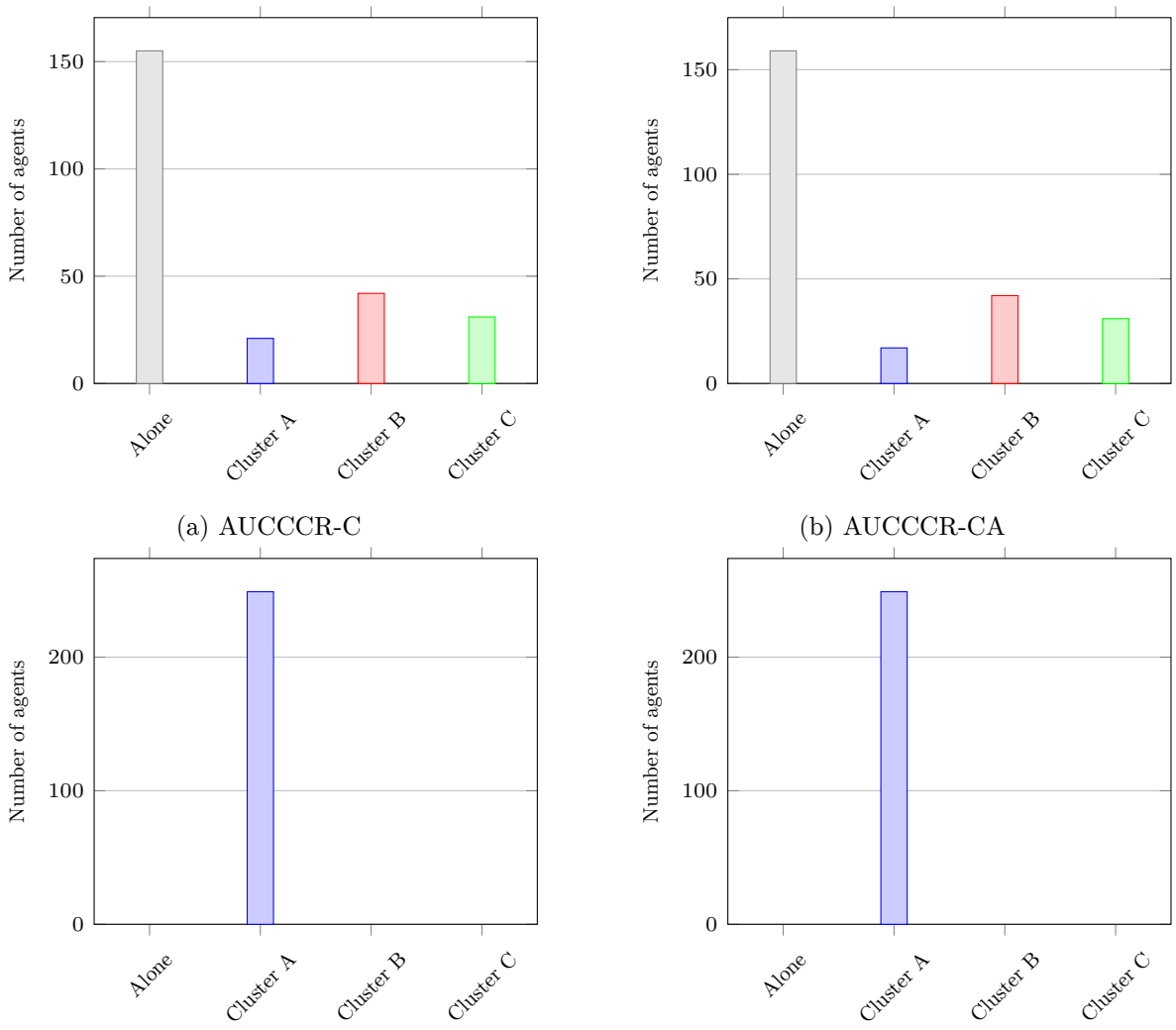
(a) Agents' losses (b) Share of agents having losses (c) Global utility achieved

Figure 4.17: Metrics with $v(x) = \min(\sqrt{x}, \sqrt{\frac{20}{1+\frac{|20-x|}{20}}})$ for all algorithms



(a) Agents' losses caused (b) Share of agents having losses (c) Global utility achieved

Figure 4.18: Metrics in BuddyMove for different scales by different algorithms



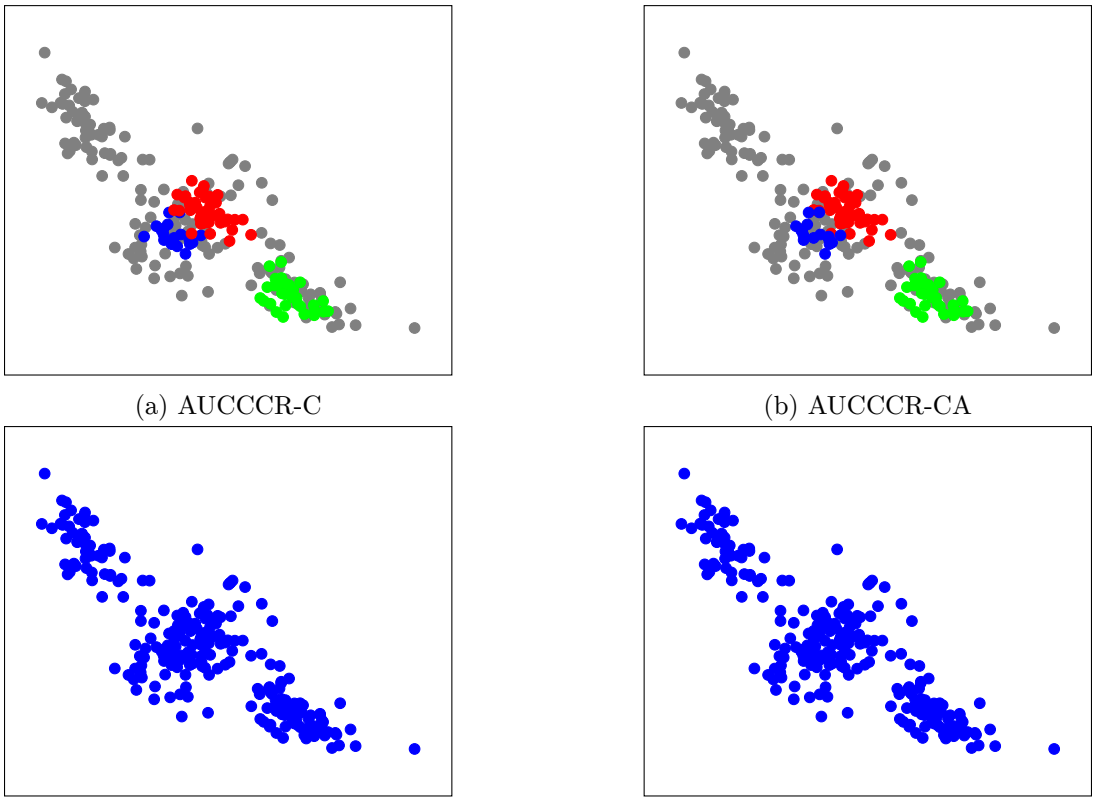
(a) AUCCCR-C

(b) AUCCCR-CA

(c) k-means

(d) OPTICS

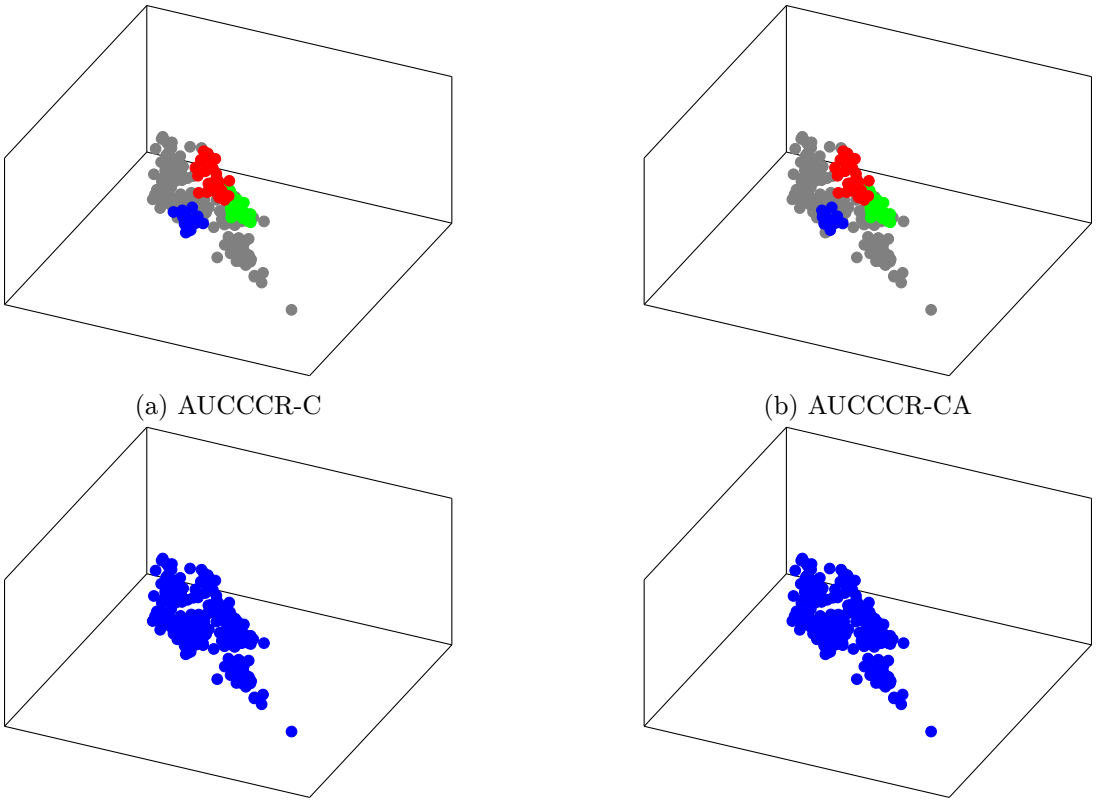
Figure 4.19: Clusters size in BuddyMove for scale 115 by different algorithms



(a) AUCCCR-C (b) AUCCCR-CA

(c) k-means (d) OPTICS

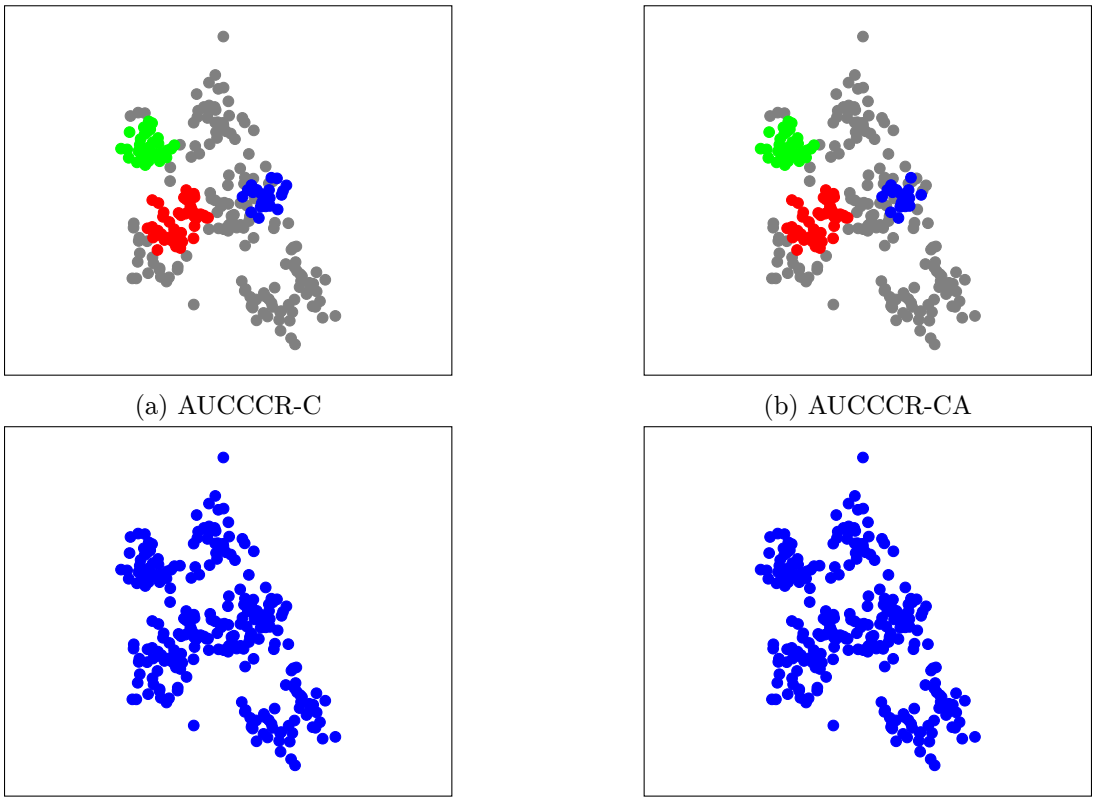
Figure 4.20: Clusters found in BuddyMove@115 (colors are clusters, grey points are alone)



(a) AUCCCR-C (b) AUCCCR-CA

(c) k-means (d) OPTICS

Figure 4.21: Clusters found in BuddyMove@115 (colors are clusters, grey points are alone)



