MAPL

Model Agnostic Peer-to-Peer Learning

Sayak Mukherjee

MAPL

Model Agnostic Peer-to-Peer Learning

by

Sayak Mukherjee

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday August 31, 2023 at 13:00.

Student number:	5222273	
Project duration:	October 28, 2022 – August 31, 2023	
Thesis committee:	Dr. ir. H. Jamali-Rad,	TU Delft and Shell, Daily supervisor
	Dr. ir. J. van Gemert,	TU Delft, Advisor
	Prof. dr. ir. G. Leus,	TU Delft, External committee member
	Prof. dr. ir. A. Simonetto,	ENSTA Paris, External Daily co-supervisor

TU Delft

Preface

This thesis report presents a culmination of my work towards obtaining the degree of Master of Science in Computer Science, in the Artificial Intelligence track at the Delft University of Technology. The research was conducted within the Computer Vision Lab at TU Delft under the supervision of Dr. Jan van Gemert and the daily supervision of Dr. Hadi Jamali-Rad and Prof. Dr. Andrea Simonetto from ENSTA Paris.

This report has been structured first to provide an introduction to the problem being addressed, followed by a scientific article containing the motivation, related work, methodology, experiments and results. The following chapters cover the necessary background supporting the scientific paper.

First, I would like to acknowledge my supervisor, Dr. Hadi Jamali-Rad, for giving me the opportunity to pursue my master's thesis on the exciting topic of decentralized learning. He has been a true mentor to me. I would also like to thank my supervisor Prof. Dr. Andrea Simonetto, for guiding me throughout the thesis. They have constantly supported me through the project by providing valuable feedback and motivation whenever needed. Finally, I would like to thank my thesis committee chair, Dr. Jan van Gemert, for his interesting Deep Learning and Computer Vision classes that further fostered my academic interest in computer vision. I would also like to thank Prof. Dr. Geert Leus for evaluating my work by participating in my thesis committee.

Lastly, I would like to thank my family and friends for their constant support and motivation, without which I would not have been able to complete this thesis.

Sayak Mukherjee
Delft, August 2023

Contents

1 Introduction

Large deep-learning models require a substantial amount of training data. However, with the rise of Internet-of-Things (IoT), data mostly resides on edge devices, making it difficult and privacy-sensitive to collect all the data. A privacy-preserving learning technique called Federated Averaging (FedAvg) [27] was proposed to address this. FedAvg trains a global model by aggregating locally trained model parameters, allowing collaborative learning without sharing local data. FedAvg has shown good performance when the local data across clients are independent and identically distributed (IID). However, in real-world scenarios, data distributions across clients are not identical, and challenges such as feature distribution drift and label distribution skew make learning a single global model challenging.

To overcome these limitations, Personalized Federated Learning (PFL) focuses on creating optimal local models tailored to client-specific objectives instead of relying on a single global model. Clustered Federated Learning (CFL) addresses personalization by clustering clients with similar tasks or underlying data distribution [24]. Although most of the work on CFL relies on model weights or gradients, some approaches utilize prototypes or centroids of feature embeddings to infer task similarity between clients in a collaborative setting [20]. Moreover, some PFL approaches directly utilize such prototypes or centroids for striking a balance between generalization and personalization without the need for clustering the clients [35, 32, 38].

Besides data heterogeneity, FL heavily relies on a central server facilitating the collaboration between all the clients, which often results in being the computational bottleneck in each global training round. Although some asynchronous FL methods attempt to address this concern, communicating with the central server is also a bottleneck due to the limited bandwidth. Furthermore, the central server is considered a trustworthy node to allow for sharing information required for aggregation, making it more prone to malicious attacks that can jeopardize the entire learning framework. Thus, decentralized FL (DFL) or peer-to-peer learning based on the concepts of decentralized learning has gained attention as a viable alternative to centralized FL, addressing computational bottlenecks and privacy concerns, alleviating the need for a central server [11, 23, 13, 39]. Decentralized learning has been extensively studied to compute the mean over multiple distributed sensors [4]. In such a setting, each client communicates with their neighbours using gossip communication to reach a consensus. Decentralized (sub)gradient descents (DGD) is another decentralized algorithm that does not rely on gossip-based consensus [33, 2]. Such decentralized algorithms only communicate with their immediate neighbours and have been shown to be useful for distributed non-convex optimization, even outperforming their centralized counterparts in some scenarios [25]. Recently DGD has received significant attention from academia, where the impact of network topology and convergence rates of such algorithms have been explored [19, 18, 36]. Existing works on current centralized FL methods state that it can be adapted to the peer-to-peer setting using multiple rounds of gossip averaging to average model parameters overall clients in the absence of a central server. However, such a setup often comes with a significant communication cost which only grows with the size of the network. Thus, there is a pressing need for further research on developing methods tailored for the peer-to-peer setting.

A key assumption in most FL or decentralized FL frameworks is that the clients share the same model architecture or an architecture derived from a common model, thereby ensuring a common subset of model parameters. Hence the network parameters can be aggregated. However, the assumption might not always be valid as each edge device may vary on computation capabilities and might choose an architecture to cater for their local data distribution, making collaboration even more challenging. Heterogeneous FL (HFL) approaches tackle this by utilizing knowledge distillation, HyperNetworks [41], or prototype-based methods. However, all these methods have severe drawbacks. Knowledge distillation, HyperNetworks-based methods would incur considerable communication costs when adapted to peer-to-peer settings making it impractical for real-world applications. On the other hand, prototype-based methods would lead to privacy violations, requiring sharing class information with neighbours without a trusted central server in the peer-to-peer setting. Furthermore, to my knowledge, model heterogeneity has been studied only in the centralized setting, although it is an important practical consideration.

Thus through this work, I aim to develop a method for collaboration among heterogeneous clients in a peer-to-peer setting. It is not a trivial problem, as adapting the existing centralized methods would result in significant communication costs or privacy concerns, as highlighted above. Specifically, I address the following two research questions by proposing a **Model Agnostic Peer-to-peer Learning (MAPL)** framework for joint learning of personalized model and collaboration weights in a model heterogeneous peer-to-peer setting:

- **RQ1**. *How to **collaborate** in a privacy-aware manner?*

  To address this, I resorted to using prototype-based feature alignment among the clients inspired by the effectiveness of such approaches in the centralized setting with heterogeneous models [29, 35]. However, in the absence of a trusted central server, sharing features or feature aggregates with the neighbours as prototypes could violate privacy. Thus, I consider learnable prototypes instead. Furthermore, I used a uniformity loss to enforce maximum separability among the prototypes and a supervised contrastive loss for improved class separation at the local client level. Therefore, besides being privacy-aware, the prototypes also effectively avoid overfitting to small client datasets.

- **RQ2**. *How to **identify neighbours** with whom to collaborate for an optimal personalized model?*

  Earlier studies ([26, 34]) showed that the classifier layer has a more significant bias towards the local data heterogeneity compared to the other layers of a deep neural network. Thus, it can effectively infer task similarity among the client's local data distribution. In my work, I formulate the graph learning problem as an optimization problem using the cosine similarity among the locally trained classifier layers and a graph regularizer controlling the sparsity of the learned collaboration graph. Such an approach is both privacy-preserving and does not result in additional communication overheads.

In the following chapters, first, I present a scientific paper on the proposed method in Chapter 2, which I aim to submit at the *twelfth International Conference on Learning Representations (ICLR 2024)*. Following this, I present an overview of deep learning, self-supervised learning and decentralized machine learning in Chapters 3, 4 and 5, respectively as additional background material for the paper. Finally, in Chapter 6, I end with concluding remarks and future research directions as a direct outcome of my thesis.

2

Scientific Paper

MAPL: MODEL AGNOSTIC PEER-TO-PEER LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Current methods in Federated and Decentralized learning presume that all clients share the same model architecture, assuming model homogeneity. However, in practice, this assumption may not always hold due to hardware differences. While prior research has addressed model heterogeneity in Federated Learning, it remains unexplored in fully decentralized or peer-to-peer settings. Therefore, in this paper, we investigate a real-world yet challenging situation involving model heterogeneity in a fully decentralized context. Furthermore, we introduced a **M**odel **A**gnostic **P**eer-to-peer **L**earning (`MAPL`) framework which allows learning of heterogeneous personalized models. Additionally, we define a graph learning objective to learn a sparse collaboration graph based on task similarity. Experiments reveal that even in this challenging scenario, `MAPL` delivers competitive results while being communication efficient in both model homogeneous and heterogeneous settings.

1 INTRODUCTION

Deep-learning models rely on massive amounts of training data for their success. In the real world, most data resides on edge devices distributed across the globe. To avoid the cost of communication and honour the privacy of local devices, there has been a growing interest in research on *decentralized learning*. The two most prevalent paradigms are *Federated learning* (FL) McMahan et al. (2017) and fully Decentralized Learning or peer-to-peer learning (P2P) Sun et al. (2022) where a global model is learned by aggregating several locally trained models on distributed data sources without communicating the local data. While both alternate between local updates and parameter aggregation across clients, the key difference is that FL considers the presence of a central server and a global model, whereas, in P2P learning, there is no server to facilitate the collaboration. However, in FL, the central server that is responsible for facilitating collaboration among the clients acts as a *communication and computation bottleneck* as the number of clients grows. Thus, P2P learning is more suitable for decentralized learning at scale. Several studies have proved that under certain conditions, P2P learning convergences to a *consensus* model Hendrikx et al. (2020); Hendrikx (2022); Vogels et al. (2022) even outperforming its centralized counterparts Lian et al. (2017).

Besides being large-scale, data is *heterogeneous* across the distributed data sources. It poses a serious threat as it can have detrimental impacts on downstream performance. To overcome the negative effects of heterogeneity, *Personalized Federated Learning (PFL)* collaboratively learns local models tailored to client-specific objectives instead of relying on a single global model. *Clustered Federated Learning (CFL)* address personalization by clustering clients with similar tasks or underlying data distribution Li et al. (2022b) by comparing model parameters, local gradients, or centroids of feature embeddings Jamali-Rad et al. (2022). Other methods of PFL utilize the feature centroids to strike a balance between generalization and personalization Tan et al. (2021); Regatti et al. (2022); Xu et al. (2023). In the P2P setting, personalized models are learned over the collaboration network either by using a pre-defined similarity graph Vanhaesebrouck et al. (2016) or, better still, by learning the collaboration graph along with personalized models. Analogous to CFL, the collaboration graph is inferred based on model weights Zantedeschi et al. (2020), local gradients Li et al. (2022a) or by knowledge distillation Jeong and Kountouris (2023).

Another key challenge of learning across decentralized devices is the presence of different model architectures as a direct consequence of hardware heterogeneity. Model averaging-based approaches operate based on the assumption that the clients share the same model architecture or that each client architecture is part of a larger client model Horváth et al. (2021). Thus, such methods are not applicable in *model heterogeneous* settings. Knowledge distillation-based approaches Zhang et al. (2021) can be used, although the presence of a publicly shared dataset is necessary. Furthermore, such a shared dataset might not be effective for the local objective. Yet another approach involving an additional HyperNetwork Shamsian et al. (2021) incur considerable communication costs, making it impractical for real-world applications. Lastly, prototype-based methods of collaboration can be viable in heterogeneous settings. However, a key challenge is to share informative class-wise representations without compromising the privacy of local clients. Existing methods Mu et al. (2021); Tan et al. (2021) share class-wise feature aggregates with a trusted central server in an FL setting. In the absence of such a trusted agent to facilitate collaboration in a peer-to-peer setting, directly sharing feature aggregates might result in information leaks under specialized attacks from adversarial clients in the network. Nonetheless, despite model heterogeneity being common in many real-world applications, it has only been studied in the FL setting. Thus, we put forth the following question:

How to collaborate among heterogeneous clients in a fully decentralized setting?

In an attempt to address this question, we introduce **M**odel **A**gnostic **P**eer-to-peer **L**earning (coined as `MAPL`). It jointly learns *personalized* model and *collaboration graph weights* in a *model heterogeneous peer-to-peer* setting. At the local level, `MAPL` utilizes inter-client and intra-client contrastive loss to learn separable class boundaries while avoiding overfitting by leveraging information from neighbours. While at the network level, `MAPL` learns the collaboration graph based on local task similarities. We further consider a graph regularizer to control the sparsity of the learned collaboration network. Our contributions can be summarized as follows:

- We propose `MAPL` to simultaneously learn *heterogeneous personalized* models in a *fully decentralized setting.*
- We design a graph learning problem to infer the optimal weights of the collaboration based on the similarity of the local dataset in a privacy-aware manner.
- Through experimentation, we show that even in a practically relevant yet challenging setting, `MAPL` can outperform existing state-of-the-art model heterogeneous FL methods (up to 2%) while being fully decentralized.

2 Related Work

Personalized Federated Learning. Federated Learning (FL) aims to enable collaborative and privacy-preserving cooperation among multiple clients. An early notable approach in this realm is FedAvg (McMahan et al., 2017), which involves alternating between local gradient descent steps and global aggregation on a central server for collaborative learning of deep learning models. However, subsequent research has revealed that FedAvg's performance suffers due to data heterogeneity (Hsieh et al., 2019). Several methods have been proposed since to address such shortcomings. FedProx (Li et al., 2018), MOON Li et al. (2021a) and SCAFFOLD (Karimireddy et al., 2019). Nonetheless, employing methods to train a single global model always involves a trade-off between local and global performance. Recent research has thus increasingly focused on personalized Federated Learning (pFL), aiming to enhance local performance through collaboration rather than training a single global model. Methods such as Ditto (Li et al., 2021b) and pFedMe (Dinh et al., 2020) optimize a global and a regularized local objective. Clustering based methods such as CFL (Li et al., 2022b) and FLT (Jamali-Rad et al., 2022) identify clients with similar data distribution and perform model aggregation within the identified clusters to facilitate personalization. A contemporary work WeiAvg (Dong et al., 2023) assigns weights to clients based on local data diversity using projection of the local model updates. Another method pFedSim (Tan et al., 2023) compute weights based on similarity of the classifier weights after performing several rounds of FedAvg. Per-FedAvg (Fallah et al., 2020) adopts a meta-learning approach to learn a good initialization for local models. FedPer (Arivazhagan et al., 2019) and FedRep (Collins et al., 2021) decouple deep neural networks into shared representation and personalized classification layers. FedProc (Mu et al., 2021) employ a globally shared set of prototypes to align the representation learning across clients. However, all these methods assume uniform client model architecture (McMahan et al., 2017) or submodels pruned from a larger server model. Worst still, some methods consider dropping the lower tier devices resulting in training bias (Kairouz et al., 2019).

Model Heterogeneous Federated Learning. Heterogeneous Federated Learning addresses both system and data heterogeneity. Methods such as pFedHN (Shamsian et al., 2021) and FedRoD (Chen and Chao, 2021) introduce a hypernetwork for personalization incuring additional computation overheads. Others utilize knowledge distillation based on a globally shared (Zhang et al., 2021) or generated datasets (Zhu et al., 2021), requiring additional parameters and dataset sharing with communication overheads. However, distillation datasets might not match clients' local objectives and the impact of using such datasets is yet to be fully explored. Therefore, FedProto (Tan et al., 2021) relies on using only class-wise feature aggregates to align the learned representations across clients. In a recent work Regatti et al. (2022) proposed aggregation of classifier layers in addition to class-wise feature centroids to improve the generalization performance of the clients. Conversely, FedClassAvg (Jang et al., 2022) showed that sharing a classifier layer suffices for representation alignment and improved generalization.

Decentralized Learning. FL relies heavily on a central server for facilitating collaboration among clients, posing communication and computation bottlenecks, particularly in the presence of a large number of clients. Moreover, the central server is always susceptible to malicious attacks as it is a single point of failure. To address this, decentralized or peer-to-peer (P2P) learning models have emerged, where clients communicate only with immediate neighbours. Early work combined gossip averaging (Xiao et al., 2007) and Stochastic Gradient Descent to obtain a global "consensus model", assuming specific topologies like doubly stochastic weights (Lian et al., 2017; Jiang et al., 2017). However, as the proposed methods were aimed at learning a single model for all clients, and hence there is no scope for personalization. Vanhaesebrouck et al. (2016) proposed a decentralized method to learn personalized models on the graph given a similarity graph as an input. It was later built on by Zantedeschi et al. (2020), who proposed a joint objective for learning both personalized ensemble models and a collaboration graph using a graph regularization term controlling the sparsity of the learned graph. The learned collaboration assigns more weight to edges of the graph such that the clients connected by the edge can jointly optimize their local objectives. Jeong and Kountouris (2023) also proposed a similar joint learning object but inferred the similarity of local tasks using knowledge co-distillation. Nonetheless, to our knowledge, the model heterogeneous setting has not been studied in a fully decentralized setting.

3 Problem Setting

Consider a setting with M clients. Each client i has an individual K-class classification task and its own data distribution $\mathcal{D}_i$ on $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{X} = \mathbb{R}^d$ is the input space of dimension d and $\mathcal{Y} := \{1, .., K\}$ is the output space. Thus, $y_i \in \mathcal{Y}$ is the label associated with the sample $x_i \in \mathcal{X}$. In a non-independently and identically distributed (non-IID) setting, the data distribution is different for every client, i.e. $\mathcal{D}_i \neq \mathcal{D}_j, \forall i, j \in [M]$. Each client i has access to $n_i = \sum_{k=1}^{K} n_i^{(k)}$ data points sampled from $\mathcal{D}_i$, with $n_i^{(k)}$ the number of samples belonging to class k. Additionally, client i has a model $h_{\Theta_i} : \mathcal{X} \to \mathcal{Y}$ in a hypothesis class $\mathcal{H}$ with parameters Θ_i that maps samples from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. Given a local loss function $\mathcal{L} : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, clients attempt to learn the network parameters such that the loss on the local dataset is minimized.

Model Heterogeneity. We consider a scenario where each client model might differ from every other, i.e. $h_{\Theta_i} \neq h_{\Theta_j}, \forall i, j \in [M]$. This can occur in real-world scenarios owing to the difference in hardware capabilities of the edge devices. Without loss of generality, the client model h_{Θ_i} can be further decoupled into a feature extractor and classifier head. The feature extractor $f_{\theta_i} : \mathcal{X} \to \mathcal{Z}$ is a learnable network parameterized by θ_i which maps samples from the input space $\mathcal{X}$ to the latent space $\mathcal{Z} = \mathbb{R}^{d_z}$. The classifier head $g_{\phi_i} : \mathcal{Z} \to \mathcal{Y}$ is also a learnable network parameterized by ϕ_i, which maps features from the latent space $\mathcal{Z}$ to the output space $\mathcal{Y}$. Therefore, a client model can be written as $h_{\Theta_i} = f_{\theta_i} \circ g_{\phi_i}$. Considering the decoupled view, we assume the feature extractor f_{θ_i} is different for every client, but the dimension of the latent space is constant across all clients.

Peer-to-peer Learning. Due to insufficient data samples available locally, the trained model is prone to overfitting and poor generalization performance. Thus, we aim at addressing this in a fully decentralized or peer-to-peer setting where each client can collaborate with any other client by sending and receiving messages without reliance on a central server. However, to ensure scalability, we consider a *collaboration graph* $\mathcal{G} = \{\boldsymbol{v}, \boldsymbol{W}\}$ such that the interaction among clients is restricted within users that share similar tasks. The graph $\mathcal{G}$ is an undirected weighted graph where the vertices

$\boldsymbol{v}$ $(= [M])$ correspond to clients in the network. The edge weight matrix $\boldsymbol{W} \in \mathbb{R}^{M\times M}$ consists of stacked row vectors $\boldsymbol{w}_i \in \mathbb{R}^{1\times M}$ corresponding to the local weight vector for each client i. Each element w_{ij} reflects the similarity among the local tasks between client i and j, with $w_{ij} = 0$ denoting the absence of an edge. A client i collaborates solely with its direct neighbours $\mathcal{N}_i = \{j|w_{ij} > 0\}$ in $\mathcal{G}$.

Objective. To simultaneously learn local personalized models $\{h_{\Theta_i}\}_{i=1}^M$ and graph weights $\boldsymbol{W}$, the joint optimization problem can be formulated as:

$$\underset{\{\Theta_i\}_{i=1}^M, \boldsymbol{W}}{\text{minimize}} \sum_{i=1}^{M} \Big(\mathbb{E}_{(x,y)\sim\mathcal{D}_i} \mathcal{L}_i(x, y; h_{\Theta_i}) + \mathcal{R}(h_{\Theta_i}, \Omega) - \sum_{j=1}^{M} \mu_1 \gamma_j \, w_{ij} \, s_{ij} + \mu_2 \, \mathcal{R}_g(\boldsymbol{w}_i) \Big), \qquad (1)$$

where γ_j is a confidence factor associated with the client j based on the size of the local dataset $|\mathcal{D}_j|$ and $\mathcal{R}(h_{\Theta_i}, \Omega)$ is a proposed regularization term to prevent h_{Θ_i} from overfitting by relating all the clients using global information Ω. The similarity s_{ij} between the local tasks of client i and j and $\mathcal{R}_g(\boldsymbol{w}_i)$ is a graph regularization term which controls the sparsity of the collaboration graph $\mathcal{G}$. Additionally, μ_1 and μ_2 are two hyperparameters that control the trade-off between task similarity and graph regularization. We will further elaborate on our choices of $\mathcal{R}$, Ω in Section 4.1 and about $\mathcal{R}_g$ in Section 4.2. As the true underlying distribution $\mathcal{D}_i$, we instead attempt to minimize the empirical loss, $\hat{\mathcal{L}}_i$, by re-writing Eq. 1 as:

$$\underset{\{\Theta_i\}_{i=1}^M, \boldsymbol{W}}{\text{minimize}} \sum_{i=1}^{M} \Big(\hat{\mathcal{L}}_i(h_{\Theta_i}) + \mathcal{R}(h_{\Theta_i}, \Omega) - \sum_{j=1}^{M} \mu_1 \, \gamma_j w_{ij} \, s_{i,j} + \mu_2 \, \mathcal{R}_g(\boldsymbol{w}_i) \Big), \qquad (2)$$

where $\hat{\mathcal{L}}_i(h_{\Theta_i}) = 1/n_i \sum_{q=1}^{n_i} \mathcal{L}_i(x_q, y_q; h_{\Theta_i})$ is the empirical loss over the client's private dataset.

4 METHODOLOGY

In this section, we propose **M**odel **A**gnostic **P**eer-to-Peer **L**earning (`MAPL`), a novel method for learning *heterogeneous personalized* models in a *decentralized* setting while simultaneously *identifying neighbours* (with similar tasks) for effective collaboration. To this aim, `MAPL` has to solve the optimization problem formulated in Eq. 2. Thus, we propose an alternating optimization approach where we: (i) apply a novel peer-to-peer collaborative learning step given a fixed collaboration graph; (ii) next, learn the collaboration graph given fixed client models.

4.1 PEER-TO-PEER LEARNING OF PERSONALIZED MODELS

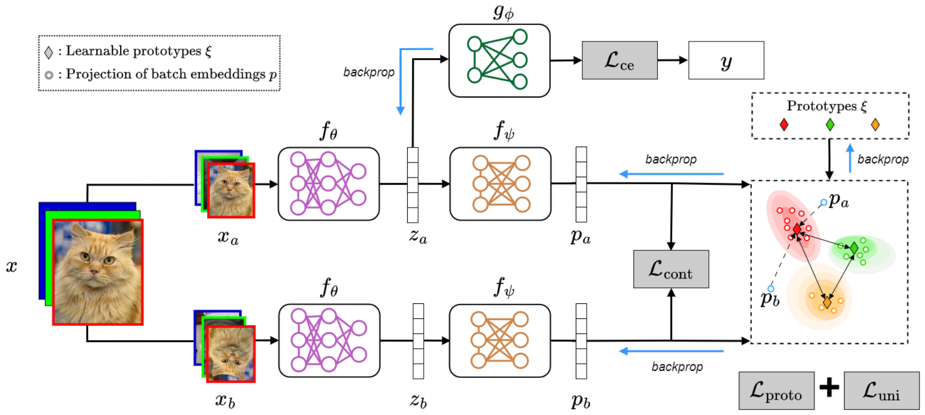

◊ : Learnable prototypes ξ
○ : Projection of batch embeddings p
backprop
g_ϕ
$\mathcal{L}_{\text{ce}}$
y
Prototypes ξ
f_θ
f_ψ
backprop
backprop
p_a
x_a
z_a
p_a
x
$\mathcal{L}_{\text{cont}}$
p_b
f_θ
f_ψ
backprop
x_b
z_b
p_b
$\mathcal{L}_{\text{proto}}$ + $\mathcal{L}_{\text{uni}}$

Figure 1: `MAPL`: A local perspective. Illustration of the local training methodology.