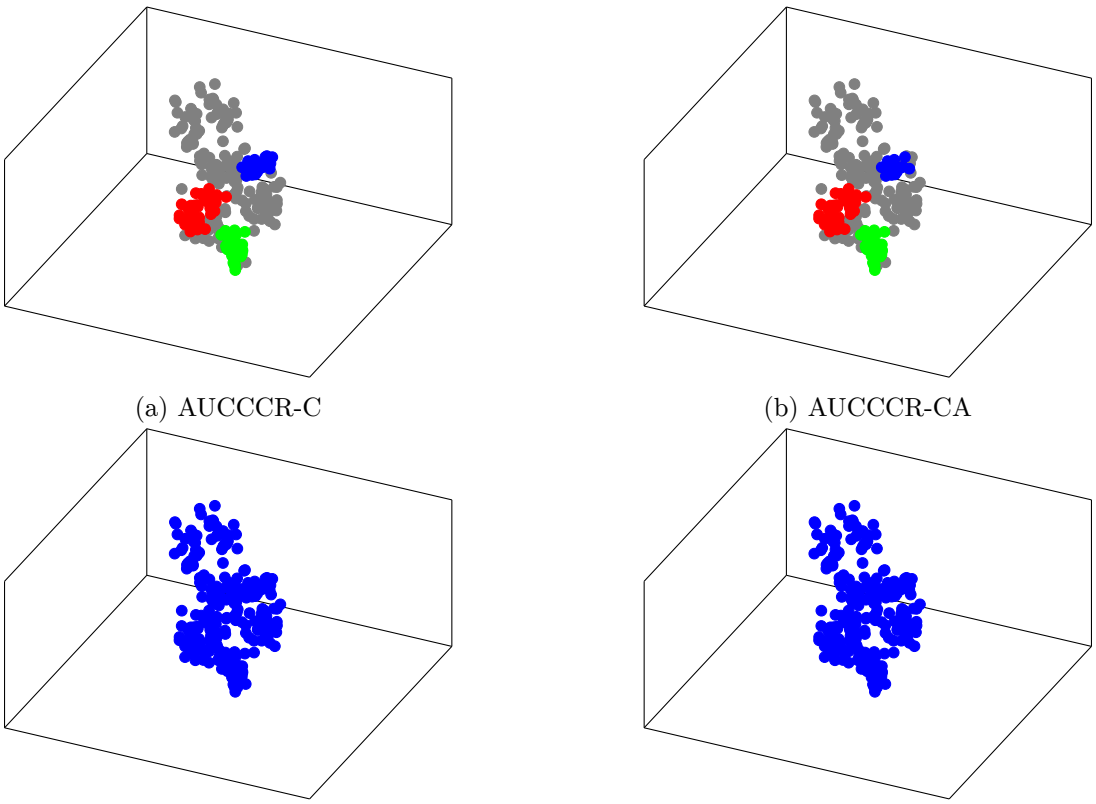
(a) AUCCCR-C

(b) AUCCCR-CA

(c) k-means

(d) OPTICS

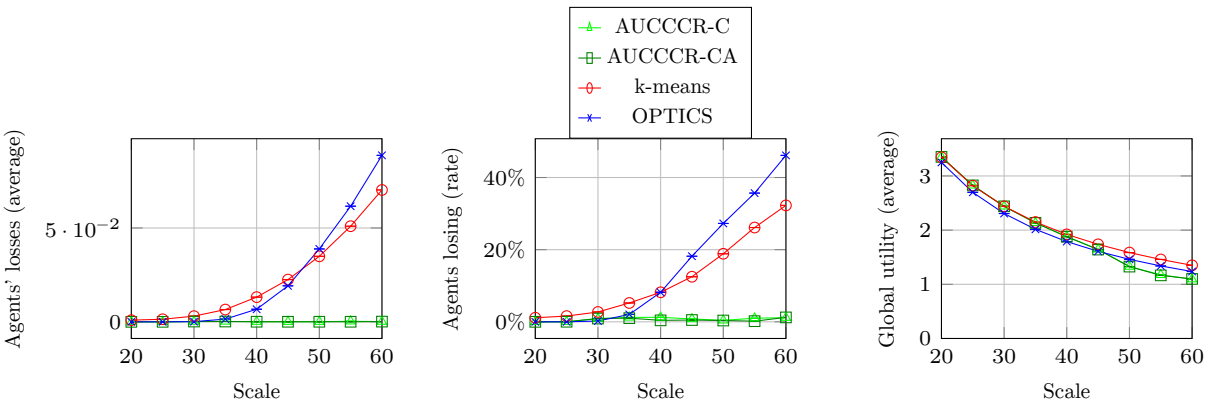
Figure 4.22: Clusters found in BuddyMove@115 (Manifold MDS; colors are clusters, grey points are alone)



(a) AUCCCR-C (b) AUCCCR-CA

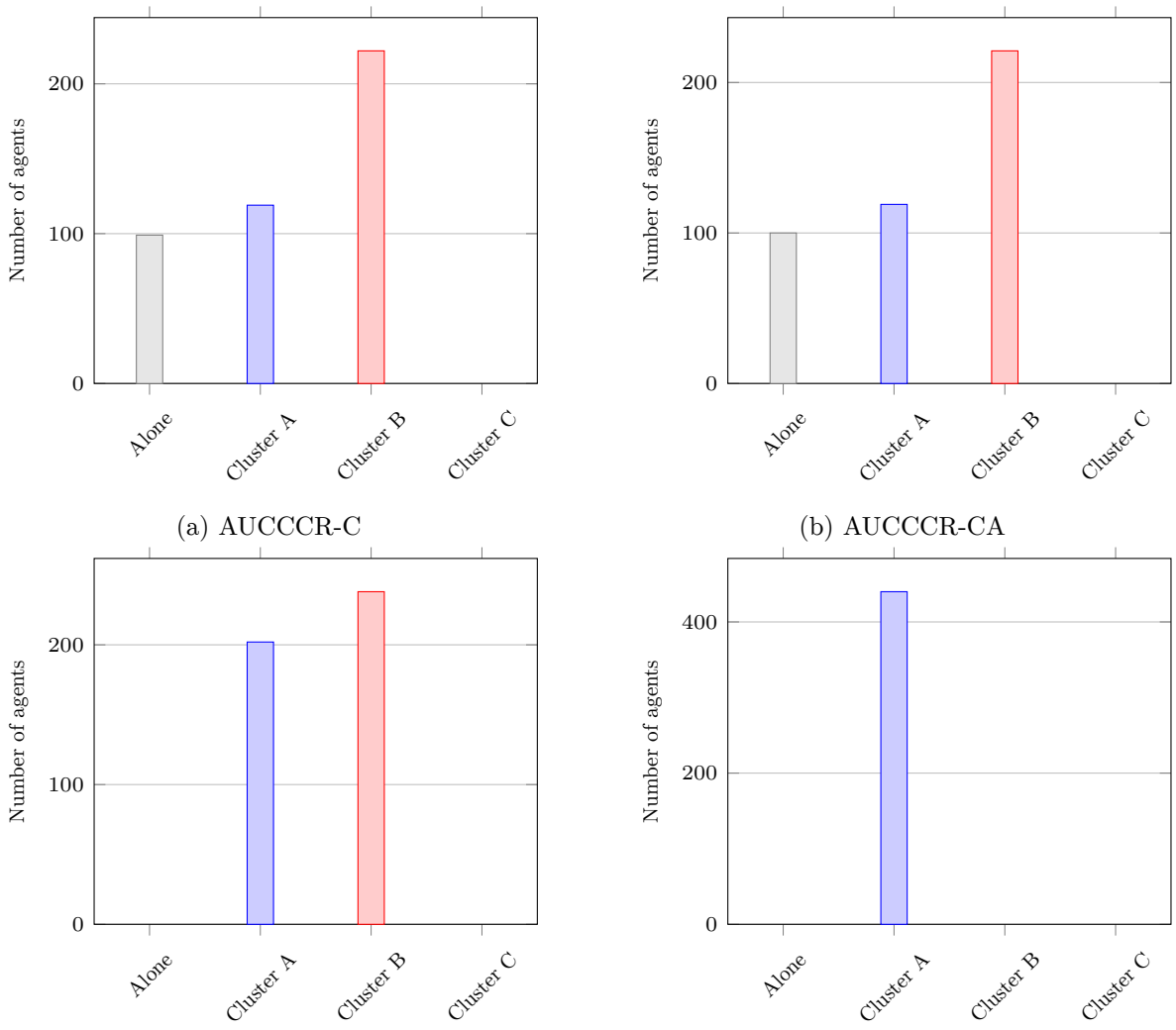
(c) k-means (d) OPTICS

Figure 4.23: Clusters found in BuddyMove@115 (Manifold MDS; colors are clusters, grey points are alone)



(a) Agents' losses caused (b) Share of agents having losses (c) Global utility achieved

Figure 4.24: Metrics in Wholesale for different scales by different algorithms



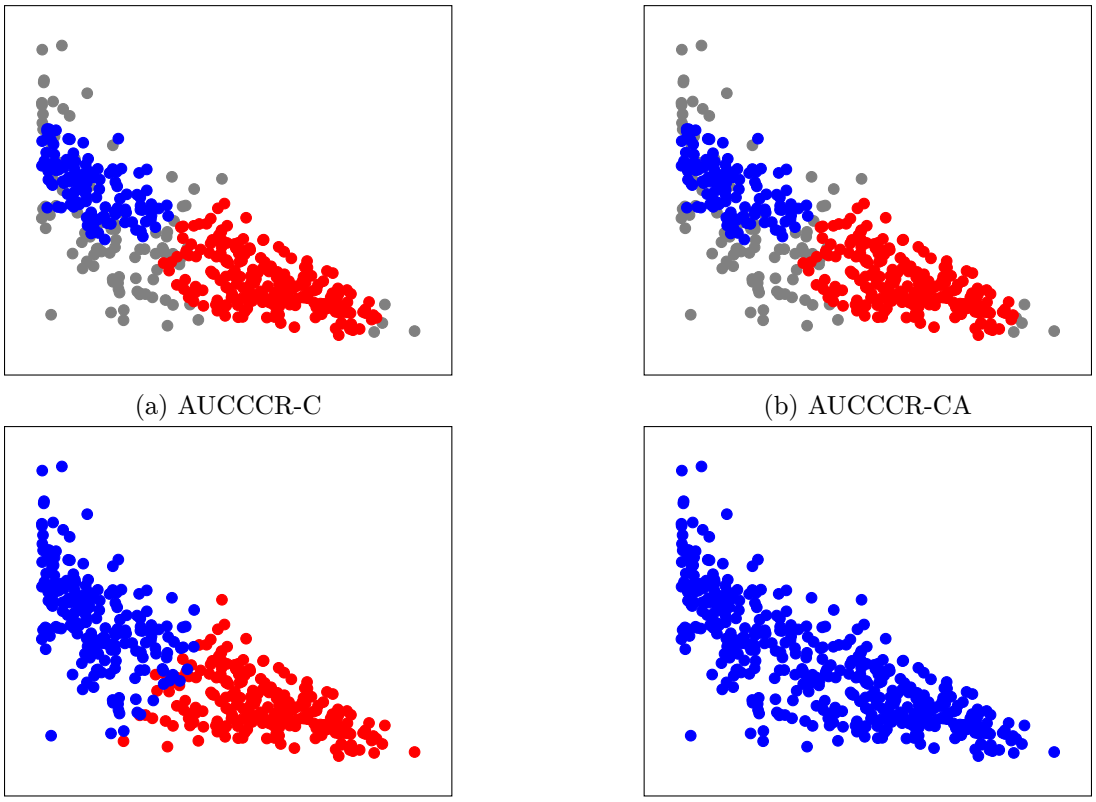
(a) AUCCCR-C

(b) AUCCCR-CA

(c) k-means

(d) OPTICS

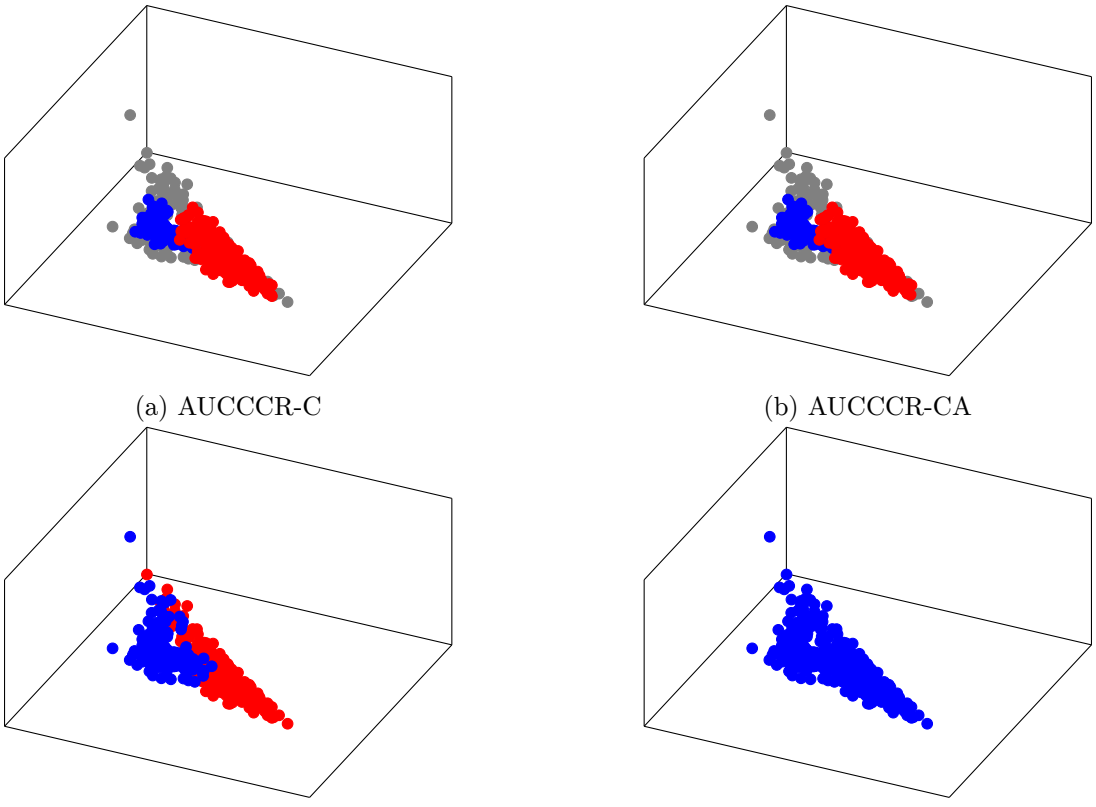
Figure 4.25: Clusters size in Wholesale for scale 45 by different algorithms



(a) AUCCCR-C (b) AUCCCR-CA

(c) k-means (d) OPTICS

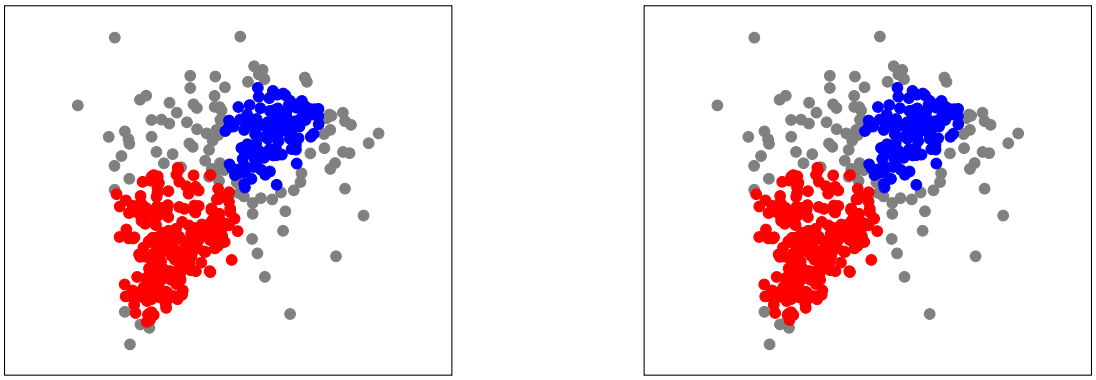
Figure 4.26: Clusters found in Wholesale@45 (colors are clusters, grey points are alone)



(a) AUCCCR-C (b) AUCCCR-CA

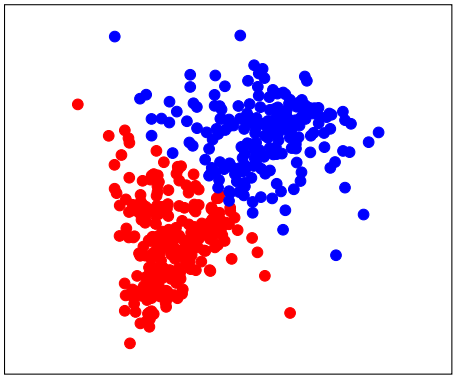
(c) k-means (d) OPTICS

Figure 4.27: Clusters found in Wholesale@45 (colors are clusters, grey points are alone)

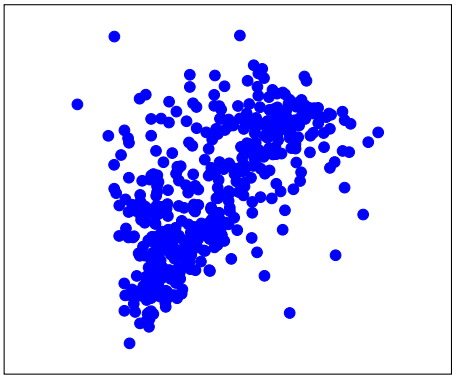


(a) AUCCCR-C

(b) AUCCCR-CA

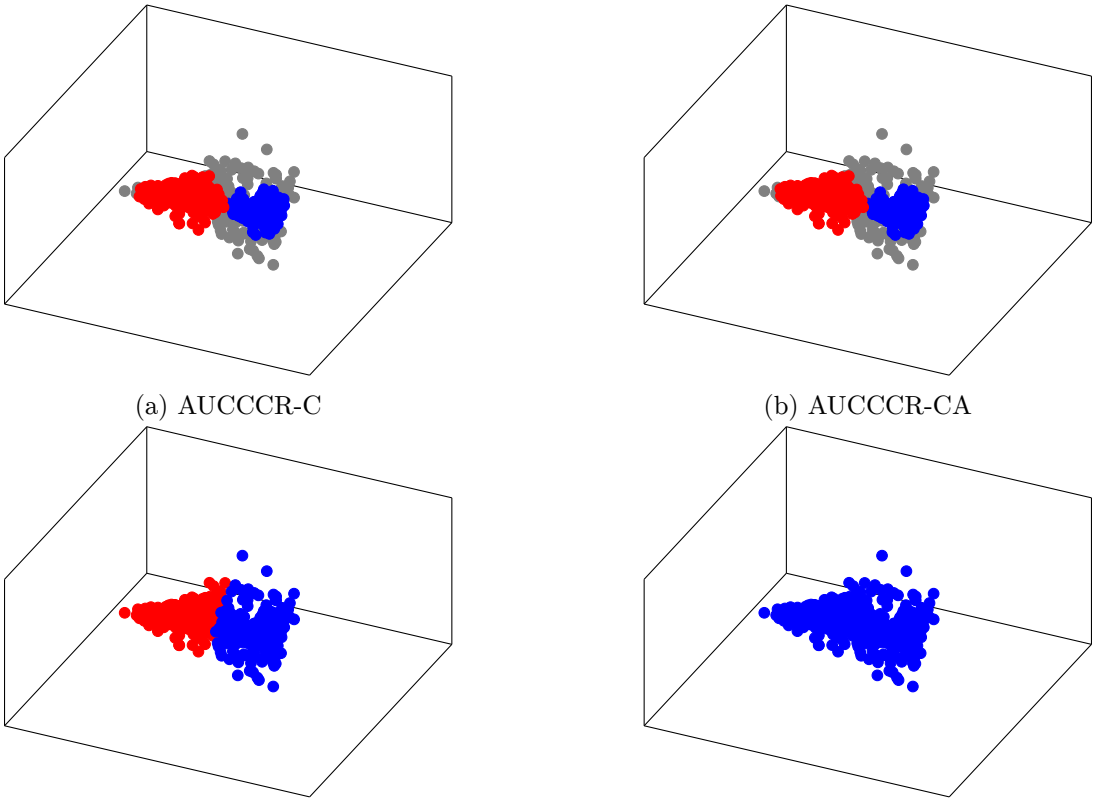


(c) k-means



(d) OPTICS

Figure 4.28: Clusters found in Wholesale@45 (Manifold MDS; colors are clusters, grey points are alone)



(a) AUCCCR-C (b) AUCCCR-CA

(c) k-means (d) OPTICS

Figure 4.29: Clusters found in Wholesale@45 (Manifold MDS; colors are clusters, grey points are alone)

Chapter 5

Application of clustering to distributed multi-task machine learning and individual interest analysis

I Introduction

In the two previous chapters, we presented our distributed multi-task machine learning system and a clustering algorithm designed to be used for cooperation recommendation. This chapter focuses on the application of our clustering algorithm to provide good recommendation for cooperation with our machine learning system.

Our original formalism for distributed multi-task machine learning involves a set of peers wanting to collaborate to learn similar but different functions, using a certain (common) machine learning method. Notice that, while we commonly use the term *peer* in Chapter 3, because of the original focus of that chapter on decentralized systems, the term *client* may also be used when talking about a federated setup[1]. In Chapter 4, we used the term *agent* to describe the same kind of entity, because agent is the term that fits our game theory-based formalism. In this chapter, as we do not want to use a term specific to a formalism that is not dominant in our analysis, we use the term *learner*.

Since our distributed multi-task learning system can use semi-local models (set of parameters shared by some but not all learners), tying similar learners together to improve the learning process, it can take advantage from an algorithm that can identify clusters of similar learners to automatically associate semi-local models with learners.

In the following, we evaluate how the use of semi-local models to which learners are assigned based on the output of our clustering algorithm can improve the multi-task learning process. We also see how, in practice, a vision centered in individual interests fits well with distributed multi-task machine learning.

II Method

We have our distributed multi-task machine learning system and our clustering algorithm designed to work with it to optimize the learning process by identifying similar learners, but how exactly do we make the two work together?

In our machine learning system, we have three kinds of models: global (shared by all learners), local (specific to one learner) and semi-local (shared by a subset of learners). A global model represents

[1] See Chapter 2, Section III.1 and Figure 2.3 for a definition of the different kinds of setups

what all learners have in common. A local model represents what a specific learner does not share with others. A semi-local models represents what some learners, but not all (a cluster of learners), have in common. Semi-local models allow more similar peers to have closer collaboration, sharing something that other learners do not share, which leads to better performance in situations where this is relevant (as shown in Chapter 3 in Sections II.1 and V.3.e). Semi-local models are where a clustering algorithm can be useful. Since clusters are subsets of relatively closer points, with points representing learners, it is logical to think that learners in a cluster would share a common semi-local model.

Since our clustering algorithms takes real vectors as input, we need to extract real vectors from our learners' neural networks. The most straightforward way would be to directly take the neural networks learned parameters, which are real values, but this is not a good way. Due to the nature of neural networks, similar learned tasks absolutely do not imply similar learned parameters. Neural network training algorithms are non-deterministic and even when two neural networks do compute the same function, it is very unlikely that their internal structure is similar. For this reason, we prefer a different method. Our intuition is that the identity of a neural network is more the function it is computing (output given a certain input) rather than the computation itself (the parameters). Thus, we decide to rely on the output of each learner's network on some input to describe it relatively to others. The idea is to start with a *local pre-training phase* before using our clustering algorithm to define semi-local models and proceed with distributed training. Each learner starts with a blank (randomly initialized) neural network with a (common) layout (defined by the designer of the system); these networks will later be used for distributed training but, for now, averaging is disabled and the training happens locally, on each learner's local training set. After the local pre-training phase, consisting in a reasonable amount of training (task dependent but should not be too long, or the time cost may offset the benefits, while still enough for learners models to converge to clearly different models, otherwise the process would not attain its goal, the right setting is a trade off), we have all neural networks work on the same benchmark data (a specific portion of dataset dedicated to this purpose). From the outputs of this phase, a real vector is extracted for each learner, the learner's *representative vector*. This real vector is to be used as input for our clustering algorithm.

The exact way real vectors should be extracted from the output of each learner's neural network depends on the task type. Vectors should describe what is each neural network's output for different types of input. Typically, for a classification task, we would group inputs in classes and then, for each learner, get a vector by averaging all output vectors (the real vector outputted by the neural network) for each given class input and concatenating the averaged vectors. This means that, if we have a classification task with 10 classes and 10 output values (like digit recognition), the representative vector would be in $\mathbb{R}^{100}$. More formally, if we have n input classes and m output values, then the representative vector is $\in \mathbb{R}^{n \times m}$. The case $n = m$ is very likely ($\mathbb{R}^{n^2}$) but it is also possible for n and m to be different. For example, a binary classification task may use only one output ($n = 2$, $m = 1$), this is the case in our VSN experiment.

It is also possible to extract this real vector from each learner's dataset or some learners' metadata (like location), if the one in charge of the configuration of the system knows a good way to perform such a classification. In that case, the algorithm responsible for this process is completely dependent on the task and thus we will not consider that possibility here (as this is a general evaluation).

The process of obtaining these vectors corresponds to the projector function p of our clustering model, as defined in Chapter 4 in Section II.1, as such we call it *projection* (as it is a projection function from the set of learners to a real vector space in which clustering takes place). As a consequence, the representative vector of a learner/learner a is called $p(a)$. For recall, the objective of the projector function is to associate a real vector (usable as input for our clustering algorithm) with each learner. After running the clustering algorithm, semi-local models (implemented in predefined parts of the neural networks) are assigned directly from clusters (one cluster, one semi-local model), then we can average models normally. The process is described visually in Figure 5.1.

This process is tailored for a federated setup, in which the clustering process can be done on a

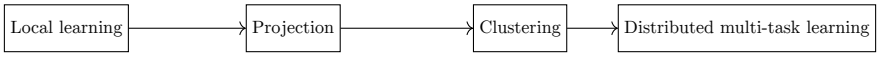


Figure 5.1: How clustering can be used to improve distributed multi-task machine learning

central server. However, it should be possible (not studied here) to adapt it to a decentralized setup. Since decentralized k-means is possible [FPG13], adapting our own clustering algorithm (k-means-based) may also be possible. Some kind of gossip consensus system [Fan+13] could make it possible to select a common set of benchmark cases. We leave the study of such a setup and the definition of a protocol to future work (we also cannot conclude on the possibility of such a setup).

Rather that projecting learners onto a real space ($\mathbb{R}^n$), other authors[Lon+17; ZBT19] proposed systems in which differences between learners are not represented by a distance in a real vector space but by the weighted edges of a graph (learners are nodes). Like us, these authors designed their representation of difference specifically for their learning system, as a consequence, they could not be applied directly to ours. These formalisms only consider proximity between learners individually rather than considering groups of learners (clusters), as it fits the way learners collaborate in these systems (putting a, difference-dependent, cost on the divergence learners' models). However, it would be possible to adapt such a formalization of similarity to our system using a community-detection algorithm[Blo+08] in place of the clustering algorithm. For our system, we preferred to use a formalism that directly fits our system rather than twisting an existing formalism designed for a different kind of collaboration method.

III Experimentation

For our experiments, we reuse two datasets used in Chapter 3: modified MNIST and VSN[2]. We first present VSN results, which are from a real application. Modified MNIST results are also presented but the clusters here are more obvious and, thus, do not allow comparing the performance of different clustering algorithms.