There are two main goals that we attempt to address through our local objective: (i) to learn personalized models that can minimize the local objective (first term in Eq. 2); (ii) to improve generalization by leveraging information from neighbours Ω (second term in Eq. 2).

```
Algorithm 1: MAPL
Require: $M$, $\{\Theta_i, f_{\theta_i}, f_{\psi_i}, g_{\phi_i}, \xi^{(i)}\}_{i=1}^{M}$, $\boldsymbol{w_i}$, $E$,
         $T$, $T_{\text{thr}}$, $\eta$, $\gamma$
1  for ($t \in [T]$) and ($i \in [M]$) in parallel do
       /* update local models */
2      for (epoch $\in [E]$) and ($(x, y) \in \mathcal{D}_i$) do
3          $x_a, x_b \leftarrow \zeta_a(x), \zeta_b(x)$ for $\zeta_a, \zeta_b \in \mathcal{A}$
4          $z_a, z_b \leftarrow f_{\theta_i}(x_a), f_{\theta_i}(x_b)$
5          $p_a, p_b \leftarrow f_{\psi_i}(z_a), f_{\psi_i}(z_b)$
6          Compute $\mathcal{L}_{\text{ce}}$ on $g_{\phi_i}(z_a)$, $g_{\phi_i}(z_b)$, $y$
7          Compute local loss (Eq. 3, 4, 5):
8              $\mathcal{L} = \mathcal{L}_{\text{cont}} + \mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{proto}} + \mathcal{L}_{\text{uni}}$
9          Update local model:
10             $\Theta_i \leftarrow \Theta_i - \eta \nabla_{\Theta_i} \mathcal{L}$
11         Update local prototypes:
12             $\xi^{(i)} \leftarrow \xi^{(i)} - \eta \nabla_{\xi^{(i)}} \mathcal{L}$
13     end
       /* collaboration graph */
14     if $t \geq \mathrm{T}_{\text{thr}}$ then $\boldsymbol{w}_i \leftarrow$ CGL(.) end
       Aggregate. prototypes: $\xi^{(i)} \leftarrow \xi^{(i)} \times \boldsymbol{w}_i$
15 end
Return: $\{\Theta_i\}_{i=1}^{M}$
```

To address our first goal, we argue that training solely with a cross-entropy loss would be suboptimal in a non-IID setting, where clients lack a balanced distribution of data across classes. Instead, we resort to a contrastive learning approach inspired by its recent success in decentralized settings Jang et al. (2022); Wang et al. (2023). Let $\zeta_a, \zeta_b \sim \mathcal{A}$ be two randomly sampled augmentations from a set of data augmentations $\mathcal{A}$. At client i, a data sample $x \sim \mathcal{D}_i$ is transformed into two views $x_a = \zeta_a(x)$, and $x_b = \zeta_b(x)$. As illustrated in Fig. 1, we adopt a symmetric Siamese architecture similar to Chen et al. (2020). Next, feature maps $z_a = f_{\theta_i}(x_a)$, and $z_b = f_{\theta_i}(x_b)$ are obtained by passing both the views through the feature extraction backbone f_{θ_i}. The extracted feature maps are then passed through a projection network f_{ψ_i} to obtain the projections $p_a = f_{\psi_i}(z_a)$, and $p_b = f_{\psi_i}(z_b)$ (line $3-5$, Algorithm 1). We adopt sample-to-sample supervised contrastive loss ($\mathcal{L}_{\text{cont}}$) (Khosla et al., 2020; Jang et al., 2022) applied on the projected views. Considering $\mathcal{Q} = \{1, \cdots, 2n_i\}$ as the indexes of the two augmented views of the samples in $\mathcal{D}_i$, the supervised contrastive loss can be formulated as:

$$\mathcal{L}_{\text{cont}} = -\sum_{q \in \mathcal{Q}} \frac{1}{|\mathcal{X}^+(y_q)|} \sum_{r \in \mathcal{X}^+(y_q)} \log \frac{\exp(\cos(\hat{p}_q, \hat{p}_r)/\tau)}{\sum_{m \in \mathcal{Q} \setminus q} \exp(\cos(\hat{p}_q, \hat{p}_m)/\tau)}, \tag{3}$$

where the set of *positive samples* for label y_q (samples with the same label) is given by $\mathcal{X}^+(y_q) \equiv \{r \in \mathcal{Q} \setminus q \,|\, y_r = y_q\}$, $\hat{p}_q = p_q/\|p_q\|_2$ is the l_2 normalized projection of x_q, and τ is the temperature hyperparameter. Furthermore, in addition to the contrastive loss, we train a classifier head g_{ϕ_i} locally by applying a cross-entropy loss ($\mathcal{L}_{ce}$) on the class logits obtained from the feature maps using g_{ϕ_i} as $-\sum_{q \in \mathcal{Q}} y_q \log(g_{\phi_i}(z_q))$ (line 6, Algorithm 1). The locally trained classifier is to ensure personalization at the local level. Here both the projection head f_{ψ_i} and predictor head g_{ϕ_i} are multilayer perceptrons (MLPs).

Next, to address our second goal, we pose the question: *How do we build personalized models while benefiting from relevant neighbors' data in a model heterogeneous setting?* We cannot average model parameters anymore. Inspired by Mu et al. (2021), we craft an intra-client contrastive loss using a shared set of *learnable* class-wise prototypes $\xi = \{\xi_1, \cdots \xi_K\}$, with $\xi_k \in \mathcal{P}$. We consider a *covariate shift assumption* which considers the labelling function to be the same across all clients but the marginals to be different. Considering the local set of prototypes $\xi^{(i)}$ at client i, we apply a sample-to-prototype contrastive loss ($\mathcal{L}_{\text{proto}}$) on the l_2 normalized projections as:

$$\mathcal{L}_{\text{proto}} = -\sum_{q \in \mathcal{Q}} \log \frac{\exp(\cos(\hat{p}_q, \xi_{y_q}^{(i)})/\tau)}{\sum_{k \in [K]} \exp(\cos(\hat{p}_q, \xi_k^{(i)})/\tau)}, \tag{4}$$

where $\xi_{y_q}^{(i)}$ is the prototype for class y_q. However, a key challenge in this approach is to devise informative class-wise representations without compromising privacy. It is where our set of learnable prototypes $\xi^{(i)}$ plays a pivotal role. Starting from a random initialization, each client i learns the local set of prototypes $\xi^{(i)}$ and is aggregates them periodically as a weighted sum over the local neighbourhood as $\xi^{(i)} \leftarrow \{w_{ii}\, \xi_k^{(i)} + \sum_{j \in \mathcal{N}_i} w_{ij}\, \xi_k^{(j)}, \forall\, k \in [K]\}$ (line 15, Algorithm 1). This forces class-wise representations to be closer to a shared class prototype and thus effectively avoids overfitting to the local data distribution. Finally, to avoid degenerate solutions, we reduce the similarity among the inter-class prototypes in $\xi^{(i)}$ using a uniformity loss ($\mathcal{L}_{\text{uni}}$) as:

$$\mathcal{L}_{\text{uni}} = \sum_{k \in [K]} \sum_{r \in [K] \setminus q} \frac{\langle \xi_k^{(i)}, \xi_r^{(i)} \rangle}{\|\xi_k^{(i)}\|_2 \cdot \|\xi_r^{(i)}\|_2}. \tag{5}$$

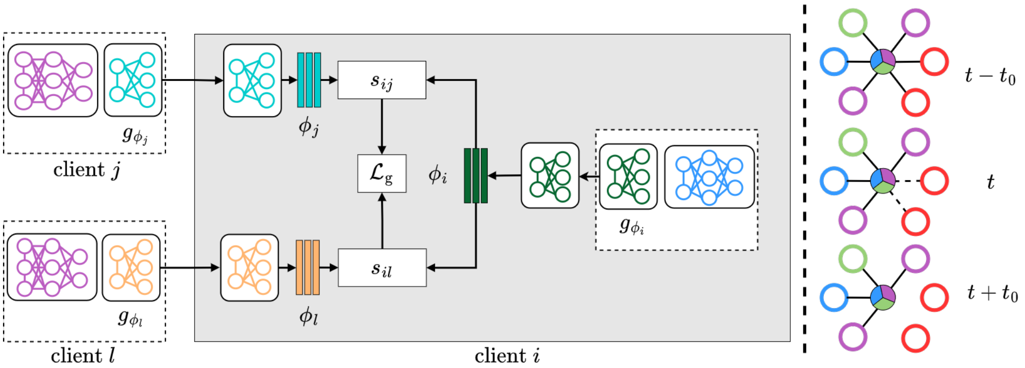


Figure 2: MAPL: A network perspective. **(Left)** Process of computing similarity between clients. **(Right)** Evolution of collaboration graph with circles representing clients and colours representing a classes.

The proposed local training objective of MAPL, as illustrated in Fig. 1, is the sum of the four losses discussed above, i.e., $\mathcal{L} = \mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cont}} + \mathcal{L}_{\text{proto}} + \mathcal{L}_{\text{uni}}$ (line 8, Algorithm 1). Clients perform the local updates for E epochs, optimizing the local objective jointly over the model parameters Θ_i (line 10, Algorithm 1) and set of prototypes $\xi^{(i)}$ (line 12, Algorithm 1) using stochastic gradient descent.

4.2 Learning the Collaboration Graph

```
Algorithm 2: CGL
Require: $\phi_i$, $\{\phi_j\}_{j=1}^{\mathcal{N}_i}$, $\mathcal{N}_i$, $\boldsymbol{w_i}$, $\mu_1$, $\mu_2$, $\eta$
1 Compute similarity with neighbours:
2     $\boldsymbol{s}_i \leftarrow s_{ij} \, \forall\, j \in \mathcal{N}_i$ (Eq. 6).
3 Compute graph learning loss:
4     $\mathcal{L}_g \leftarrow \mu_2 \, \mathcal{R}_g(\boldsymbol{w}_i) - \mu_1(\boldsymbol{w}_i \times \gamma \times \boldsymbol{s}_i)$
5 Update graph weights:
6     $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i - \eta \nabla_{\boldsymbol{w}_i} \mathcal{L}_g$
7 Project graph weights:
8     $\boldsymbol{w}_i \leftarrow \texttt{Proj}(\boldsymbol{w}_i)$
Return: $\boldsymbol{w}_i$
```

Here, we discuss our approach to **C**ollaborative **G**raph **L**earning (CGL), also summarized in Algorithm. 2. To ensure a fully decentralized setting, each client i only maintains a local weight vector $\boldsymbol{w}_i$, in contrast to existing methods that consider an overlay with a view of the entire collaboration network $\boldsymbol{W}$. The local weight vector $\boldsymbol{w}_i$ is updated iteratively based on the local task similarity and a confidence measure γ as stated earlier in Eq. 2. A key challenge, however, is inferring task similarity among the clients as the local datasets cannot be shared to honor data privacy. Recent studies have shown that the locally trained classifier head is biased toward the local data heterogeneity (Arivazhagan et al., 2019; Collins et al., 2021; Tan et al., 2023). Thus, to infer client similarity, we propose to utilize the parameters ϕ_i of the locally trained classifiers g_{ϕ_i} as shown in Fig. 2 (line 2, Algorithm. 2). Considering the classifier weight as $\phi_i = [\Phi_i^{(1)}, \cdots \Phi_i^{(K)}]$ with $\Phi_i^{(k)} \in \mathbb{R}^{d_z}, \forall k \in [K]$, task similarity is computed as the average cosine similarities of the classifier weight vectors in clients i and j as:

$$s_{ij} = \frac{1}{K} \sum_{k=1}^{K} \frac{\langle \Phi_i^{(k)}, \Phi_j^{(k)} \rangle}{\|\Phi_i^{(k)}\| \cdot \|\Phi_j^{(k)}\|}. \tag{6}$$

The optimization problem for collaboration graph weight learning at each client i can thus be formulated as:

$$\begin{aligned} \underset{\boldsymbol{w}_i}{\text{minimize}} \Big\{ \mathcal{L}_g := \mu_2 \, \mathcal{R}_g(\boldsymbol{w}_i) - \mu_1 \Big(\sum_{j \in \mathcal{N}_i} w_{ij} \, \gamma_j \, s_{ij} + w_{ii} \, \gamma_i \Big) \Big\}&, \\ \text{subject to,} \quad \boldsymbol{w}_i \succeq 0&, \\ \boldsymbol{w}_i \mathbb{1}^T = 1&, \end{aligned} \tag{7}$$

where $\mathbb{1}$ is a vector of all ones. In Eq. 7, we enforce the elements in $\boldsymbol{w}_i$ to sum up to 1, and each component to be non-negative using projected gradient descent. First, a gradient step is performed based on the graph loss $\mathcal{L}_g$ (line 4, Algorithm. 2) followed by a projection onto a unit simplex (line 7, Algorithm. 2). For the projection function $\texttt{Proj}(.)$ we follow Condat (2016, Algorithm 1).