III.1 VSN

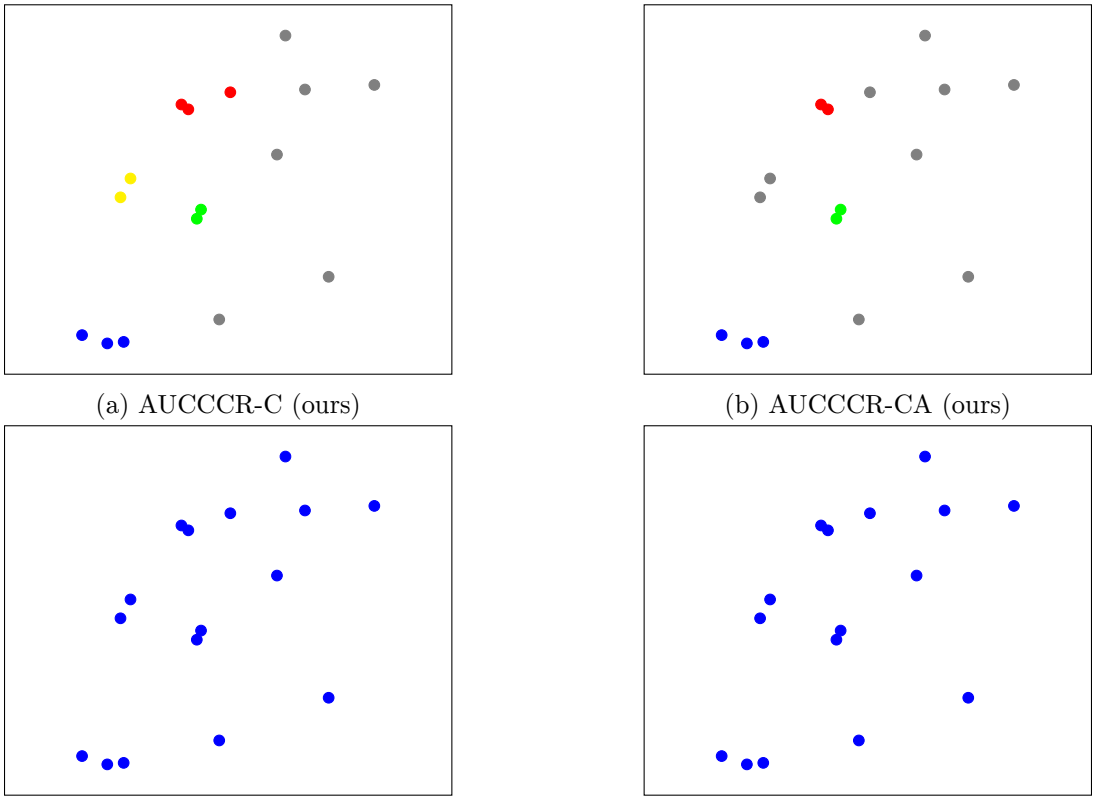
To evaluate our system in a real use case, we use the following task: vehicle recognition from sensors. We use a dataset from [DH04], which contains data from sensors (notably seismic and acoustic sensors) produced while some (military) vehicle passes near the sensors (grouped in nodes, which will be our learners). In this case, no obvious group affectation is possible without relying on clustering, allowing us to evaluate how our approach enables to effectively get better performance than a baseline.

The task proposed is to recognize the class of each vehicle (between *assault amphibious* or *dragon wagon*) from each sensor node's data (binary classification). To allow an MLP to perform this task, the data from each set of sensors (node) is first transformed into 50 seismic and 50 acoustic features (100 total inputs). This process (based on the Fast Fourier Transform) is described in the original paper ([DH04]).

For our distributed multi-task setup, we consider each sensors node as a learner. Each learner's task is to classify vehicles only from its local data. The tasks are similar, since all learners want to classify the same kinds of vehicles from the same kind of data, but different due to the location of sensors influencing their output.

To get real vectors for our clustering algorithms, we take locally pre-trained networks (one for each learner) and test them on a standard set of inputs. From the obtained output, we construct vectors $p(a) \in \mathbb{R}^2$ from the average output of networks on each type of vehicle.

[2]We do not use FEMNIST due to its the lack of clusters. Maybe a similar dataset not limited to US hand-writers would have offered more interesting features.



(a) AUCCCR-C (ours) (b) AUCCCR-CA (ours)

(c) k-means (d) OPTICS

Figure 5.2: Clusters found by various algorithms in VSN (colors are clusters, grey points are alone)

For this experiment we use 16 peers, each with a training set of 50 samples. The mini-batch size is 25. We use an MLP with a 100=50=20=1 layout (for details on this notation and on experiments with MLP in general, see Chapter 3, Section V.1), trained with stochastic gradient descent, $\lambda = 1$ sigmoid activation functions and a 0.1 learning rate. Our clustering algorithms use Euclidean distance for d, $n(x) = \frac{1}{1+15x}$ and $v(x) = \sqrt{x}$. $prc = 20$, $mmt = 5$. Results are average values over 10 runs.

We start with a local pre-training phase of 400 mini-batches. Then, we use the output of the pre-trained networks on a (identical for all peers) test set of 1000 samples to get inputs for clustering. With the output of our clustering algorithm, we define partial models to use for the real training phase of 400 mini-batches (we restart from scratch, using a new network with randomly initialized parameters, for a fair comparison with baseline results) and perform tests on 200 samples (specific to each learner). Here, the performance of our clustering algorithm is not evaluated by the affectation itself but by the performance of the resulting machine learning process (median accuracy).

Figure 5.2 gives examples of cluster affectations produced by different algorithms. Figure 5.3 compares the learning performance of the baseline from Chapter 3, the same distributed multi-task learning approach without benefits from clustering, to what can be achieved using affectations obtained from our clustering algorithms (k-means and OPTICS both failed to identify any cluster, returning a single global cluster containing all peers, thus we ignore them in the rest of this experiment, as they are equivalent to "no clustering").

While no obvious cluster affectation is visible, AUCCCR manages to find some clusters (differences between samples from the two variants of AUCCCR are more likely due to randomness than real differences between the two algorithms). k-means and OPTICS however both fail in this difficult case (using them would be equivalent to the baseline).

k-means and OPTICS fail to provide meaningful clustering information, not allowing any gain over the baseline (all logical affectations derivable from their outputs are in the baseline). With AUCCCR,

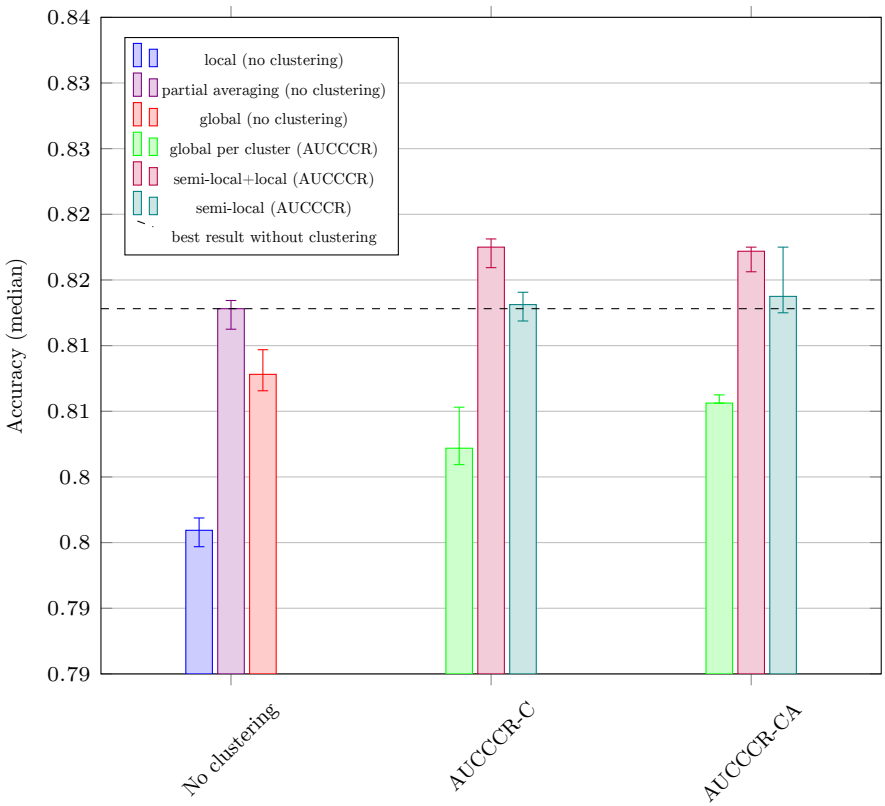


Figure 5.3: Different configurations possible with AUCCCR clustering compared to baseline configurations (no clustering) on the VSN dataset

however, we get two cluster affectations, from each of which several averaging schemes can be derived.

Interpreting clusters to define models can be done in various ways. We could have peers in each cluster learn a single, cluster-specific, model, without individual divergence nor collaboration outside a cluster. We could have collaboration between clusters but keep models completely identical in each cluster. Finally, we could allow divergence inside a cluster and collaboration between clusters while still maintaining closer collaboration inside each cluster. Here, we present the results of three different schemes, which implement, using partial models, the configurations we described (other configurations are possible but these seemed to be the most obvious to us):

- **global per cluster:** A 100-50-20-1 (full network) model for each cluster (clusterless peers are trained locally).
- **semi-local:** A 100-35-14-1 model for all peers and a 0-15-6-0 one (dependent on the previous one; see Chapter 3, Section II.3 for the definition of "dependency") for each cluster (leaving nothing local for peers in clusters).
- **semi-local+local:** A 100-35-14-1 model for all peers and a 0-7-3-0 one (dependent on the previous one; see Chapter 3, Section II.3 for the definition of "dependency") for each cluster (leaving a 0-8-3-0 local model for peers in clusters).

Notice that these are only three of the numerous possibilities offered in our case by our distributed multi-task machine learning approach. We chose them because they gave good results in Chapter 3 (in terms of configuration to use) but further optimizations are possible and may lead to better results, even though in Chapter 3 we concluded that the solution we used in semi-local+local was the best in those experiments.

As we see in our results (Figure 5.3), *global per cluster* does not beat the baseline (no clustering), likely due to many peers (6 out of 16) being out of clusters and therefore performing purely local learning (the performance is slightly higher than that of local learning). Semi-local achieve a baseline-like performance, likely because the gain from the clusters is cancelled by the loss caused by a lack of intra-cluster differentiation. Semi-local+local, however, manages to obtain (slightly) better-than-baseline results. The gain is limited but similar to what the baseline achieved with partial averaging compared to global.

III.2 Modified MNIST

The task associated with the MNIST dataset [LeC+98] is handwritten digit recognition. From this task, we want to obtain a cooperation recommendation task, paving the way for the application of AUCCCR to the distributed multi-task learning method presented in Chapter 3. This method implies cooperation between different MLPs (Multi Layer Perceptrons). Now, we want our MLPs to have more or less similar tasks. To this aim, like in Chapter 3, we simply modified the MNIST task for some of those MLPs: part of our MLPs have to perform the standard digit recognition task, while others have to perform a modified task where two digits are exchanged, so a [9] would be classified as an "eight" and vice versa. The clustering task consists in assigning MLPs with the same task to the same cluster and MLPs with different tasks to different clusters. Since the optimal affectation is known for this test (as tasks differences are set manually), the objective of this test is to obtain this optimal clustering affectation, we do not consider the distributed training that would come after (since it would be the same as what is obtained with the manual affectation studied in Chapter 3).

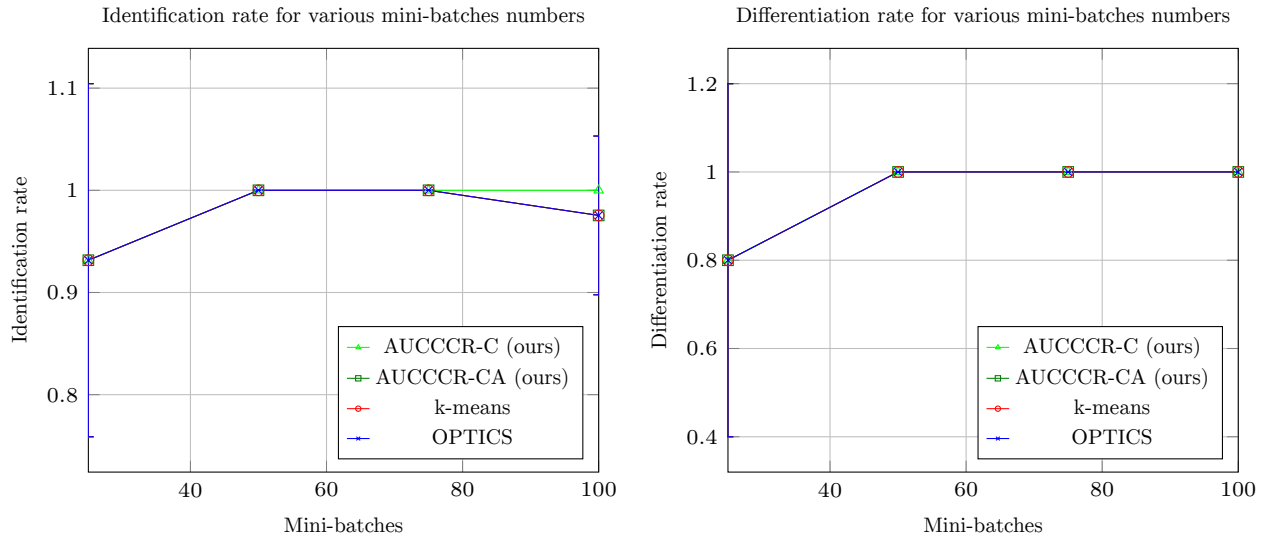
Regarding the input vectors for our clustering algorithm, we use the method described earlier, adapted to the MNIST dataset. After a reasonable amount of (independent) training (from 25 to 100 mini-batch of 1000 samples each), we made each MLP work on the test part of the MNIST dataset. For each different digit tested (from 0 to 9), we took the average of the output vectors of each MLP (10 real values, which are the MLPs' estimated likeliness of the input image to be each of the digits from 0 to 9) and concatenated them in one vector $p(a) \in \mathbb{R}^{100}$. These vectors will be used as input for our clustering algorithms.

We use two metrics to evaluate the quality of the resulting affectation (of learners to models): the Identification rate, which is the proportion of the pairs of MLPs with identical tasks being in the same cluster, and the differentiation rate, which is the proportion of the pairs of MLPs with different tasks being in different clusters. Since in this task, the optimal affectation is known (a cluster = a permutation), these two metrics allow evaluating how the results of the clustering is close to this optimal affectation. We want both values to be as close to 1 as possible, which corresponds to the optimal affectation. A lower identification rate means that some learners that should have been part of the same cluster are not part of the same cluster. A lower differentiation rate means that some learners that should not have been part of the same cluster are part of the same cluster.