Graph Regularization. In addition to the similarity measure in Eq. 6, we utilize a graph regularization term similar to (Zantedeschi et al., 2020) to learn a sparse graph:

$$\mathcal{R}_g(w_i) = \beta\|\boldsymbol{w}_i\|_2 + \log(d(i) + \varepsilon), \tag{8}$$

where $d(i) = \sum_{j\in\mathcal{N}_i} w_{ij}$ is the degree of the node i. The log term prevents each client from being completely isolated from others. l_2-norm along with the hyperparameter β controls the sparsity in the learned collaboration graph $\mathcal{G}$.

5 Experimentatal Evaluation

Our Goal. In this section, we address the following two questions:

Q1 How does MAPL perform against the *state-of-the-art* methods?

Q2 Does MAPL correctly identify the clusters with similar data distribution?

5.1 Implementation Details

Benchmark Dataset. We study the effectiveness of MAPL using three widely used benchmark datasets: (1) Cifar10, (2) MNIST and (3) FashionMNIST. Further details about the datasets can be found in the Appendix.

Data Heterogeneity. The data can be partitioned in several ways to simulate non-IIDness. Here, we consider label distribution skew, also often referred to as *pathological* partitioning together with quantity skew. Specifically, we group clients into C clusters, with each cluster being assigned a subset of the available classes. Then, within each cluster, clients are further allocated $n_i = 100\,(\pm 25)$ samples from each of the assigned classes. For the class assignment to each cluster, we consider two scenarios: (**Scenario 1**) clusters are allocated a disjoint subset of classes; (**Scenario 1**) clusters are allocated an overlapping subset of classes.

Training Setup. We use PyTorch for our implementations. In the heterogeneous model setup, each client i selects a feature extraction head f_{θ_i} randomly from a set of four models: GoogLeNet, ShuffleNet, ResNet18, and AlexNet. Conversely, in the homogeneous model setup, all clients use ResNet18 for feature extraction. Unless otherwise specified, our experiments consist of 1000 global communication rounds, with each client performing 1 local epoch. Further details about the implementation and hyperparameters can be found in the Appendix.

5.2 Evaluation Results

We report average test accuracies for 20 clients along with their standard deviation, using 15 test samples per allocated class for each client. We compare the performance of MAPL against a state-of-the-art model heterogeneous FL baselines (Pillutla et al., 2022; Jang et al., 2022; Tan et al., 2021). In the model homogeneous setting, we additionally compare with (McMahan et al., 2017). Lastly, in the P2P setting, we only compare against model homogeneous baseline (Sun et al., 2022), as, to the best of our knowledge, MAPL is the first attempt in addressing model heterogeneity in a fully decentralized context.

Q1-a: Model Heterogenous Setting. We present the results on Cifar10, MNIST, and FashionMNIST in Table 1. Unlike the other methods, MAPL functions through P2P communication without relying on a central server. Additionally, MAPL utilizes graph learning to optimize edge weights, resulting in a sparse collaboration graph and a substantial reduction in the number of communications. Despite the challenging conditions, MAPL delivers competitive performance, even surpassing the centralized methods on Cifar10 and MNIST by a margin of approximately $1 - 2\%$.

Q1-b: Model Homogeneous Setting. In the model heterogeneous setting, we evaluate MAPL in two different configurations. First, similar to the model heterogeneous setting, the backbone model parameters are not aggregated. Second, we introduce MAPL+, which aggregates the backbone in addition to our proposed method. The result on Cifar10, MNIST and FashionMNIST is reported in Table 2. MAPL approaches the baseline performance in both settings, while MAPL+ outperforms the baselines on all datasets, achieving up to a 7% improvement.

Table 1: Average test accuracies in (% ± std) for *heterogeneous* models on Cifar10, MNIST and FashionMNIST with `Sc 1`: denoting scenario 1 and `Sc 2`: scenario 2. Style: **best** and second best.

Method	Data Heterogeneity	Setting	Cifar10	MNIST	FashionMNIST
Local	`Sc. 1`	`Local`	59.53 (±8.41)	89.87 (±7.82)	84.33 (±6.35)
FedSim (Pillutla et al., 2022)	`Sc. 1`	`Cent.`	55.87 (±10.08)	88.2 (±13.11)	84.27 (±9.88)
FedClassAvg (Jang et al., 2022)	`Sc. 1`	`Cent.`	66.13 (±8.82)	94.73 (±3.46)	87.3 (±6.7)
FedProto (Tan et al., 2021)	`Sc. 1`	`Cent.`	60.4 (±10.31)	95.27 (±3.75)	**89.73 (±5.56)**
MAPL (Ours)	`Sc. 1`	`P2P`	**67.27 (±8.53)**	**96.07 (±2.68)**	87.33 (±5.5)
Local	`Sc. 2`	`Local`	47.62 (±6.29)	87.62 (±6.93)	77.76 (±6.53)
FedSim (Pillutla et al., 2022)	`Sc. 2`	`Cent.`	46.0 (±7.66)	85.33 (±11.15)	78.29 (±8.14)
FedClassAvg (Jang et al., 2022)	`Sc. 2`	`Cent.`	54.91 (±6.73)	92.43 (±4.01)	78.48 (±8.59)
FedProto (Tan et al., 2021)	`Sc. 2`	`Cent.`	50.76 (±6.56)	92.81 (±3.68)	**82.95 (±4.94)**
MAPL (Ours)	`Sc. 2`	`P2P`	**56.71 (±7.2)**	**93.52 (±2.99)**	82.62 (±4.92)

Table 2: Average test accuracies in (% ± std) for *homogeneous* models on Cifar10, MNIST and FashionMNIST with `Sc.1`: denoting scenario 1 and `Sc.2`: scenario 2. Style: **best** and second best.

Method	Data Heterogeneity	Setting	Cifar10	MNIST	FashionMNIST
Local	`Sc. 1`	`Local`	47.80 (±8.66)	77.88 (±15.24)	70.2 (±12.72)
FedAvg (McMahan et al., 2017)	`Sc. 1`	`Cent.`	57.06 (±15.7)	95.8 (±4.77)	88.27 (±6.76)
FedSim (Pillutla et al., 2022)	`Sc. 1`	`Cent.`	52.4 (±8.07)	87.73 (±6.13)	84.46 (±8.41)
FedClassAvg (Jang et al., 2022)	`Sc. 1`	`Cent.`	63.07 (±8.80)	93.06 (±3.76)	84.73 (±6.33)
FedProto (Tan et al., 2021)	`Sc. 1`	`Cent.`	56.46 (±9.37)	95.06 (±3.53)	87.06 (±5.25)
DFedAvgM (Sun et al., 2022)	`Sc. 1`	`P2P`	57.46 (±15.26)	95.4 (±5.39)	88.73 (±6.42)
`MAPL` (Ours)	`Sc. 1`	`P2P`	63.13 (±8.81)	94.2 (±3.89)	86.46 (±5.50)
`MAPL+` (Ours)	`Sc. 1`	`P2P`	**75.8 (±5.4)**	**97.0 (±2.34)**	**89.67 (±4.57)**
Local	`Sc. 2`	`Local`	39.12 (±5.94)	76.44 (±6.87)	66.81 (±8.64)
FedAvg (McMahan et al., 2017)	`Sc. 2`	`Cent.`	60.04 (±5.97)	94.24 (±2.91)	83.09 (±6.01)
FedSim (Pillutla et al., 2022)	`Sc. 2`	`Cent.`	44.04 (±5.94)	82.67 (±4.70)	77.66 (±6.37)
FedClassAvg (Jang et al., 2022)	`Sc. 2`	`Cent.`	50.33 (±5.40)	91.0 (±2.59)	80.62 (±5.94)
FedProto (Tan et al., 2021)	`Sc. 2`	`Cent.`	45.28 (±6.72	92.47 (±2.48)	79.19 (±6.52)
DFedAvgM (Sun et al., 2022)	`Sc. 2`	`P2P`	60.24 (±7.45)	94.52 (±2.94)	82.47 (±5.73)
`MAPL` (Ours)	`Sc. 2`	`P2P`	52.52 (±5.18)	91.33 (±3.36)	81.95 (±5.84)
`MAPL+` (Ours)	`Sc. 2`	`P2P`	**67.33 (±4.94)**	**95.09 (±3.39)**	**85.52 (±4.24)**

Q2: Visualization of learned collaboration graph. We visualize the learned weight matrix $\boldsymbol{W}$ along with data distribution over 10 clients for both scenarios in Fig. 3. It can be seen that `MAPL` identifies the true underlying clusters distinctly for scenario 1. Whereas, for scenario 2, the clusters are not distinct, which is expected. However, it is interesting to note that `MAPL` assigns relatively higher weights to clients within each cluster.

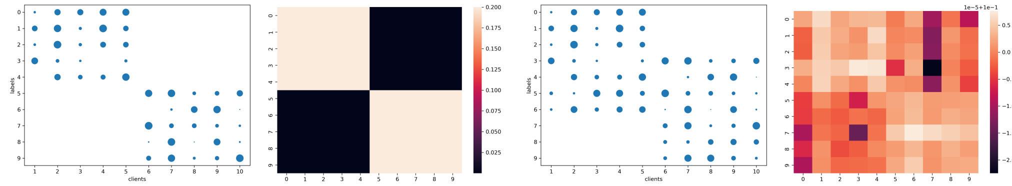
5.3 Ablation studies

In this section, we will investigate the impact of the main components of `MAPL` and evaluate performance using 10 and 20 clients in both Scenario 1 and 2. We will be considering the model heterogeneous setting with the Cifar10 dataset for all our ablation studies.

Main Component Abalation. Table 3 investigates the necessity of individual loss terms used in `MAPL` by sequentially adding each term. In all cases, the cross-entropy loss ($\mathcal{L}_{\text{ce}}$) and sample-to-prototype loss ($\mathcal{L}_{\text{proto}}$) is applied as $\mathcal{L}_{\text{ce}}$ is necessary for training the local classifier head, and the prototype-based alignment among the client facilitates the collaboration. We refer to it as the base performance and compare others with it. First, we observe that without any other loss term to guide

Table 3: Ablation of `MAPL`'s main components for Cifar10.

$\mathcal{L}_{\text{g}}$	$\mathcal{L}_{\text{cont}}$	$\mathcal{L}_{\text{uni}}$	Acc (% ± std)	
			Scenario 1	Scenario 2
-	-	-	60.67 (±8.13)	49.19 (±6.25)
-	✓	-	65.4 (±9.85)	55.66 (±9.01)
-	-	✓	61.73 (±7.12)	50.33 (±0.66)
-	✓	✓	67.2 (±8.78)	55.52 (±8.87)
✓	✓	✓	**67.27 (±8.53)**	**56.71 (±7.2)**



(a) (b) (c) (d)

Figure 3: Client data distribution and collaboration graph for Scenario 1 (Fig. a and b) and Scenario 2 (Fig. c and d). Point size in Fig. a and c represent the number of training samples.

Table 5: Performance with a varying number of clients using Cifar10.

Method	**Setting**	**Acc (% ± std)**			
		Scenario 1		**Scenario 2**	
		10 clients	20 clients	10 clients	20 clients
Local	Local	60.67 (±9.45)	59.53 (±8.41)	49.04 (±5.39)	47.62 (±6.29)
FedSim (Pillutla et al., 2022)	Cent.	57.33 (±8.98)	55.87 (±10.08)	48.0 (±7.24)	46.0 (±7.66)
FedClassAvg (Jang et al., 2022)	Cent.	65.06 (±10.21)	66.13 (±8.82)	54.28 (±5.47)	54.91 (±6.73)
FedProto (Tan et al., 2021)	Cent.	64.0 (±9.78)	60.4 (±10.31)	50.85 (±7.46)	50.76 (±6.56)
MAPL (Ours)	P2P	**66.4 (±9.25)**	**67.27 (±8.53)**	**55.71 (±8.10)**	**56.71 (±7.2)**

the learning, the performance is the lowest in both scenarios. Second, the sample-to-sample contrastive loss ($\mathcal{L}_{\text{cont}}$) is a pivotal component significantly enhancing MAPL's performance. Further, when $\mathcal{L}_{\text{cont}}$ is used in conjunction with the uniformity loss ($\mathcal{L}_{\text{uni}}$), there is a marked improvement over the base results. Finally, the introduction of the graph learning loss, which involves learning the collaboration graph weights, offers the best performance.

Effect of Similarity and Confidence on Graph Learning. Table 4 illustrates the influence of the confidence parameter (γ) and similarity parameter (s) in the graph learning objective. When used individually, their impact on performance is similar. However, their synergy produces the optimal outcome. Notably, in Scenario 2, the similarity parameter exhibits a more significant effect compared to Scenario 1. This underscores the importance of inferring similarity between client data distributions in scenarios with partial class overlaps.