We train 16 MLPs, nine of them performing a standard classification task and seven of them having digits 8 and 9 permuted. MLPs are trained each on a different (disjoint) part of the MNIST dataset training set of size 3500. The MLPs have a 784=300=100=10 configuration with $\lambda = 1$ sigmoid activation functions and a 0.1 learning rate. The $p(a) \in \mathbb{R}^{100}$ vectors are built from the 1000 first values of the MNIST dataset test part. Since this is just the pre-model affectation phase, as defined earlier, the training is purely local (no global no semi-local models).

Our clustering algorithms use Euclidean distance for d, $n(x) = \frac{1}{1+15x}$ and $v(x) = \sqrt{x}$. $prc = 20$, $mmt = 5$. Results are average values over 10 runs, error bars indicate standard deviation. Sample examples, based on a single run, are also given. In these, a color identifies a cluster, gray points represent standalone peers (or peers in a cluster of size 1). "AUCCCR-C" is Algorithm 8 in its normal variant, 'AUCCCR-CA" is the atomic variant.

The MLPs are trained with stochastic gradient descent, with a mini-batch size of 1000 and different numbers of mini-batches. Figure 5.4a presents the identification rate for various numbers of mini-



(a) Identification rate

(b) Differentiation rate

Figure 5.4: Metrics for various mini-batches numbers and different clustering algorithms for modified MNIST

batches and Figure 5.4b the differentiation rate.

We essentially observe that all algorithms remain close to perfect for all tests. While these tests on a very simple case do not allow determining which algorithm would be the best, they prove the validity of the idea of using clustering algorithms to provide cooperation recommendations for distributed machine learning.

Figure 5.5 gives examples of cluster affectations produced by AUCCCR-C. Since we used vectors from a space of dimension 100 ($p(a) \in \mathbb{R}^{100}$), we selected two relevant dimensions, 89 and 99 (corresponding to permuted digits), to make these 2D plots.

In that case, the clusters output are the same that an optimal manual affectation would have give.

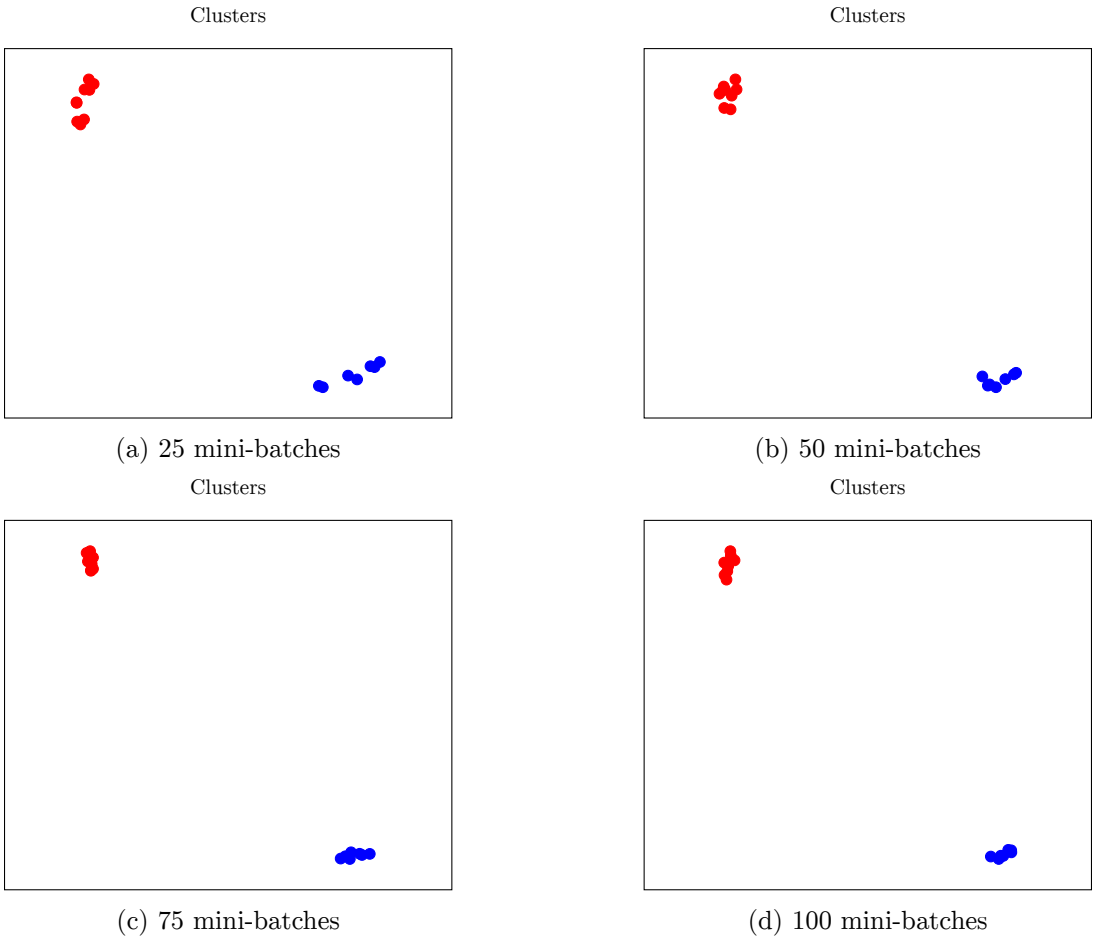
III.3 General analysis of results

We proved that our idea of using clustering algorithms to identify good collaboration opportunities in distributed machine learning is valid, with our modified MNIST example, and that it could lead to performance improvement in cases where no manual affectation is possible, with our VSN example.

We can conclude that our initial idea of using clustering to improve performance in distributed multi-task machine learning works as expected. Also, our VSN results show that AUCCCR is superior to k-means and OPTICS in this context.

IV Individual interest analysis

Our clustering system's design is based on the individual interest of learners. Such a design comes from the idea that learners may have more or less interest for collaboration in different contexts. This question is not only important in contexts where such a clustering algorithm can be used. More generally, we would like to see if distributed machine learning is in the interest of learners individually and not just collectively. The point of this section is to evaluate the benefits learners get from our distributed machine learning system in both kinds of situations: where clustering is used and where it is not used.



(a) 25 mini-batches (b) 50 mini-batches

(c) 75 mini-batches (d) 100 mini-batches

Figure 5.5: Clusters found by AUCCCR-C in MNIST for various numbers of mini-batches (colors are clusters, grey points are alone)

IV.1 With clustering (VSN)

Figure 5.6 represents accuracies for each peer in the same experiment as Figure 5.3. For readability reasons, results for AUCCCR-CA, which are similar to AUCCCR-C, are omitted. Peers are ordered by average (over configurations) median (over runs) accuracy for aesthetic reasons.

In this figure, we see that global averaging would have caused significant losses (compared to local learning) to many learners. The "global per cluster" clustering solution (the least efficient) may also cause losses compared to local training, but this effect is significantly lower than for global training, despite its better average accuracy. All other clustering solutions did not cause losses. This is also the case of partial averaging. Partial averaging seem to prevent losses by itself, even though clustering-based solutions, other than "global per cluster", give better results.

IV.2 Without clustering

IV.2.a FEMNIST

Among the datasets we tested in Chapter 3, VSN and FEMNIST differentiate by the fact that they have real learners. Thus, let us study FEMNIST to evaluate how our multi-task learning system performs regarding individual losses without clustering. Our FEMNIST task has three variants. FEMNIST-A, the classical FEMNIST task, with all characters (62 classes). FEMNIST-M, were some similar-looking characters are merged in a single class as proposed in the NIST Special Database 19 documentation [Nat16] (47 classes). FEMNIST-D, limited to digits (10 classes). The results are pre-

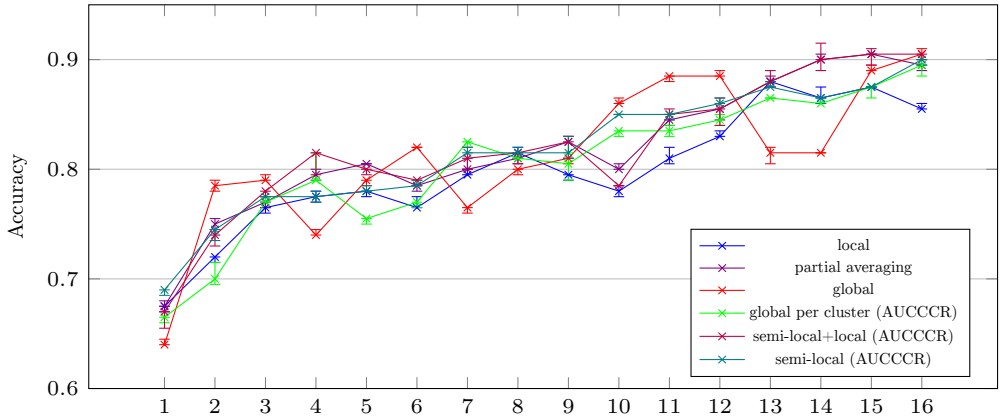


Figure 5.6: Individual peers' accuracies for various averaging levels on VSN (peers sorted by average (over configurations) median (over runs) accuracy) - Same experiment (other perspective) as Figure 5.3

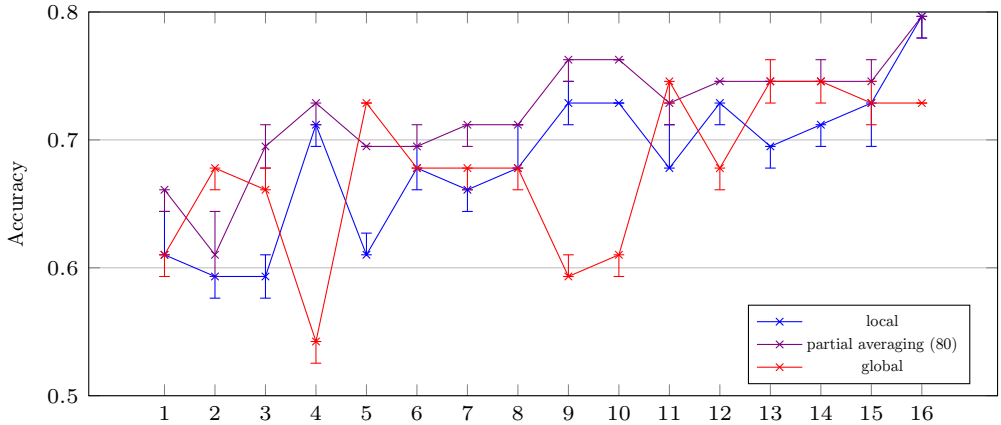


Figure 5.7: Individual peers' accuracies for various averaging levels on FEMNIST-A (peers sorted by average (over configurations) median (over runs) accuracy) - Same experiment (other perspective) as Figure 3.11

sented in Figures 5.7, 5.8 and 5.9 (in order, A, M and D variants). The raw data from which those results are extracted is the same as for Figures 3.11, 3.12 and 3.13. Peers are ordered by average (over configurations) median (over runs) accuracy for aesthetic reasons.

These figures show that partial averaging is over local learning for nearly all learners. The same is not true for global training. This last case can be better for some learners but it also causes significant losses to several learners. In an application with rational learners, this means that a global averaging solution would suffer from the collapse we mentioned when designing AUCCCR (learners preferring to be alone leaving clusters, reducing their value for other learners, which in turn leave and so on), unlike partial averaging. Some learners would have found global averaging better but this is not a stable situation, as several learners have interest to leave. We can conjecture that learners more "in the middle of the crowd" (with expectations close to the average of the group) would prefer global averaging above anything, while it would be the worst solution for more "dropout" learners.

IV.2.b Modified MNIST

Figure 5.10 represents the accuracy for each peer in the same experiment as Figure 3.20. Since in this experiment the original position of peers are directly linked to their tasks, the original order of peers is kept.

This figure shows us an interesting phenomenon. In this test, peers 1 to 9 have the same task,

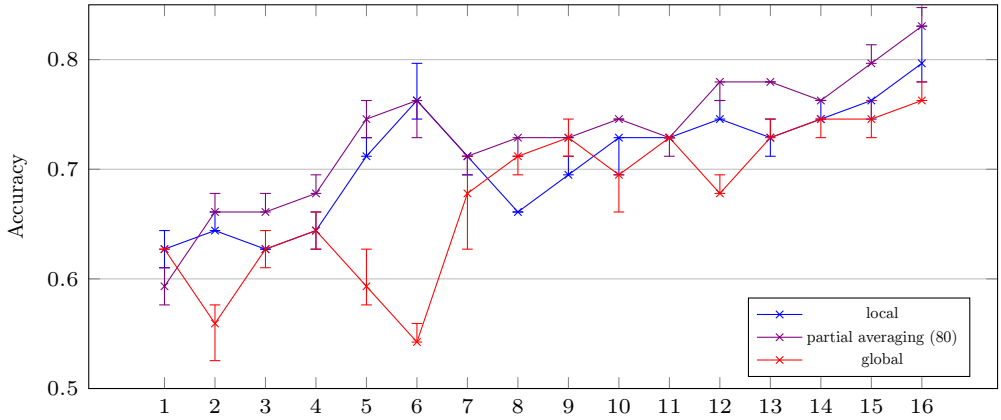


Figure 5.8: Individual peers' accuracies for various averaging levels on FEMNIST-M (peers sorted by average (over configurations) median (over runs) accuracy) - Same experiment (other perspective) as Figure 3.12

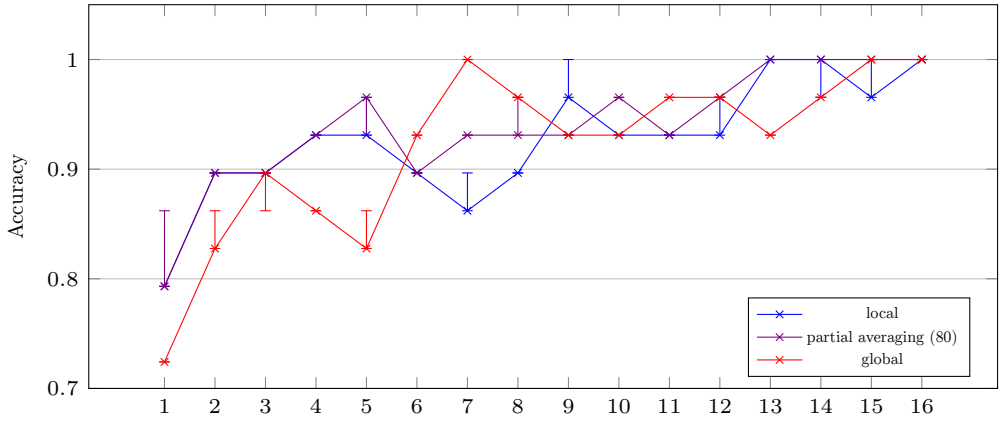


Figure 5.9: Individual peers' accuracies for various averaging levels on FEMNIST-D (peers sorted by average (over configurations) median (over runs) accuracy) - Same experiment (other perspective) as Figure 3.13

so do peers 10 to 16. The first group consists of 9 peers, the other of 7 peers. There is no notable difference between these two groups with local training nor with partial averaging. We just see that partial averaging is significantly better than local training for all peers. However, when it comes to global averaging, we see that some peers from the first group may benefit from it but others not, more importantly, global averaging causes a major loss for all peers of the second group. We can conjecture that, when using global averaging, a "majority voting" effect appears. The first group, the larger, "dominates" the second, forcing the global solution to be optimal for the first group and bad for the second. That said, even the first group does not significantly benefit from global averaging. The "struggle" between the two groups for the "control" of the training offsets any gain, even for the larger group. Such a "struggle" does not appear with partial averaging.