Table 4: Ablation of the graph learning objective of MAPL using Cifar10.

similarity	**confidence**	**Acc (% ± std)**	
		Scenario 1	**Scenario 2**
-	✓	67.13 (±8.22)	55.23 (±8.35)
✓	-	66.93 (±8.52)	56.09 (±7.25)
✓	✓	**67.27 (±8.53)**	**56.71 (±7.2)**

Varying number of clients. We compare MAPL against the model heterogeneous baselines using 10 and 20 clients. The results as shown in Table 5, demonstrate that MAPL attains competitive performance, outperforming the baselines by a margin of approximately $1 - 2\%$.

6 Conclusion

We explored a practical yet challenging model heterogeneous scenario in a fully decentralized setting which has not been studied in the literature. To this end, we proposed MAPL to simultaneously learn heterogeneous personalized models in a fully decentralized setting. Furthermore, we formulated a graph learning objective to infer the optimal collaboration weights based on local task similarity between the clients. Through experimentation, we observed that even in this difficult setting, MAPL provides competitive results while being communication efficient owing to the sparse collaboration graph. Finally, even in the model homogeneous setting, MAPL was found to outperform both the centralized and decentralized baselines.

REFERENCES

Arivazhagan, M. G., Aggarwal, V., Singh, A. K., and Choudhary, S. (2019). Federated Learning with Personalization Layers.

Chen, H. Y. and Chao, W. L. (2021). On Bridging Generic and Personalized Federated Learning for Image Classification. *ICLR 2022 - 10th International Conference on Learning Representations*.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A Simple Framework for Contrastive Learning of Visual Representations. *37th International Conference on Machine Learning, ICML 2020*, PartF168147-3:1575–1585.

Collins, L., Hassani, H., Mokhtari, A., and Shakkottai, S. (2021). Exploiting Shared Representations for Personalized Federated Learning.

Condat, L. (2016). Fast projection onto the simplex and the l1 ball. *Mathematical Programming*, 158(1-2):575–585.

Dinh, C. T., Tran, N. H., and Nguyen, T. D. (2020). Personalized Federated Learning with Moreau Envelopes. *Advances in Neural Information Processing Systems*, 2020-December.

Dong, F., Abbasi, A., Drew, S., Leung, H., Wang, X., and Zhou, J. (2023). WeiAvg: Federated Learning Model Aggregation Promoting Data Diversity.

Fallah, A., Mokhtari, A., and Ozdaglar, A. (2020). Personalized Federated Learning with Theoretical Guarantees: A Model-Agnostic Meta-Learning Approach. *Advances in Neural Information Processing Systems*, 33:3557–3568.

Hendrikx, H. (2022). A principled framework for the design and analysis of token algorithms.

Hendrikx, H., Bach, F., and Massoulié, L. (2020). Dual-Free Stochastic Decentralized Optimization with Variance Reduction. *Advances in Neural Information Processing Systems*, 33:19455–19466.

Horváth, S., Laskaridis, S., Almeida, M., Leontiadis, I., Venieris, S. I., and Lane, N. D. (2021). FjORD: Fair and Accurate Federated Learning under heterogeneous targets with Ordered Dropout. *Advances in Neural Information Processing Systems*, 16:12876–12889.

Hsieh, K., Phanishayee, A., Mutlu, O., and Gibbons, P. B. (2019). The Non-IID Data Quagmire of Decentralized Machine Learning. *37th International Conference on Machine Learning, ICML 2020*, PartF168147-6:4337–4348.

Jamali-Rad, H., Abdizadeh, M., and Singh, A. (2022). Federated Learning With Taskonomy for Non-IID Data. *IEEE Transactions on Neural Networks and Learning Systems*.

Jang, J., Ha, H., Jung, D., and Yoon, S. (2022). FedClassAvg: Local Representation Learning for Personalized Federated Learning on Heterogeneous Neural Networks. In *Proceedings of the 51st International Conference on Parallel Processing*, pages 1–10.

Jeong, E. and Kountouris, M. (2023). Personalized Decentralized Federated Learning with Knowledge Distillation.

Jiang, Z., Balu, A., Hegde, C., and Sarkar, S. (2017). Collaborative Deep Learning in Fixed Topology Networks. *Advances in Neural Information Processing Systems*, 2017-December:5905–5915.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R. G. L., Eichner, H., Rouayheb, S. E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konečný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Raykova, M., Qi, H., Ramage, D., Raskar, R., Song, D., Song, W., Stich, S. U., Sun, Z., Suresh, A. T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F. X., Yu, H., and Zhao, S. (2019). Advances and Open Problems in Federated Learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210.

Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S. J., Stich, S. U., and Suresh, A. T. (2019). SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. *37th International Conference on Machine Learning, ICML 2020*, PartF168147-7:5088–5099.

Khosla, P., Teterwak, P., Wang, C., Sarna, A., Research, G., Tian, Y., Isola, P., Maschinot, A., Liu, C., and Krishnan, D. (2020). Supervised Contrastive Learning.

Li, Q., He, B., and Song, D. (2021a). Model-Contrastive Federated Learning. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 10708–10717.

Li, S., Zhou, T., Tian, X., and Tao, D. (2022a). Learning To Collaborate in Decentralized Learning of Personalized Models.

Li, T., Hu, S., Beirami, A., and Smith, V. (2021b). Ditto: Fair and Robust Federated Learning Through Personalization.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. (2018). Federated Optimization in Heterogeneous Networks.

Li, Z., Lu, J., Luo, S., Zhu, D., Shao, Y., Li, Y., Zhang, Z., Wang, Y., and Wu, C. (2022b). Towards Effective Clustered Federated Learning: A Peer-to-peer Framework with Adaptive Neighbor Matching. *IEEE Transactions on Big Data*, pages 1–16.

Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., and Liu, J. (2017). Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. *Advances in Neural Information Processing Systems*, 30.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and Arcas, B. A. y. (2017). Communication-Efficient Learning of Deep Networks from Decentralized Data.

Mu, X., Shen, Y., Cheng, K., Geng, X., Fu, J., Zhang, T., and Zhang, Z. (2021). FedProc: Prototypical Contrastive Federated Learning on Non-IID data.

Pillutla, K., Malik, K., Mohamed, A., Rabbat, M., Sanjabi, M., and Xiao, L. (2022). Federated Learning with Partial Model Personalization.

Regatti, J. R., Lu, S., Gupta, A., and Shroff, N. (2022). Conditional Moment Alignment for Improved Generalization in Federated Learning. In *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022)*.

Shamsian, A., Navon, A., Fetaya, E., and Chechik, G. (2021). Personalized Federated Learning using Hypernetworks. *Proceedings of Machine Learning Research*, 139:9489–9502.

Sun, T., Li, D., and Wang, B. (2022). Decentralized Federated Averaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Tan, J., Zhou, Y., Liu, G., Wang, J. H., and Yu, S. (2023). pFedSim: Similarity-Aware Model Aggregation Towards Personalized Federated Learning.

Tan, Y., Long, G., Liu, L., Zhou, T., Lu, Q., Jiang, J., and Zhang, C. (2021). FedProto: Federated Prototype Learning across Heterogeneous Clients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8):8432–8440.

Vanhaesebrouck, P., Bellet, A., and Tommasi, M. (2016). Decentralized Collaborative Learning of Personalized Models over Networks. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*.

Vogels, T., Hendrikx, H., and Jaggi, M. (2022). Beyond spectral gap: The role of the topology in decentralized learning.

Wang, L., Zhang, K., Li, Y., Tian, Y., and Tedrake, R. (2023). Does Learning from Decentralized Non-IID Unlabeled Data Benefit from Self Supervision? In *11th International Conference on Learning Representations*.

Xiao, L., Boyd, S., and Kim, S. J. (2007). Distributed average consensus with least-mean-square deviation. *Journal of Parallel and Distributed Computing*, 67(1):33–46.

Xu, J., Tong, X., and Huang, S.-L. (2023). Personalized Federated Learning with Feature Alignment and Classifier Collaboration. *11th International Conference on Learning Representations, ICLR 2023 - Conference Track Proceedings*.

Zantedeschi, V., Bellet, A., and Tommasi, M. (2020). Fully Decentralized Joint Learning of Personalized Models and Collaboration Graphs.

Zhang, J., Guo, S., Ma, X., Wang, H., Xu, W., and Wu, F. (2021). Parameterized Knowledge Transfer for Personalized Federated Learning. *Advances in Neural Information Processing Systems*, 13:10092–10104.

Zhu, Z., Hong, J., and Zhou, J. (2021). Data-Free Knowledge Distillation for Heterogeneous Federated Learning. *Proceedings of Machine Learning Research*, 139:12878–12889.

A Additional Details of Benchmark Datasets

Cifar10. The Cifar10 dataset consists of 60,000 images from 10 disjoint classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The data is further split into 50,000 training samples and 10,000 samples for testing. Each data instance is of dimension 32×32.

MNIST. MNIST dataset consists of 60,000 images of 10 handwritten digits. Each image is of dimension 28×28.

FashionMNIST. FashionMNIST consists of 60,000 images of dimension 28×28 from 10 disjoint fashion classes.

B Additional Details of Training Setup

Each client is assigned one of the following four convolution networks as feature extractors: ResNet-18, ShuffleNetV2, GoogLeNet, and AlexNet. The latent dimension is set to 512. We used one fully connected layer as the classification head g_{ϕ_i} of each model having an input dimension of 512 and an output dimension as the number of classes. For the projection network f_{ψ_i}, we consider a Multilayer Perceptron with 4096 hidden neurons. To train using a single GPU, all the clients are trained sequentially during each global training round. For local updates, we used Adam optimizer with a learning rate of 0.0001.

C MAPL Pytorch-like code

Algorithm 3: MAPL

```
class MAPL:

    def __init__():
        self.id = id
        self.feature_extractor = feature_extractor
        self.projector = projector
        self.classifier = classifier
        self.dataset = dataset
        self.epochs = epochs
        self.warmup = warmup
        self.neighbor = neighbor
        self.weights = torch.ones(num_clients) / num_clients
        self.proto = nn.Linear(10, 1024)

    def train(round):
        # local updates
        for iter in range(self.epochs)
            for x in loader:
                x_a, x_b = augment(x)
                (z_a, z_b) = self.feature_extractor(x)
                (p_a, p_b) = self.projector(z_a, z_b)
                (o_a, o_b) = self.classifier(z_a, z_b)

                # calculation of local loss
                loss_cont = contrastive_loss(p_a, p_b)
                loss_proto = proto_loss(p_a, p_b, self.prototypes)
                loss_uni = uniformity_loss(self.proto)
                loss = loss_cont + loss_proto + loss_uni

                # optimization step
                loss.backwards()
                optimizer.step()

        # graph learning step
        if round ≥ self.warmup:
            for client in self.neighbor:
                sim[j] = sim(self.classifier, client.classifier)

            sim[self.id] = 1.
            sim = max_norm(sim)
            loss = (sim * self.weights) + graph_reg(self.weights)

            # optimization step
            loss.backwards()
            optimizer.step()

            self.weights = proj(self.weights)

        # aggregation
        self.proto = self.proto * self.weights[self.id]
        for client in self.neighbor:
            self.proto = client.proto * self.weights[client.id]
```

3

Deep Learning

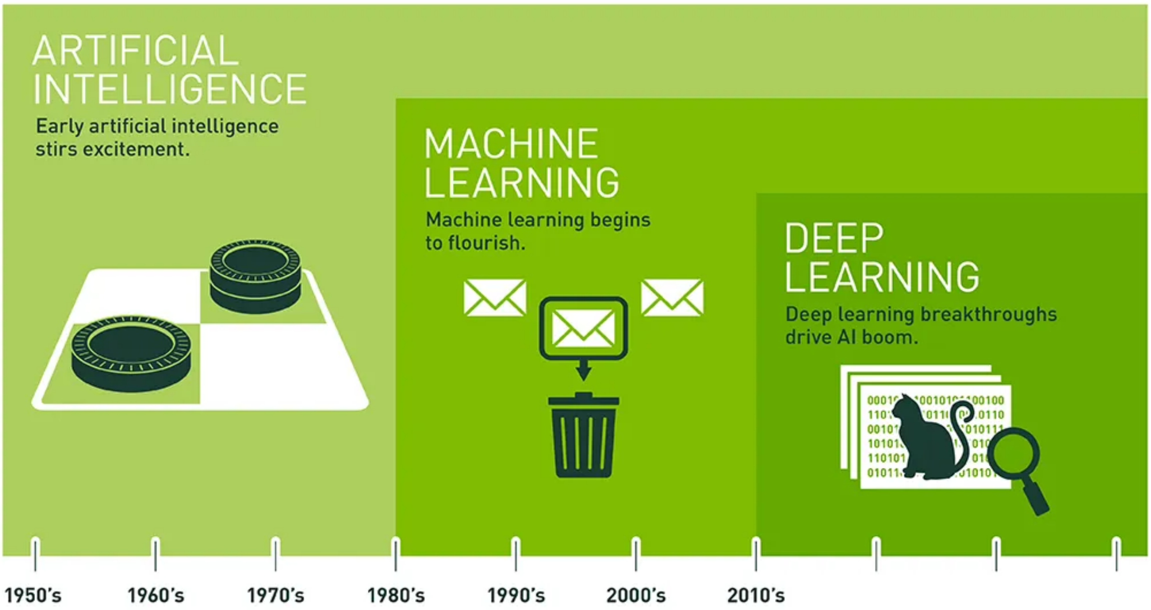

ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Figure 3.1: Overview of Artificial Intelligence [1]

Computer systems in their earlier days required knowledge to be hard-coded by programmers along with all possible edge-cases. However, as capturing complex knowledge in the form of static code is difficult, Artificial Intelligence (AI) systems emerged to extract such knowledge directly from the patterns present in raw data. The process of learning **mapping** from the input data to some output variable is known as **machine learning**. Simple machine learning models such as *logistic regression* and *random forest* where soon found to be extremely effective for various tasks such as spam email detection. Nonetheless, the performance of such simple algorithms relied heavily on the **representation** of data being provided also called **features**. If the features are uncorrelated with the variable being predicted, the machine learning models cannot perform. Identifying and extracting these *relevant* features from data became a key challenge in machine learning. However, for complex data types such as images, creating hand-crafted feature extractors is not be feasible. Therefore, an alternative is to use the machine learning model to learn the representations of the data along with the mapping. It is referred to as **representation learning**. Learned representations besides being effective for the downstream task also allow the models to adapt to novel tasks. The goal of such algorithms is to separate the **factors of variation** that can be used to explain the observable data. Such factors might not be observable and

[1] Source: NVIDIA Blog. Accessed: August 29, 2023

often there are high-order interactions among them. **Deep Learning** offers the means to capture such high-level abstract features from the data as a nested hierarchy of representations obtained using a series of simpler transformations in the form of **layers** in a deep neural network.

3.1. Deep Feed-forward Networks

Deep feed-forward networks, also commonly known as feedforward neural networks **Multilayer Perceptron** (MLP), are a fundamental class of deep learning models. The goal of such networks is to approximate some function f^*. For a K-class classification task, f^* is the ground-truth labelling function that maps a data point $x \in \mathcal{X}$ in from the input space $\mathcal{X} = \mathbb{R}^d$ to a category $y \in \mathcal{Y}$ in the output space $\mathcal{Y} := \{1, \cdots, K\}$. Deep feed-forward networks approximate the function f^* as $\hat{y} = f(x; \theta)$ and learns the value of the parameter θ.

The information flows in a one-way direction in these models, from the input it passes through the computations used to approximate f^* to the output. Thus they are known as **feedforward** networks. The process of passing an input sample x through is referred to as **forward pass**. The term **network** on the other hand comes from the fact that it usually comprises of several functions. It is accompanied by a directed acyclic graph signifying how the functions are composed together. As an example we can consider three functions $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$ which are connected to form a chain $f(x) = f^{(1)}(f^{(2)}(f^{(3)}(x)))$. These chain structures are the most commonly found in deep neural networks where the first layer ($f^{(3)}$) is the input layer, the last layer ($f^{(1)}$) is the output layer with the length of the chain denoting the **depth** of the network. It is also what gives rise to the term **deep learning**. The training data along with training strategy is what dictates the output of the final layer. The layers in the middle (here $f^{(2)}$) that do not directly give the output are called **hidden** layers.

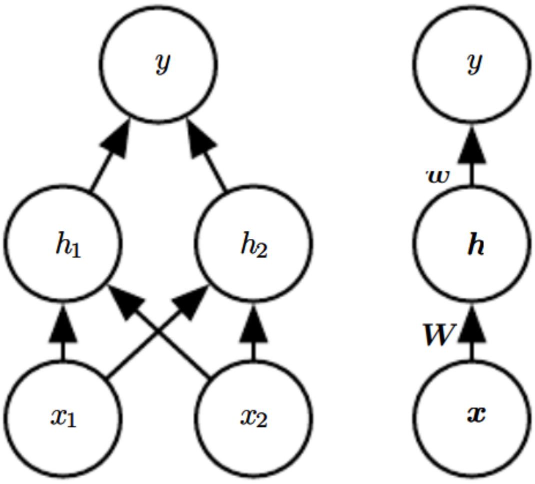


Figure 3.2: Example of a feedforward network from [15]

The hidden layers of the network is typically vector valued whose dimension determines the **width** of the network. Each layer can be considered to be composed of several units that act in parallel instead of considering each layer as a single unit. Each element of the vector acts as a neuron. The Figure 3.2 shows two depictions of a feed-forward network with one hidden layer. The hidden layer has 2 units as depicted on the left of the Figure 3.2 which can also be denoted by a single 2D vector $h \in \mathbb{R}^2$ as shown on the right.

To clearly understand the deep feed-forward networks, we can start from a linear model parameterized by θ consisting of weight w and **bias** b

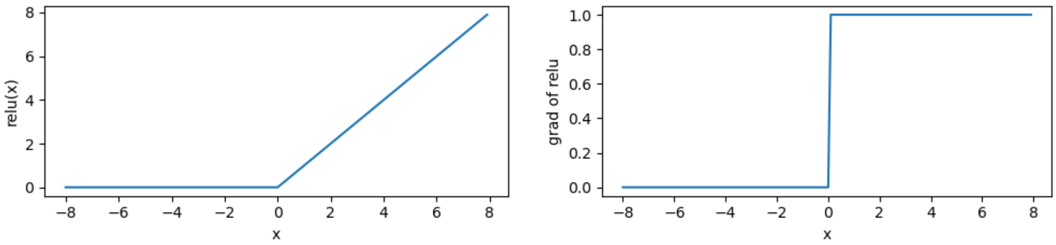
$$f(x; \theta) = f(x; w, b) = w^T x + b \tag{3.1}$$

Although such a linear model often is associated with a closed form solution, a limitation is the inability to fit to nonlinear data. Even if we use a multiple layers such that each layer is a linear transformation, the overall transformation is still linear. However, most of the real-world data is non-linear. To accomodate for this, we can consider a applying the linear model not on the original data x but on the transformed data $\Phi(x)$ where Φ is a non-linear transformation. Another commonly applied technique is to apply **affine transformation** at each layer controlled by the network parameters θ followed by a non-linear **activation** function.

3.2. Non-linearity

Non-linearity is incorporated in the deep neural networks are incorporated as **activation functions**. Activation functions are differentiable operators that decide if the the neurons of will be activated for an input. Below we will be discussing three of the most commonly used activation functions

3.2.1. ReLU



(a) ReLU activation function

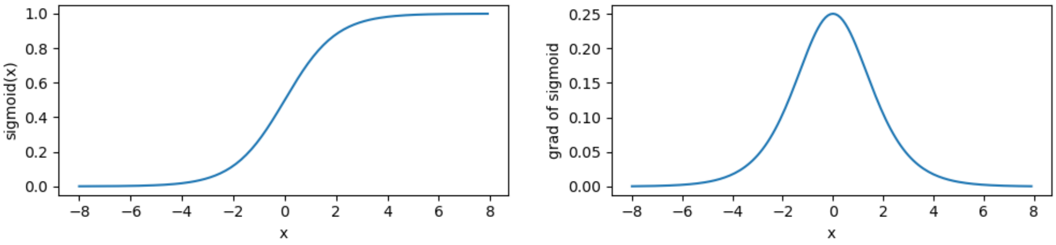
(b) Gradient of ReLU activation function

Rectified Linear Unit (ReLU) is the most commonly used activation function due to its computational simplicity and its effectiveness. Given an input x, ReLU outputs the maximum between 0 and the value of x as shown in the Equation 3.2. In essence ReLU discards the negative parts of the inputs by mapping them to zero as can be seen in the Figure 3.2. ReLU is differentiable except at 0. The derivative of ReLU as seen in the Figure 3.3b is 0 for all the negative inputs and 1 for positive inputs.

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.2}$$

However, it's worth noting that ReLU can also suffer from a problem known as the **dying ReLU** problem, where neurons consistently output zero during training. To address this, variations of ReLU such as **Leaky ReLU**, and **Exponential Linear Units** (ELU) have been proposed, each with their own modifications to the basic ReLU formulation.

3.2.2. Sigmoid



(a) Sigmoid activation function

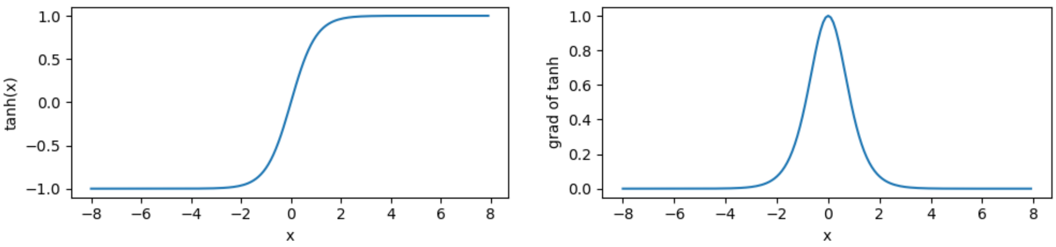
(b) Gradient of Sigmoid activation function

The sigmoid activation function is a mathematical function commonly used in neural networks and machine learning models. It transforms an input value into an output value that ranges between 0 and 1. This makes it particularly advantageous for models where the goal is to predict probabilities, as probabilities inherently exist within the 0 to 1 range. Given an input x, the sigmoid function is formulated as

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + e^{x}} \tag{3.3}$$

The sigmoid activation function is characterized by its S-shaped curve, which smoothly transitions from near 0 to near 1 as the input value changes as shown in the Figure 3.4a. It is differentiable at any point. The smooth gradient of the sigmoid function as shown in Figure 3.4b is another advantage, as it prevents abrupt transitions in output values, ensuring a smoother learning process. However, it is susceptible to the problem of gradient vanishing, occurring when the sigmoid function's value is either extremely high or low, leading to an almost negligible derivative (« 1).

3.2.3. Tanh



(a) Tanh activation function

(b) Gradient of Tanh activation function

Tanh is a hyperbolic tangent function. The curve for tanh as seen in the Figure 3.5a is similar to the sigmoid curve with some exceptions. Unlike sigmoid, the tanh curve is 0 centered and the values can range between -1 and 1. The major advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero. Given an input x, the tanh function is formulated as

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.4}$$

It is also differentiable at every point on the curve. The curve corresponding to the derivative is shown in the Figure 3.5b. Generally, tanh is considered for hidden layers whereas sigmoid is considered for the final output layer. However, the final choice of activation functions is based on the task that the neural network is designed to address.

3.3. Loss Functions

An objective function or **loss** function is a mathematical expression that quantifies the difference between the predicted outputs of the network and the actual target values. The choice of a specific loss function depends on the type of task the neural network is designed to solve. Different tasks, such as classification, regression, or even more specialized objectives, require different types of loss functions. Some notable ones are briefly discussed below considering an input $x \in \mathcal{X}$, the ground truth $y \in \mathcal{Y}$ and neural network $f(\cdot;\theta)$:

- **Mean Squared Error (MSE)**: Commonly used for regression tasks ($\mathcal{Y} = \mathbb{R}$), MSE calculates the average of the squared differences between predicted and actual values. It penalizes larger errors more heavily.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{3.5}$$

- **Cross-Entropy Loss**: Suitable for classification problems, cross-entropy loss quantifies the dissimilarity between predicted class probabilities and actual class labels. It's particularly effective for tasks with multiple classes.

$$\mathrm{CE} = \frac{1}{N}\sum_{i=1}^{N} y_i \frac{\exp(\hat{y}_i)}{\sum_{j=1}^{K}\exp(\hat{y}_j)} \tag{3.6}$$

- **Binary Cross-Entropy Loss**: Similar to cross-entropy but used for binary classification, where there are only two classes.

$$\mathrm{BCE} = \frac{1}{N}\sum_{i=1}^{N}\Big[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\Big] \tag{3.7}$$

3.4. Optimization

As mentioned earlier, the goal of any deep learning model is to learn an approximation of a mapping $f^* : \mathcal{X} \to \mathcal{Y}$ from the input space to the output space. In order to do so, it needs to find an optimal set of parameters θ. However, directly finding an accurate value of θ is difficult as there are many unknowns involved. Thus, we formulate the learning problem as an optimization problem and use an algorithm to search a good set of parameters from a feasible set of values such that the loss is minimized.