To complete this evaluation on modified MNIST, we analyze individual peers over two variables defined in Chapter 3: difference rate (number of digits permuted) and number of peers with permutation. These results are presented in Figure 5.11 for the difference rate, which is obtained for the same raw data as Figure 3.25 and Figure 5.12 for the numbers of peers with permutations, which is obtained for the same raw data as for Figure 3.26.

Figure 5.11 confirms that the difference rates strongly affect minority peers' performance. We see that majority peers suffer from losses also. This is likely due to reduced efficiency of the training

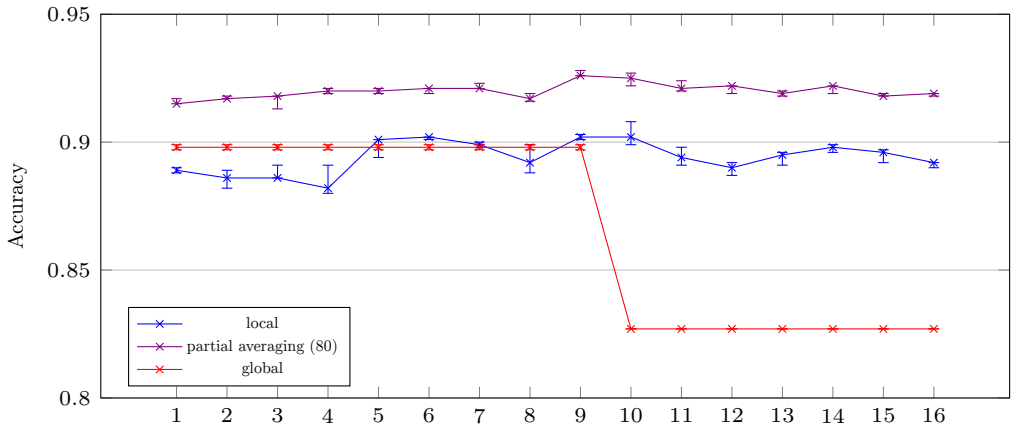


Figure 5.10: Individual peers' accuracies for various averaging levels on modified MNIST (peers in original order) - Same experiment (other perspective) as Figure 3.20

process. The difference rate has a much lower effect on partial averaging, but we still see that a very high difference also affects partial averaging performance and some kind of majority effect becomes visible on it only for 10 permuted digits.

Figure 5.12 confirms that the effect on modified MNIST is more due to a majority vote than just distance to the average. Effectively, increasing the number of peers with permutation only slightly reduces the advantage of majority peers, though this reduction is still clearly visible, proving the existence of a distance effect. However, when groups are of equal sizes, the gap is much lower. Notice that this majority effect is specific to modified MNIST, due to the "binary" nature of the differences (permutation or no permutation). No such effect is visible on the digit-classification task FEMNIST-D (Figure 5.9).

IV.3 General analysis of results

We confirmed that our clustering system, and partial averaging, when clustering is not relevant, respect the individual interest of learners. Also, we saw that, when forced to collaborate without any freedom, learners tend to enter a form of competition, which is won by the most "central" or the most "numerous" learners.

While one may fear that distributed multi-task learning could lead to losses for some learners, our results indicate that our partial averaging system is rather resilient to that issue, unlike complete averaging.

V Conclusion

In this chapter, we showed that our idea of using clustering algorithms to identify good collaboration opportunities in distributed machine learning can help determine how peers should collaborate with each other in a decentralized learning context. We showed that this idea, when applied with our clustering algorithm AUCCCR, could lead to performance improvement in cases where no manual affectation is possible (with our VSN example).

We also showed that partial averaging in general is much more respectful of individual interests than global averaging, which can cause significant losses to "dropout" learners.

A possible extension of this work would be to use a hierarchical version of AUCCCR (future work) with our distributed multi-task machine learning system on a task where this would be pertinent, such as natural language processing with national variants of a language as a first level of hierarchy and regional variants as a second. Hierarchical clustering, unlike classical clustering, can produce a

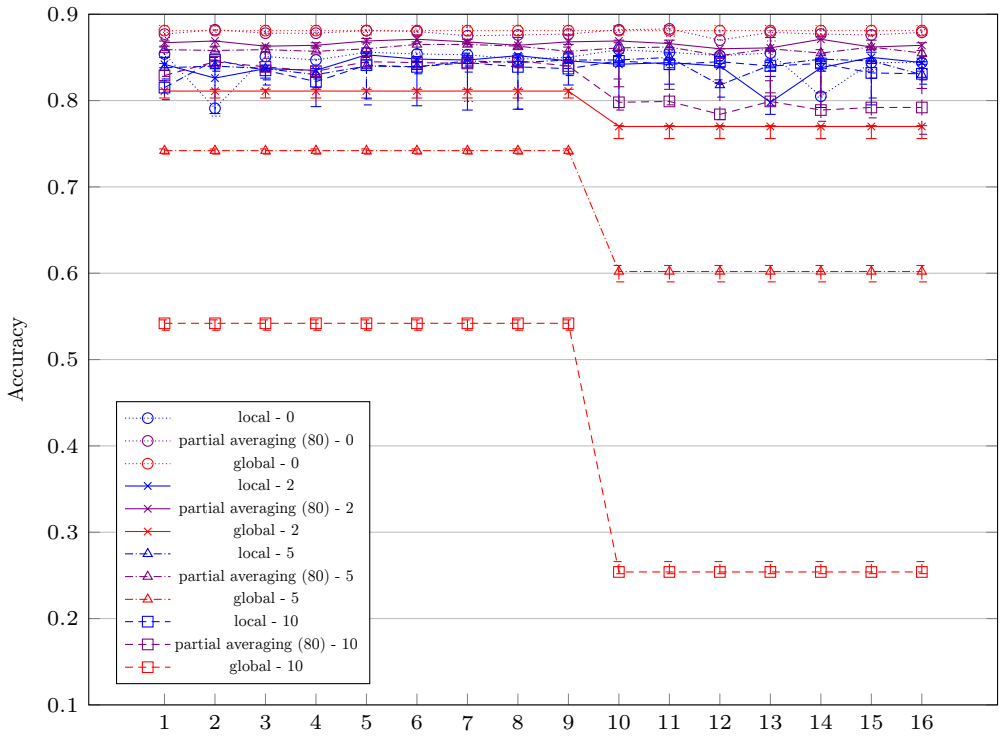


Figure 5.11: Individual peers' accuracies for various averaging levels and difference rates on modified MNIST (peers in original order) - Same experiment (other perspective) as Figure 3.25

complete hierarchy of clusters, subclusters, etc. Our multi-task machine learning system allowing a complex hierarchy of semi-local models, a hierarchical clustering collaboration recommender would be able to exploit its flexibility better than with classical clustering. With such a system, we could have a semi-local model covering a certain set of peers and several other semi-local models covering subsets of this set. This way, we could exploit different levels of similarity between learners more deeply.

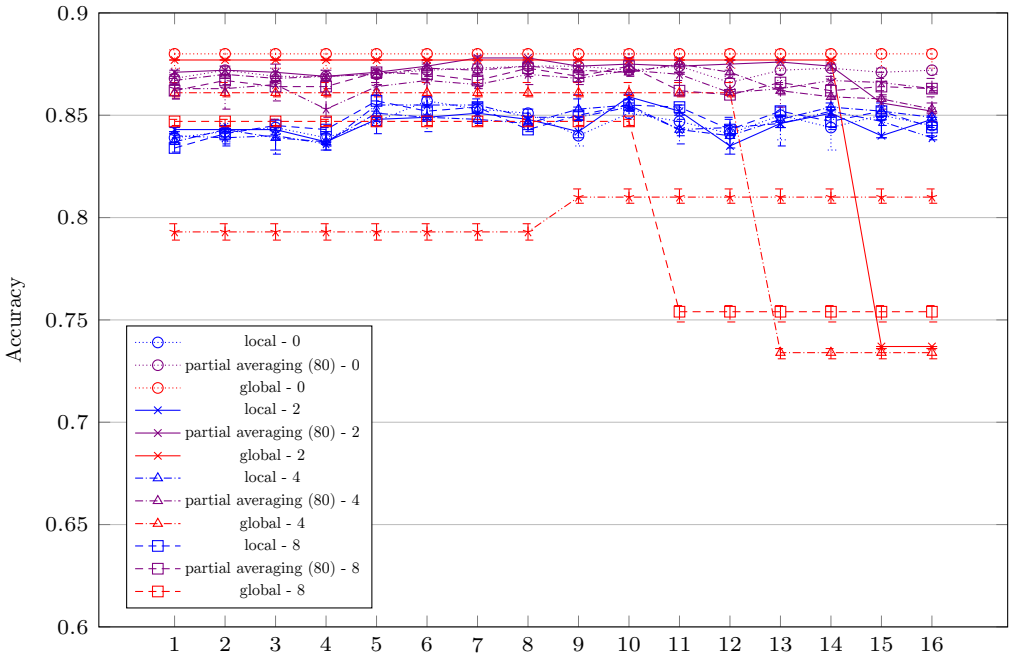


Figure 5.12: Individual peers' accuracies for various averaging levels and numbers of peers with permutation on modified MNIST (peers in original order) - Same experiment (other perspective) as Figure 3.26

Chapter 6

Conclusion

I Objective

Machine learning is widely used in industry and lots of research works have been produced in this field. Despite this, effective application of machine learning in a distributed context remains rare, and this situation is partially caused by a relative lack of research work in this direction. Notably, the problem of distributed multi-task machine learning with neural networks lacked effective solution prior to our work.

This thesis started with the following question: "How can we have several agents aiming to learn similar but different functions collaborate to achieve their objective better than alone?". We identified several challenges to overcome in order to find an answer to this question.

Our main challenge was the combination of two elements: distributed learning and multi-task learning. Distributed learning comes in two variants: decentralized and federated. In a federated setup, while the learning process itself is done on clients, the coordination of clients is managed by a central server. In a decentralized setup, peers have to coordinate themselves, without the help of a central server.

Outside a multi-task context, this problem had already been studied by other researchers. This allowed us to find a proper existing solution: *Federated Averaging* [McM+17]. While designed for a federated setup, this solution can be extended to decentralized setups, relying on *gossip averaging* [JMB05].

However, Federated Averaging does not tackle the other element of our main challenge: multi-task learning. A large part of our work thus consisted in proposing a solution for distributed multi-task learning. We wanted our approach to be flexible, enabling engineers to optimize the learning process notably based on which learners have the most similar tasks and on which parts of those tasks (implemented in different parts of the learned models) are the most similar.

The ability to optimize the learning process based on the similarity of learners' tasks leads to another question. Since learners are likely to be numerous, this process should be automated. Furthermore, while in some situations metadata about learners may enable to easily perform this process automatically (e.g. natural language processing, when location can be used) in some other situations (e.g. handwriting recognition), it is harder to define which learners are more similar than others.

To tackle this issue, we decided to abstract this problem to something more general: cooperation recommendation. The principle of cooperation recommendation is to provide recommendations to agents on groups they should form with other agents to cooperate on some task; recommendations agents may reject if they are not in their individual interest. This left us with two question to solve: first, find a proper solution to cooperation recommendation and second, effectively use this general approach in a machine learning context.

In addition, another question emerged: may collaborative learning have negative impact for some learners while still being positive for the whole population of learners?

II Contributions

Our contributions can be divided in three parts, each corresponding to a chapter of this thesis:

- Distributed multi-task learning proper [Chapter 3]
- Cooperation recommendation applicable to distributed multi-task learning [Chapter 4]
- Evaluation of our cooperation recommendation system and exploration of learners' individual interest [Chapter 5]

II.1 Distributed multi-task learning proper

The motivation of this work is to provide an effective solution to the distributed multi-task machine learning problem we defined as our main objective. Our approach is based on the same principle as Federated Averaging [McM+17] to enable distributed learning: averaging the parameters (real values) learned by learners. While this base approach is simple, enabling multi-task learning with this system required developing new concepts.

The key concept of this approach is the notion of *partial model*, enabling a partial averaging process where only portions of learners' neural networks are averaged, rather than the whole neural network. Partial models allow learners to learn collaboratively (on a shared model) while still being able to diverge from each other (with their personal model).

We extended this beyond just having shared models (*global model*) and personal models (*local models*) with *semi-local models*, shared by several learners but not all. This enables to group more similar learners together, further optimizing the collaborative learning process.

Since the geometry of neural networks is defined by the neurons, we wanted the models to be associated with neurons, but the parameters to be averaged are (mostly) weights connecting two neurons. This raised the question of which model parameters connecting two neurons from different models should be part of. To answer this question, we introduced the notion of *dependency* between models. If a model A depends on a model B, then all learners implementing model A must also implement model B. This allowed us to assign the parameters connecting two neurons from different models to a single model, the lower in the hierarchy of dependencies.

We evaluated our approach with several experiments on various datasets: FEMNIST [Cal+19], modified MNIST [LeC+98], VSN [DH04] and an artificial associative task. This evaluation showed that our system performs better than reference (non multi-task distributed learning and local learning) and proved the interest of the two novel notions we introduced: partial models and dependency.

II.2 Cooperation recommendation applicable to distributed multi-task learning

Our objective for this work was to create an algorithm that would enable to automatically group learners together for optimized multi-task learning. To create our cooperation recommendation system, we started from the general formalism of *Hedonic Games*. There is no general solution to the problem formalized by hedonic games and existing algorithms based on this formalism were not applicable to our situation.

Hedonic games rely on the notion of *utility*, the interest each agent (for example, a learner) has for a certain group. Assuming that this utility can be computed by a certain formula based only on the size of the group and the distance between the agent and the barycenter of the group, we managed to reformulate our hedonic game as a clustering problem that can be solved by an algorithm similar in its principle to what is used for centroid-based clustering.

The best known algorithm for centroid-based clustering is k-means [Llo82]. However, this algorithm is only based on distance to barycenter, not on the size of groups. Thus, we created a novel algorithm using the same iterative local optimization principle as k-means, but able to take the group size into account.

We first evaluated our clustering algorithm on synthetic data: Gaussian mixtures. It exhibited good performance compared to reference (k-means and OPTICS [Ank+99]). Also, while our original idea was that larger groups would always be better, we showed that this algorithm still works well in situations were the utility function of agents is defined to prefer large enough but not too large groups.

We also evaluated our clustering algorithm on two real datasets: the BuddyMove [RA14] dataset and the Wholesale [Fer11] dataset, which corresponds to potential real applications of cooperation recommendation. Similarly to the case of Gaussian mixtures, our algorithm showed good performance compared to reference (k-means and OPTICS).

II.3 Evaluation of our cooperation recommendation system and exploration of learners individual interest

For this part of our work, we combined our cooperation recommendation system and our distributed multi-task learning system and study more precisely the implication of the notion of individual interest in this context. The first step for this contribution was to precisely define how our cooperation recommendation system and our distributed multi-task learning system could work together. While the data, extracted from learners, to be used as input for the clustering algorithm could come from any source (the most likely, outside what we did with benchmarking pre-trained models, is metadata on the learner), we still had to choose one to use for our experiments, and we wanted it to be as general as possible.

We choose to extract this data from outputs of locally pre-trained neural networks on a common benchmark dataset. With this approach, we showed the efficiency of our cooperation recommendation system combined with our distributed multi-task learning system, notably its ability to reach a global optimum (it terms of quality of learner groups) on a modified MNIST task and its ability to improve performance in a VSN task, where manual optimization is not straightforward.

We further explored the question of individual interest of agents and found that, while Federated Averaging often caused individual losses (in cases where it is globally better than local learning), our distributed multi-task learning system is more respectful of individual interest. Performance in this context is also improved by our cooperation recommendation system. This answers our question on possible negative individual impact of collaborative learning: it is not a significant issue with our system.

III Future developments

III.1 Technical aspects

Our system was tested on a range of possible applications, but this does not cover the whole range of neural networks applications. The networks we considered we rather small compared to what is used in modern deep learning. We also did not test our system with recurrent neural networks like LSTM, Convolutional Neural Networks or on more complex deep neural networks. Evaluating our system in these contexts would be a major step toward effective adoption for several applications. Also, while our distributed multi-task machine learning system is working, it can still be improved. Notably, we did not consider practical communication limitations. Working on reducing the communication overhead or adaptation to unreliable networks may be interesting.

Also, neither our distributed multi-task machine learning system nor our cooperation recommendation system were created for hostile contexts. Working on protecting our system for potential attackers could be interesting, notably with its ability to be used in decentralized contexts. This may notably include application with our previous work on Robust Privacy-Preserving Gossip Averaging [BFT19].

Concerning our cooperation recommendation system, a significant development would be to create a hierarchical version of our clustering algorithm, AUCCCR, and to use it with our distributed multi-task machine learning system to create a hierarchy of semi-local models that would exploit more of

the possibilities of the dependency system in more complex applications (such as natural language processing).

Finally, the considerations on individual interest we presented may benefit from a more advanced game theoretical study (where individual unsatisfaction may occur and affect the whole system).

III.2 Practical aspects

Outside technical considerations, effective adoption of the methods proposed in this thesis will imply several practical challenges. The first of these would be to appeal to actors of the machine learning ecosystem. This should not be too hard, as major companies are already working on federated machine learning and non-profit organizations are interested in decentralized solutions to provide alternatives to large companies.

Organizations adopting our system would also have to make users adopt our system and get benefits from it. For the federated variant, we can assume that major technological companies, already working on federated learning, will find a way to make our system profitable. It will improve the efficiency of their machine learning based services (multi-task aspect) while delegating training tasks to clients' devices (distributed aspect). The idea of "personalized" machine learning and keeping data local could appeal to users wanting more efficient and personalized services while still preserving their privacy. The decentralized variant (or a potential lightweight federated variant, with low synchronization overhead) may also enable non-profit organizations to provide effective machine learning-based services without having to invest in costly infrastructure. They may thus become a more effective alternative to major companies. While the respect of individual interest may also appear as a good feature to client, this affirmation must be moderated. In the past, Google's game theory-optimized advertisement auction system did not convince clients unaware of game theory, even though they were professional clients. However, in our case, the system is no unintuitive as this example was (the auction was won by the highest bidder but the price was the second bidder's one, which is an optimal auction system), so math-free communication should work to present this feature of our system to the general public.

Other practical concerns include legal considerations, notably data protection regulations such as the European General Data Protection Regulation (GDPR), or similar regulations that can be adopted elsewhere (CCPA, PIPL...). At the time of writing this text, GDPR is still too recent to know how courts will effectively rule cases linked to it, notably in the case of machine learning. While some raised concerns on GDPR making most machine learning applications effectively illegal (due to models being based on private data), it seems highly improbable that effective application of the law will be restrictive to such an extreme point, due to the major business interests associated with machine learning. We may thus assume that if our system gets adopted by major companies, such extreme law interpretations would not challenge it. Since machine learning models, while derived from private data, are likely to be, in practice, not considered as private data themselves, companies' lawyers are likely to present valuables arguments in favor of federated/decentralized learning (including our system), which is more protective of the training (private) data (kept local rather than sent to a central server).

Bibliography

[Ach+07] Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, Ina Müller-Gorman, and Arthur Zimek. "Detection and Visualization of Subspace Cluster Hierarchies." In: *Advances in Databases: Concepts, Systems and Applications: DASFAA 2007*. 2007, pp. 152–163.

[AH15] Subutai Ahmad and Jeff Hawkins. "Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory." In: *CoRR* (2015).

[AHS85] David H. Ackley, Geoffrey Everest Hinton, and Terrence Joseph Sejnowski. "A Learning Algorithm for Boltzmann Machines." In: *Cognitive Science* 9.1 (1985), pp. 147–169.

[Alo+09] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. "NP-hardness of Euclidean sum-of-squares clustering." In: *Machine Language* 75.2 (2009), pp. 245–248.

[Ang+18] Cosimo Anglano, Massimo Canonico, Paolo Castagno, Marco Guazzone, and Matteo Sereno. "A game-theoretic approach to coalition formation in fog provider federations." In: *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*. 2018, pp. 123–130.

[Ank+99] Mihael Ankerst, Markus Breunig, Hans-Peter Kriegel, and Joerg Sander. "OPTICS: Ordering Points To Identify the Clustering Structure." In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. Vol. 28. 2. 1999, pp. 49–60.

[Ari+19] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. "Federated Learning with Personalization Layers." 2019.

[AS16] Haris Aziz and Rahul Savani. "Hedonic Games." In: *Handbook of Computational Social Choice*. 2016, pp. 356–376.

[AV07] David Arthur and Sergei Vassilvitskii. "K-means++: The Advantages of Careful Seeding." In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Vol. 8. SODA '07. 2007, pp. 1027–1035.

[Bel+18] Aurélien Bellet, Rachid Guerraoui, Mahsa Taziki, and Marc Tommasi. "Personalized and Private Peer-to-Peer Machine Learning." In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. Vol. 84. Proceedings of Machine Learning Research. 2018, pp. 473–481.

[BFT19] Amaury Bouchra Pilet, Davide Frey, and François Taïani. "Robust Privacy-Preserving Gossip Averaging." In: *21st International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2019)*. Vol. 11914. Lecture Notes in Computer Science. 2019, pp. 38–52.

[BFT20] Amaury Bouchra Pilet, Davide Frey, and François Taïani. "Foiling Sybils with HAPS in Permissionless Systems: An Address-based Peer Sampling Service." In: *25th IEEE Symposium on Computers and Communications (ISCC 2020)*. 2020.

[BFT21a] Amaury Bouchra Pilet, Davide Frey, and François Taïani. "AUCCCR: Agent Utility Centered Clustering for Cooperation Recommendation." In: *9th International Conference on Networked Systems (NETYS 2021)*. 2021.

[BFT21b] Amaury Bouchra Pilet, Davide Frey, and François Taïani. "Simple, Efficient and Convenient Decentralized Multi-Task Learning for Neural Networks." In: *19th Symposium on Intelligent Data Analysis (IDA 2021)*. 2021.

[BG97] Ingwer Borg and Patrick J. F. Groenen. *Modern Multidimensional Scaling - Theory and Applications*. 1997.

[Blo+08] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Étienne Lefebvre. "Fast unfolding of communities in large networks." In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (2008), P10008.

[BMS96] Paul Stephen Bradley, Olvi L. Mangasarian, and William Nick Street. "Clustering via concave minimization." In: *Proceedings of the 9th International Conference on Neural Information Processing Systems (NIPS 1996)*. NIPS'96. 1996, pp. 368–374.

[Bon+19] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Hugh Brendan McMahan, Timon van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. "Towards Federated Learning at Scale: System Design." In: *Proceedings of Machine Learning and Systems*. Vol. 1. 2019, pp. 374–388.

[Cal+19] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, Hugh Brendan McMahan, Virginia Smith, and Ameet Talwalkar. "LEAF: A Benchmark for Federated Settings." 2019.

[Cam+15] Ricardo J. G. B. Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. "Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection." In: *ACM Transactions on Knowledge Discovery from Data* 10.1 (2015).

[Cau47] Augustin Louis Cauchy. "Méthode générale pour la résolution des systèmes d'équations simultanées." In: *Comptes rendus hebdomadaires des séances de l'Académie des Sciences*. Vol. 25. 1847, pp. 536–538.

[CB19] Luca Corinzia and Joachim M. Buhmann. "Variational Federated Multi-Task Learning." 2019.

[Cha03] Charles Elkan. "Using the Triangle Inequality to Accelerate k-Means." In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML 2003)*. ICML'03. 2003, pp. 147–153.

[Che+17] Min Chen, Yixue Hao, Kai Hwang, Lu Wang, and Lin Wang. "Disease Prediction by Machine Learning Over Big Data From Healthcare Communities." In: *IEEE Access* 5 (2017), pp. 8869–8879.

[CST18] Sebastian Caldas, Virginia Smith, and Ameet Talwalkar. "Federated Kernelized Multi-Task Learning." In: *SysML Conference 2018*. 2018.

[DDT20] Enmao Diao, Jie Ding, and Vahid Tarokh. "HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients." 2020.

[Dem+87] Alan John Demers, Dan H. Greene, Carl H. Hauser, Wes Irish, John Larson, Scott J. Shenker, Howard E. Sturgis, Dan C. Swinehart, and Doug B. Terry. "Epidemic Algorithms for Replicated Database Maintenance." In: *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC)*. 1987, pp. 1–12.

[DG80] Jacques H. Drèze and Joseph Greenberg. "Hedonic Coalitions: Optimality and Stability." In: *Econometrica* 48.4 (1980), pp. 987–1003.

[DH04] Marco F. Duarte and Yu Hen Hu. "Vehicle classification in distributed sensor networks." In: *Journal of Parallel and Distributed Computing* 64.7 (2004), pp. 826–838.

[DKM20] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. "Adaptive Personalized Federated Learning." 2020.

[DLR77] Arthur Pentland Dempster, Nan McKenzie Laird, and Donald Bruce Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm." In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 1–38.

[Duo+15] Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. "Low Resource Dependency Parsing: Cross-lingual Parameter Sharing in a Neural Network Parser." In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. 2015, pp. 845–850.

[DV05] Ian Davidson and Sergei Vassilvitskii. "Agglomerative Hierarchical Clustering with Constraints: Theoretical and Empirical Results." In: *Knowledge Discovery in Databases: PKDD 2005*. Vol. 3721. Lecture Notes in Computer Science. 2005, pp. 59–70.

[Est+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xu Xiaowei. "A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. 1996, pp. 226–231.

[Est02] Vladimir Estivill-Castro. "Why so many clustering algorithms: a position paper." In: *ACM SIGKDD Explorations Newsletter* 4 (2002), pp. 65–75.

[Fan+13] Maria Pia Fanti, Mauro Franceschelli, Agostino Marcello Mangini, Giovanni Pedroncelli, and Walter Ukovich. "Discrete consensus in networks with constrained capacity." In: *52nd IEEE Conference on Decision and Control*. 2013, pp. 2012–2017.

[Fer11] Nuno Gonçalo Costa Fernandes Marques de Abreu. "Análise do perfil do cliente Recheio e desenvolvimento de um sistema promocional." MA thesis. Instituto Universitário de Lisboa, 2011.