3.4.1. Gradient Descent

Gradient descent is an optimization algorithm used to update the weights and biases of a neural network in order to minimize the loss function. The idea is to iteratively adjust the parameters in the direction that decreases the loss. Given a function $f(x)$, the first order derivative $f'(x)$ is called the gradient or the slope of the function at the point x. The gradient indicates how adding a small change ϵ to x will affect the output: $f(x) \approx f(x) + \epsilon f'(x)$. An illustration of gradient descent on the function $f(x) = \frac{1}{2}x$ is shown in the Figure 3.6.

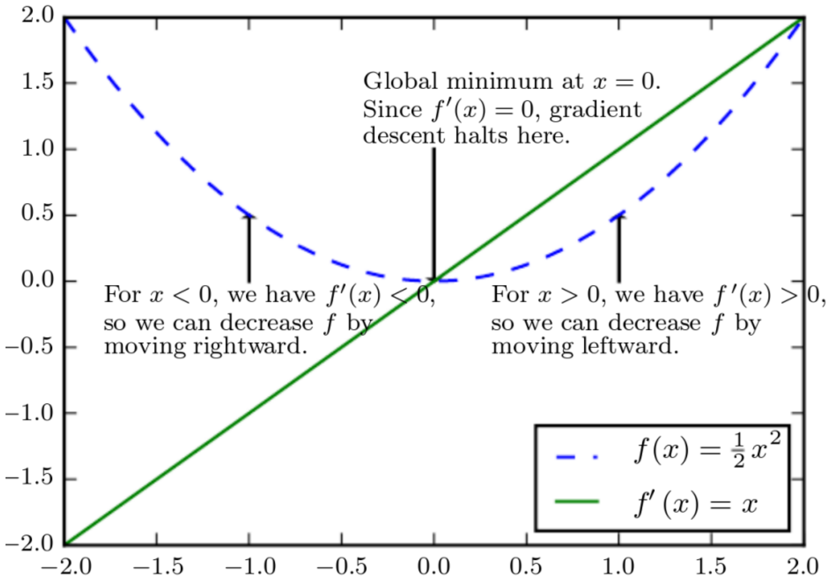


Figure 3.6: Gradient descent algorithm illustration for finding the minimum of a function $f(x) = \frac{1}{2}x$

While training neural networks, the objective function is the loss incurred by the model on a training sample an we minimize it with respect to the model parameters θ. Starting with some random value we iteratively moving in the opposite direction of the gradient. Gradient descent can be of different types namely **Stochastic Gradient Descent** (loss is computed on a randomly chosen data from the dataset), **Batch Gradient Descent** (loss is computed as an average over the entire dataset) and **Mini-batch Gradient Descent** (loss is computed over a subset of data from the dataset).

3.4.2. Backpropagation

As deep neural network consists of several layers with each layer having an associated parameter, there is need to propagate the error back through the entire network in a process called **backpropagation**. Once the prediction is generated in a forward pass, the difference between the prediction and the actual target (the error) is computed using the chosen loss function. This error is then propagated backward through the network. Backpropagation calculates the gradients of the loss with respect to each parameter which are then used to adjust the weights and biases in a way that reduces the error.

3.4.3. Optimizers

Following the calculation of gradient at each layer, the weights are updated by a small value η in the direction of the gradient. The value η is called the **learning rate** and determines the step size in the parameter space. A larger learning rate can lead to faster convergence, but it might also cause the optimization process to oscillate or overshoot the optimal solution. A smaller learning rate can lead to more stable convergence, but it might make the optimization process slower. Additionally, the loss functions used in deep learning often does not have a sample loss curve as shown in Figure 3.6. Instead, there can be multiple local minimas along with the global minima, Thus, there is always a possibility to get suboptimal results if the learning rate is chosen incorrectly. One way to counter this is to observe the loss landscape and alter the learning rate accordingly to avoid such minimas. However, in practice it is not feasible. Thus we use various **optimizers** that monitor the gradients and make changes to the learning rate based on the past. Some of the most commonly used ones are:

- **Momentum**: Smooths the average of noisy gradients with Exponentially Weighted Moving Average (EWMA)

$$\begin{aligned} v_i &= \rho v_{i-1} + (1-\rho)\nabla_\theta, \\ \theta_i &= \theta_{i-1} - \eta v_i \end{aligned} \tag{3.8}$$

  where ρ is a hyperparameter.

- **RMSprop**: Smooths the zero-centered variance of noisy gradients with EWMA

$$\begin{aligned} r_i &= \rho r_{i-1} + (1-\rho)\nabla_\theta^2, \\ \theta_i &= \theta_{i-1} - \eta \frac{\nabla_\theta}{\sqrt{r_i + \delta}}, \end{aligned} \tag{3.9}$$

  where δ is a small value added to prevent 0 in the denominator.

- **Adam**: It is combination of **momentum** and **rmsprop**

$$\begin{aligned} v_i &= \rho_1 v_{i-1} + (1-\rho_1)\nabla_\theta \\ \hat{v}_i &= \frac{v_i}{1-\rho_1^i} \\ r_i &= \rho_2 r_{i-1} + (1-\rho_2)\nabla_\theta^2, \\ \hat{r}_i &= \frac{r_i}{1-\rho_2^i} \\ \theta_i &= \theta_{i-1} - \eta \frac{\hat{v}_i}{\sqrt{\hat{r}_i + \delta}}, \end{aligned} \tag{3.10}$$

3.5. Regularization

Regularization is a technique in deep learning used to prevent overfitting, a phenomenon where a neural network learns to perform exceptionally well on the training data but performes poorly on new, unseen data. Regularization methods add additional constraints or penalties to the training process to encourage the network to learn more generalized patterns.

[2] Source: Matlab Page. Accessed: August 29, 2023

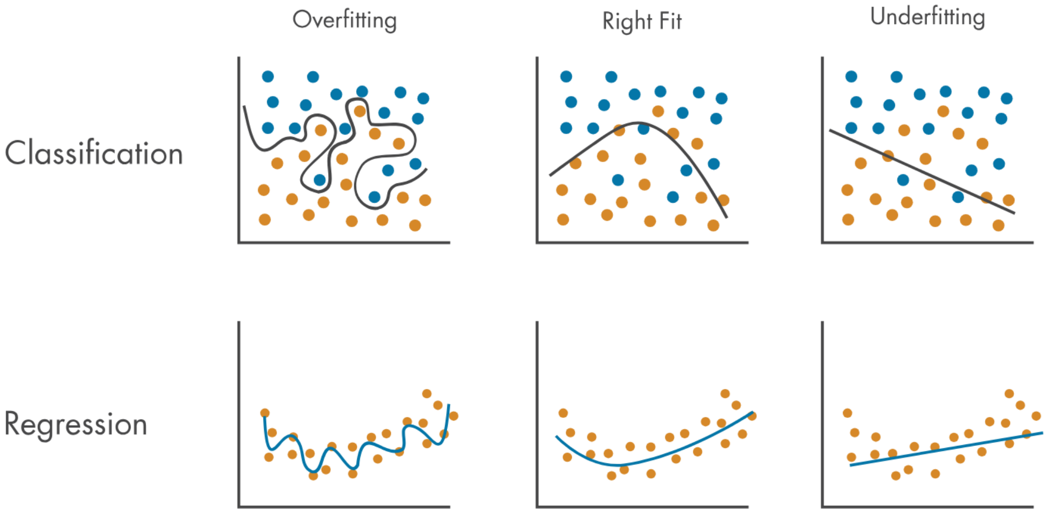


Figure 3.7: Overfitting and underfitting [2]

- **L1 and L2 Regularization (Lasso and Ridge)**: L1 regularization adds a penalty term to the loss function that is proportional to the absolute values of the weights. L2 regularization adds a penalty term that is proportional to the square of the weights. Both techniques encourage the model to use smaller weights, effectively preventing some neurons from having strong influences.

- **Dropout**: During each training iteration, randomly selected neurons are ”dropped out” or ignored during forward and backward passes. This prevents any single neuron from becoming overly dependent on specific input features, leading to a more robust network.

- **Data Augmentation**: While not strictly a regularizer, data augmentation involves applying random transformations (like rotations, flips, or translations) to the training data. This artificially increases the diversity of the training data, helping the model generalize better to new data.

- **Early Stopping**: Early stopping involves monitoring the model's performance on a validation dataset and stopping training when the performance starts to degrade. This prevents the model from fitting noise in the training data.

- **Batch Normalization**: Batch normalization involves normalizing the inputs of each layer in a way that helps maintain a stable distribution of values throughout training. This can prevent the network from becoming overly sensitive to small changes in input.

3.6. Convolutional Neural Network

Convolutional Neural Networks (CNNs) are special type of deep feedforward network used in image and signal processing. One of the earliest CNN was introduced by LeCun et al. [22] as shown in the Figure 3.8. It primarily consists of two key components, namely a **feature extraction head** made of several convolution and pooling layers stacked over each other and a **task-specific head** consisting of fully connected layers. A simplied visualization of a convolutional feature extractor is shown in the Figure 3.9.

3.6.1. Convolution

Convolution is a fundamental operation in image and signal processing, particularly in the context of neural networks used for computer vision tasks. A small matrix (also referred to as a filter or kernel) is defined, typically smaller than the input image. This kernel contains numerical values that represent the pattern to be detected or extracted. Then convolution involves sliding of the filter or kernel over an input image or signal to extract meaningful features or patterns. As the filter slides over the input, dot product is computed between the pixel values and the kernel matrix and the output is stored in an

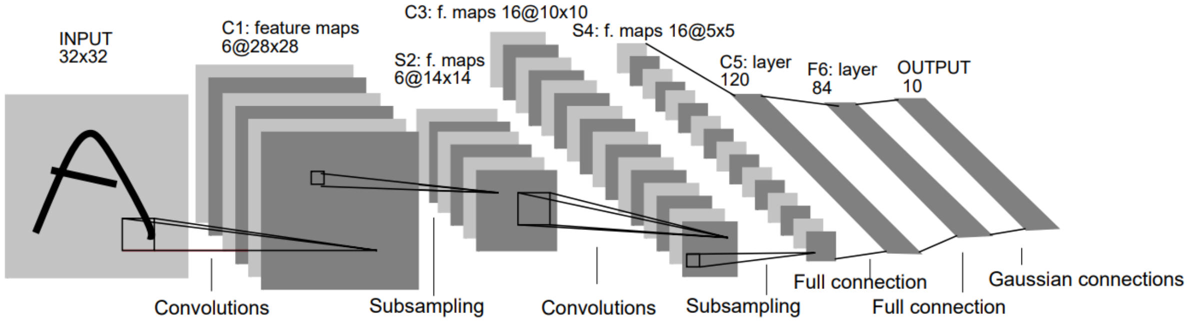


Figure 3.8: LeNet 5 architecture.

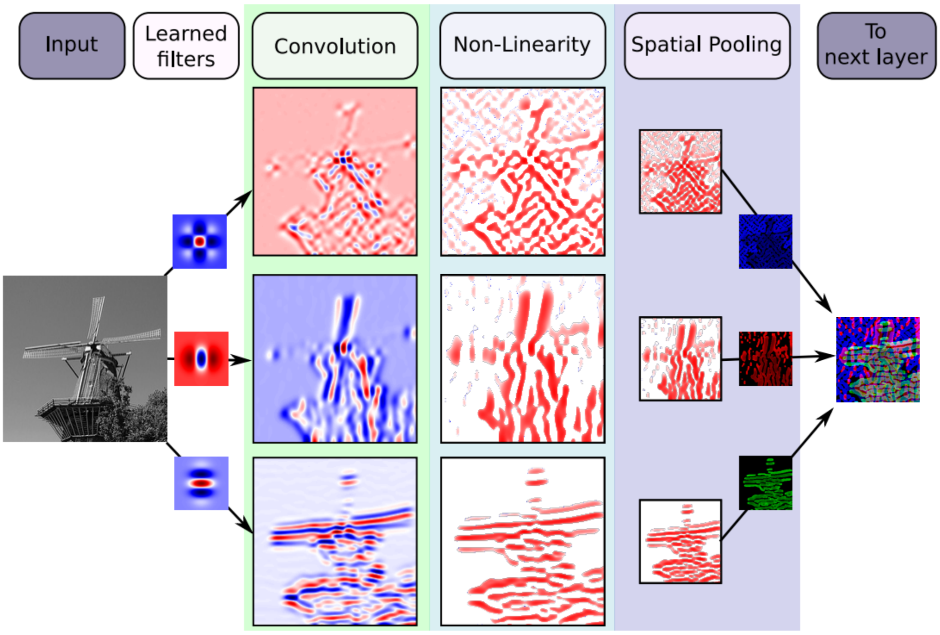


Figure 3.9: Visualization of CNN. Taken from the Deep Learning course (CS4240) at TU Delft.

output matrix. The final output matrix is called a **feature map** or an **activation map**. The convolution operation has several key properties:

- **Sparse Interactions**: Since the filter is smaller than the input, it only interacts with a small portion of the image at a time, capturing local patterns.
- **Parameter Sharing**: The same filter is used at all positions across the image, allowing the network to learn common features regardless of their location.
- **Translation Invariance**: Convolution allows the network to recognize patterns regardless of their position in the image, making the model robust to object translations.