[FMO20] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. "Personalized Federated Learning with Theoretical Guarantees: A Model-Agnostic Meta-Learning Approach." In: *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*. Vol. 33. 2020, pp. 3557–3568.

[FPG13] Jérôme Fellus, David Picard, and Philippe-Henri Gosselin. "Decentralized K-Means Using Randomized Gossip Protocols for Clustering Large Datasets." In: *IEEE 13th International Conference on Data Mining Workshops*. 2013, pp. 599–606.

[Fuk80] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." In: *Econometrica* 36 (1980), pp. 193–202.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 2016.

[GCT14] Nuwan Ganganath, Chi-Tsun Cheng, and Chi Kong Tse. "Data Clustering with Cluster Size Constraints Using a Modified K-Means Algorithm." In: *2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. 2014, pp. 158–161.

[Ger99] Felix A. Gers. "Learning to Forget: Continual Prediction with LSTM." In: *9th International Conference on Artificial Neural Networks: ICANN '99*. 1999, pp. 850–855.

[GN02] Michelle Girvan and Mark E. J. Newman. "Community structure in social and biological networks." In: *Proceedings of the National Academy of Sciences of the United States of America* 99.12 (2002), pp. 7821–7826.

[GSC00] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to Forget: Continual Prediction with LSTM." In: *Neural Computation* 12.10 (2000), pp. 2451–2471.

[Haw+19] Jeff Hawkins, Subutai Ahmad, Scott Purdy, and Alexander Lavin. "Biological and Machine Intelligence (BAMI)." 2019.

[Haz+18] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective." In: *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2018, pp. 620–629.

[Heb49] Donald Olding Hebb. *The Organization of Behavior: A Neuropsychological Theory*. 1949.

[Hop82] John Joseph Hopfield. "Neural networks and physical systems with emergent collective computational abilities." In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory." In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. 2nd ed. 2009.

[HW79] J. A. Hartigan and M. A. Wong. "Algorithm AS136 - A K-Means Clustering Algorithm." In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979), pp. 100–108.

[Jac88] Robert A. Jacobs. "Increased rates of convergence through learning rate adaptation." In: *Neural Networks* 1.4 (1988), pp. 295–307.

[Jia+19] Yihan Jiang, Jakub Konečný, Keith Rush, and Sreeram Kannan. "Improving Federated Learning Personalization via Model Agnostic Meta Learning." 2019.

[JMB05] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. "Gossip-Based Aggregation in Large Dynamic Networks." In: *ACM Transactions on Computer Systems* 23.3 (2005), pp. 219–252.

[Kam+18] Michael Kamp, Linara Adilova, Joachim Sicking, Fabian Hüger, Peter Schlicht, Tim Wirtz, and Stefan Wrobel. "Efficient Decentralized Deep Learning by Dynamic Model Averaging." In: *Machine Learning and Knowledge Discovery in Databases*. 2018, pp. 393–409.

[KB14] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A Method for Stochastic Optimization." In: *2015 International Conference on Learning Representations (ICLR 2015)*. 2014.

[Kon+16] Jakub Konečný, Hugh Brendan McMahan, Daniel Ramage, and Peter Richtarik. "Federated Optimization: Distributed Machine Learning for On-Device Intelligence." 2016.

[KT79] Daniel Kahneman and Amos Tversky. "Prospect Theory: An Analysis of Decision under Risk." In: *Econometrica* 47.2 (1979), pp. 263–291.

[Kul59] Solomon Kullback. *Information Theory and Statistics*. 1959.

[LB95] Yann LeCun and Yoshua Bengio. "Convolutional networks for images, speech, and time-series." In: *The handbook of brain theory and neural networks*. Vol. 1. 1995, pp. 255–258.

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[Let+59] Jerry Y. Lettvin, Humberto R. Maturana, Warren Sturgis McCulloch, and Walter Harry Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *Proceedings of the IRE* 47.11 (1959), pp. 1940–1951.

[Li+14] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. "Scaling Distributed Machine Learning with the Parameter Server." In: *11th USENIX Symposium on Operating Systems Design and Implementation*. 2014, pp. 583–598.

[Li+20] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. "Fair Resource Allocation in Federated Learning." In: *International Conference on Learning Representations*. 2020.

[Lin20] Grace W. Lindsay. "Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future." In: *Journal of Cognitive Neuroscience* (2020), pp. 1–15.

[Lit74] William A. Little. "The existence of persistent states in the brain." In: *Mathematical Biosciences* 19.1 (1974), pp. 101–120.

[Llo82] Stuart P. Lloyd. "Least squares quantization in PCM." In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[Lon+17] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Philip S. Yu. "Learning Multiple Tasks with Multilinear Relationship Networks." In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 1593–1602.

[Lu+17] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogerio Feris. "Fully-adaptive Feature Sharing in Multi-Task Networks with Applications in Person Attribute Classification." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 1131–1140.

[Mac67] James MacQueen. "Some methods for classification and analysis of multivariate observations." In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. 1967, pp. 281–297.

[Mar+21] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. "Federated Multi-Task Learning under a Mixture of Distributions." In: *International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with 38th International Conference on Machine Learning (FL-ICML 2021)*. 2021.

[McM+17] Hugh Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data." In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Vol. 54. Proceedings of Machine Learning Research. 2017, pp. 1273–1282.

[Mei+19] Jie Mei, Chen Chen, Jianhui Wang, and James Kirtley. "Coalitional Game Theory Based Local Power Exchange Algorithm for Networked Microgrids." In: *Applied Energy* 239 (2019), pp. 133–141.

[MP43] Warren Sturgis McCulloch and Walter Harry Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[MP69] Marvin Lee Minsky and Seymour Aubrey Papert. *Perceptrons: An Introduction to Computational Geometry*. Expanded. 1969.

[NAS18] Alex Nichol, Joshua Achiam, and John Schulman. "On First-Order Meta-Learning Algorithms." 2018.

[Nat16] US Government National Institute of Standards and Technology. *NIST Special Database 19*. 2016.

[NSH19] Milad Nasr, Reza Shokri, and Amir Houmansadr. "Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning." In: 2019, pp. 739–753.

[OHJ12] Róbert Ormándi, István Hegedűs, and Márk Jelasity. "Gossip learning with linear models on fully distributed data." In: *Concurrency and Computation: Practice and Experience* 25.4 (2012), pp. 556–571.

[Pea85] Judea Pearl. "Bayesian networks: A model of self-activated memory for evidential reasoning." In: *Proceedings of the 7th Conference of the Cognitive Science Society*. 1985, pp. 329–334.

[Ped+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830.

[PMK91] Lorien Y. Pratt, Jack Mostow, and Candace A. Kamm. "Direct Transfer of Learned Information Among Neural Networks." In: *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*. 1991, pp. 584–589.

[PY09] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning." In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2009), pp. 1345–1359.

[RA14] Shini Renjith and C Anjali. "A personalized mobile travel recommender system using hybrid algorithm." In: *2014 First International Conference on Computational Systems and Communications (ICCSC)*. 2014, pp. 12–17.

[Ran+07] Nagarajan Ranganathan, Upavan Gupta, Rashmi Shetty, and Ashok Murugavel. "An Automated Decision Support System Based on Game Theoretic Optimization for Emergency Management in Urban Environments." In: *Journal of Homeland Security and Emergency Management* 4.2 (2007).

[Rec+11] Benjamin Recht, Christopher Ré, Stephen J. Wright, and Feng Niu. "Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent." In: *Advances in Neural Information Processing Systems 24*. 2011, pp. 693–701.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Internal Representations by Error Propagation." In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1. 1986, pp. 318–362.

[Roc+56] Nathaniel Rochester, J. H. Holland, L. H. Haibt, and W. L. Duda. "Tests on a cell assembly theory of the action of the brain, using a large digital computer." In: *IRE Transactions on Information Theory* 2.3 (1956), pp. 80–93.

[Rok05] Lior Rokach. "Clustering methods." In: *Data mining and knowledge discovery handbook*. 2nd ed. 2005, pp. 269–298.

[Ros58] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408.

[Rud17] Sebastian Ruder. "An Overview of Multi-Task Learning in Deep Neural Networks." 2017.

[Saa+10] Walid Saad, Zhu Han, Are Hjorungnes, Dusit Niyato, and Ekram Hossain. "Coalition Formation Games for Distributed Cooperation Among Roadside Units in Vehicular Networks." In: *IEEE Journal on Selected Areas in Communications* 29.1 (2010), pp. 48–60.

[Sae+17] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P. Sheth. "Machine learning for internet of things data analysis: a survey." In: *Digital Communications and Networks* 4.3 (2017), pp. 161–175.

[Sch+18] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. "Progress & Compress: A scalable framework for continual learning." In: *Proceedings of the 35th International Conference on Machine Learning (PMLR)*. Vol. 80. Proceedings of Machine Learning Research. 2018, pp. 4528–4537.

[Shi+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. "Edge Computing: Vision and Challenges." In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646.

[Smi+16] Virginia Smith, Simone Forte, Chenxin Ma, Martin Takáč, Michael Jordan, and Martin Jaggi. "CoCoA: A General Framework for Communication-Efficient Distributed Optimization." In: *Journal of Machine Learning Research* 18 (2016).

[Smi+17] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. "Federated Multi-Task Learning." In: *Advances in Neural Information Processing Systems 30*. 2017, pp. 4424–4434.

[Smo86] Paul Smolensky. "Information Processing in Dynamical Systems: Foundations of Harmony Theory." In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1. 1986, pp. 194–281.

[SMS20] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. "Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization Under Privacy Constraints." In: *IEEE Transactions on Neural Networks and Learning Systems* 32.8 (2020), pp. 3710–3722.

[Ste56] Hugo Steinhaus. "Sur la division des corps matériels en parties." In: *Bulletin de l'Académie Polonaise des Sciences*. Vol. 4. 12. 1956, pp. 801–804.

[SvW19] Gjorgji Strezoski, Nanne van Noord, and Marcel Worring. "Learning Task Relatedness in Multi-Task Learning for Images in Context." In: ICMR '19. 2019, pp. 78–86.

[Tan+19] Wei Tang, Yang Yang, Lanling Zeng, and Yongzhao Zhan. "Size Constrained Clustering With MILP Formulation." In: *IEEE Access*. Vol. 8. 2019, pp. 1587–1599.

[VC71] Vladimir Naumovich Vapnik and Alexey Yakovlevich Chervonenkis. "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities (О равномерной сходимости частот появления событий к их вероятностям)." In: *Theory of Probability & Its Applications (Теория вероятностей и ее применения)*. Vol. 16. 2. 1971, pp. 264–280.

[vM44] John von Neumann and Oskar Morgenstern. *Theory Of Games And Economic Behavior*. 1944.

[Wer74] Paul John Werbos. "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science." PhD thesis. Harvard University, 1974.

[WKS16] Jialei Wang, Mladen Kolar, and Nathan Srerbo. "Distributed Multi-Task Learning." In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Vol. 51. Proceedings of Machine Learning Research. 2016, pp. 751–760.

[WL90] Bernard Widrow and Michael A. Lehr. "30 years of adaptive neural networks: perceptron, Madaline, and backpropagation." In: *Proceedings of the IEEE* 78.9 (1990), pp. 1415–1442.

[ZBT19] Valentina Zantedeschi, Aurélien Bellet, and Marc Tommasi. "Fully Decentralized Joint Learning of Personalized Models and Collaboration Graphs." 2019.

[ZCY17] Tian Zhang, Wei Chen, and Feng Yang. “Data offloading in mobile edge computing: A coalitional game based pricing approach.” In: *IEEE Access* 6 (2017), pp. 2760–2767.

UNIVERSITÉ DE RENNES 1

Titre : Contributions à l'apprentissage machine distribué multitâche

Mot clés : Distribué, Décentralisé, Fédéré, Apprentissage machine, Multitâche, Réseaux de neurones

Résumé : L'apprentissage machine est un des domaines les plus importants et les plus actifs dans l'informatique moderne. La plupart des systèmes d'apprentissage machine actuels utilisent encore une architecture essentiellement centralisée. Même si l'application finale doit être délivrée sur de nombreux systèmes, parfois des millions (voire des milliards) d'appareils individuels, le processus d'apprentissage est toujours centralisé dans un centre de calcul. Ce peut être un problème notamment si les données d'apprentissage sont sensibles, comme des conversations privées, des historiques de recherche ou des données médicales.

Dans cette thèse, nous nous intéressons au problème de l'apprentissage machine distribué dans sa forme multitâche : une situation dans laquelle différents utilisateurs d'un même système d'apprentissage machine ont des tâches similaires, mais différentes, à apprendre, ce qui correspond à des applications majeures de l'apprentissage machine moderne, comme la reconnaissance de l'écriture ou de la parole.

Nous proposons tout d'abord le concept d'un système d'apprentissage machine distribué multitâche pour les réseaux de neurones. Ensuite, nous proposons une méthode permettant d'optimiser automatiquement le processus d'apprentissage en identifiant les tâches les plus similaires. Enfin, nous étudions comment nos propositions correspondent aux intérêts individuels des utilisateurs.

Title: Contributions to distributed multi-task machine learning

Keywords: Distributed, Decentralized, Federated, Machine learning, Multi-task, Neural networks

Abstract: Machine learning is one of the most important and active fields in present computer science. Currently, most machine learning systems are still using a mainly centralized design. Even when the final application is to be delivered in several systems, potentially millions (and even billions) of personal devices, the learning process is still centralized in a large datacenter. This can be an issue if the training data is sensitive, like private conversations, browsing histories, or health-related data.

In this thesis, we tackle the problem of distributed machine learning in its multi-task form: a situation where different users of a common machine learning system have similar but different tasks to learn, which corresponds to major modern applications of machine learning, such as handwriting recognition or speech recognition.

We start by proposing a design of an effective distributed multi-task machine learning system for neural networks. We then propose a method to automatically optimize the learning process based on which tasks are more similar than others. Finally, we study how our propositions fit the individual interests of users.