3.6.2. Pooling

Similar to convolution, pooling involves sliding a kernel over the input image. However instead of computing a dot product, the pixels values falling inside the kernel boundaries are aggregated and stored in the output matrix. Depending on the aggregation function used the pooling can be of two types, namely **Max-Pooling** where the maximum value over and **Average Pooling** where an average is computed over the concerned pixel values.

4 Self-Supervised Learning

4.1. Why SSL?

Deep learning has shown immense progress in recent years in various tasks, especially in computer vision and natural language processing. Supervised learning has a proven record of creating models specialized for a particular task. However, the supervised paradigm has an inherent bottleneck: relying on a massive corpus of manually labelled data. Furthermore, supervised learning methods need a dedicated labelled corpus for each new task. Labelling data is expensive and requires a substantial amount of time. A glaring example is the ImageNet dataset that took years to label only a small subset of semantic classes known to humankind. Moreover, some tasks, such as novelty detection, lack a sufficient amount of data by definition. Chollet et. al. [10] referred to intelligence in the context of AI as the efficiency of transforming new data into new skills. Supervised approaches have a severe limitation in developing such ”intelligent” models.

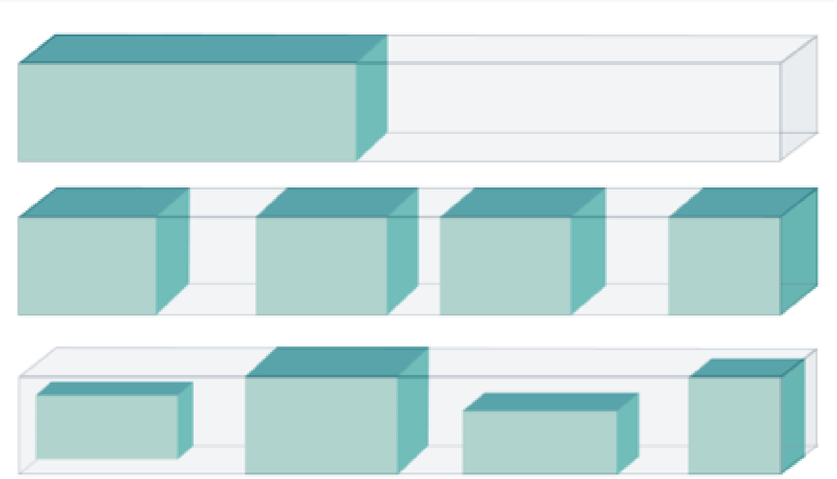


Figure 4.1: In SSL, the system is trained to predict hidden parts of the input (in gray) from visible parts of the input (in green)

One of the promising paradigms to overcome the shortcomings of supervised learning is self-supervised learning (SSL), which obtains supervisory signals from the data without relying on any labelled corpus. Such methods learn the underlying structure of the data using some *pretext task* [12]. One of the major achievements of self-supervised learning is in the domain of natural language processing (NLP). NLP models like BERT are pre-trained on huge uncurated data extracted from the internet by masking some segments of the sentences and making the models predict the masked sections. Such a pre-training method allows the language models to learn the inherent structures in natural language, enabling them to demonstrate superior performance on any downstream language task. Based on the success in the language domain, self-supervised methods have drawn considerable attention from researchers in recent years for various computer vision tasks.

The main problem in the adaptation of SSL in computer vision is the dimensionality of data and the infinite number of possible options for a missing patch in an image or a missing frame in a video compared to the number of feasible options for a missing word in a sentence which is often restricted to the size of the vocabulary. Thus predictive pretext tasks employed in NLP are not feasible for computer vision. Early pretext tasks in computer vision involved solving jigsaw puzzles [30] or predicting image rotations [14]. However, it was noticed that such pre-training did not generalize well to the downstream tasks. In essence, the main goal of pretext tasks in the context of computer vision is establishing similarity between images or video frames and learning invariance to external factors such as illumination, scale, deformation and variance within each semantic class. Thus most SSL methods in computer vision use joint-embedding networks to learn invariance to data augmentations as a pretext task. An important consideration while using joint-embedding networks is preventing them from collapsing to trivial solutions such that the input is ignored.

Recent methods in SSL avoid the trivial solution by either using various techniques. Currently, employed methods can be broadly categorized into contrastive learning, clustering, distillation, and redundancy reduction-based methods.

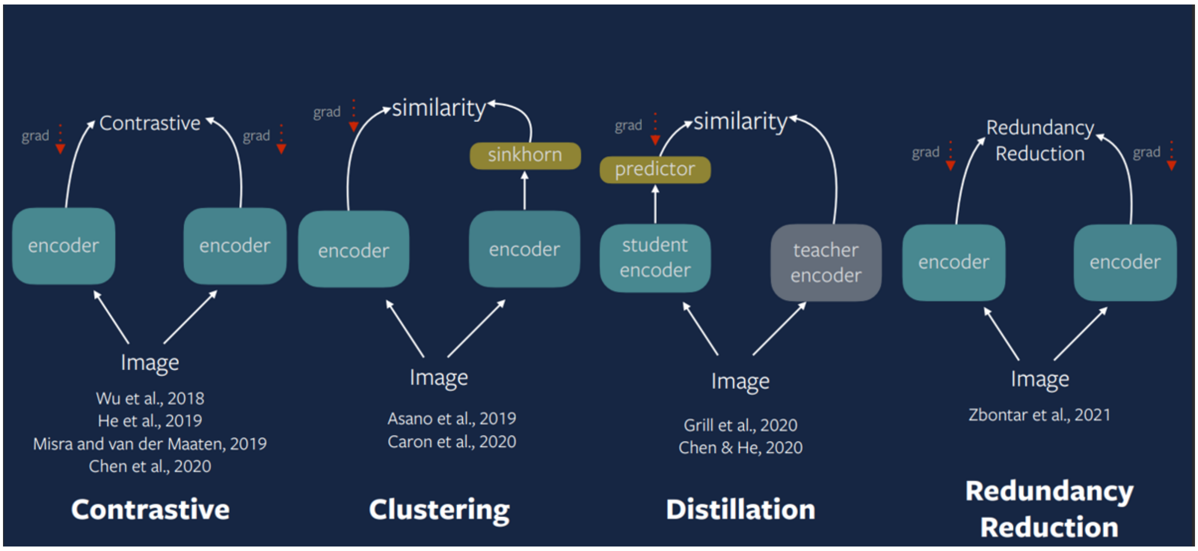


Figure 4.2: Overview of SSL methods. [1]

4.2. Contrastive-based Approaches

Contrastive learning-based methods (PIRL [28], SimCLR [7, 8], MoCo [17]) use a loss function that forces embeddings from related images to be closer in the embedding space while pushing the embeddings from the unrelated images further apart using negative samples to prevent a trivial solution (all samples mapping to the sample point). Related images in such a setup are created by taking different views of the same image by applying augmentations. In contrast, unrelated images (or negative samples) can be taken either from a distant location within the same image [28] or from a different semantic class [17]. However, the InfoNCE loss used in the contrastive approaches needs many negative samples [31] for effective learning. SimCLR addresses this issue by having a large batch size across multiple GPUs, whereas MoCo utilizes a memory bank of embeddings, resulting in the requirement of extra computing power or memory for training.

[1] Source: Ishan Mishra SSL Slides. Accessed: August 29, 2023

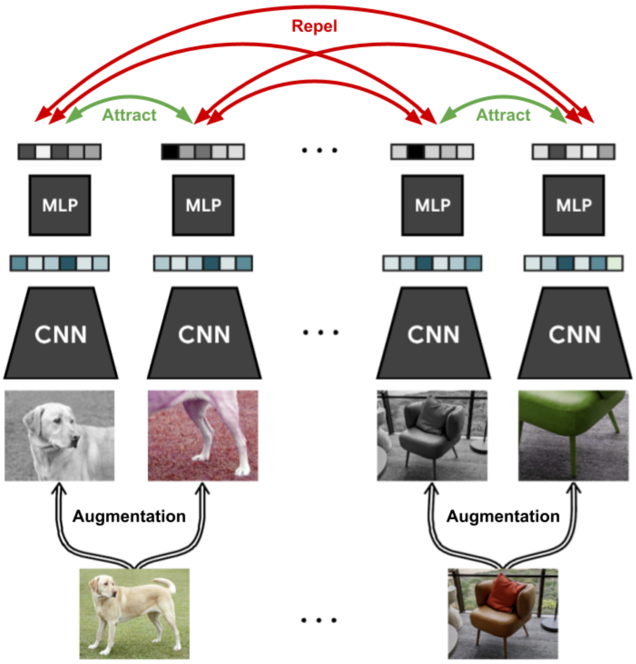


Figure 4.3: Visualization of SimCLR [7]

4.3. Clustering-based Approaches

Looking deeper into contrastive learning reveals that it essentially clusters related embeddings in the latent space. Hence, one possible way to overcome the reliance on negative samples is to directly apply clustering (DeepClustering [5], SeLA [1], SwAV [6]). The trivial solution with clustering-based approaches is the assignment of all embeddings to a single cluster. The proposed methods prevent such a trivial solution using an equipartition constraint based on optimal transport.

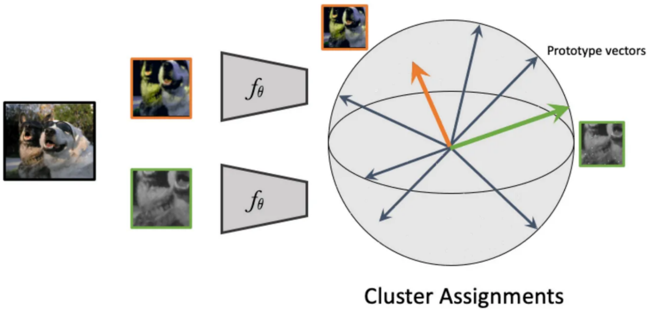


Figure 4.4: Visualization of SwAV [6]

4.4. Distillation-based Approaches

Another approach to bypass the need for negative samples are using a distillation-based approach (BYOL [16], SimSiam [9]) involving a teacher-student interaction. In such methods, the trivial solution is avoided by incorporating some asymmetry either in the learning rule or in the architecture of the teacher and student. Whereas BYOL used 3 sources of asymmetry to avoid trivial solutions, namely, architectural, learning rule and weights asymmetry, SimSiam demonstrated that similar performance could be obtained even with the first two types of asymmetry.

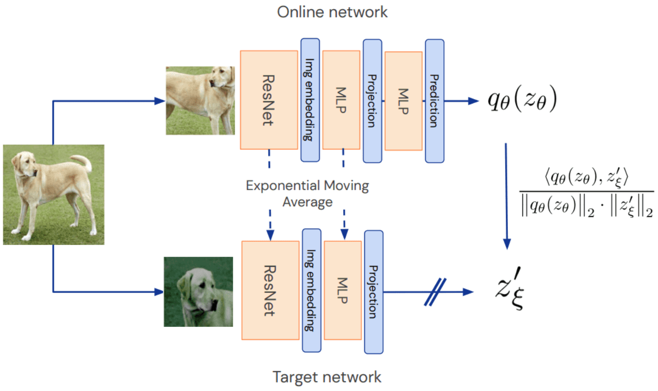


Figure 4.5: Visualization of BYOL [16]

4.5. Redundancy reduction-based Approaches

Redundancy reduction-based methods are inspired by neuroscience and information theory. Barlow Twins [40] is based on Horace Barlows Efficient Coding Hypothesis [3], which talks about the nervous system transmitting a signal using efficient ”spiking code” such that the number of spikes is limited. Barlow Twins primarily satisfy two constraints - invariance to data augmentations and independence of neurons for a reduction in redundancy. This setup does not include any asymmetry like the distillation-based approaches or negatives like the contrastive methods. Instead, it relies on the invariance and redundancy-based loss terms to prevent collapse.

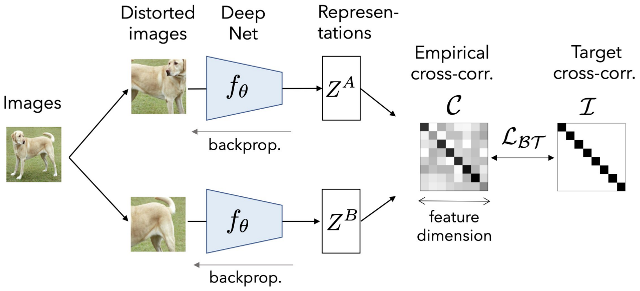
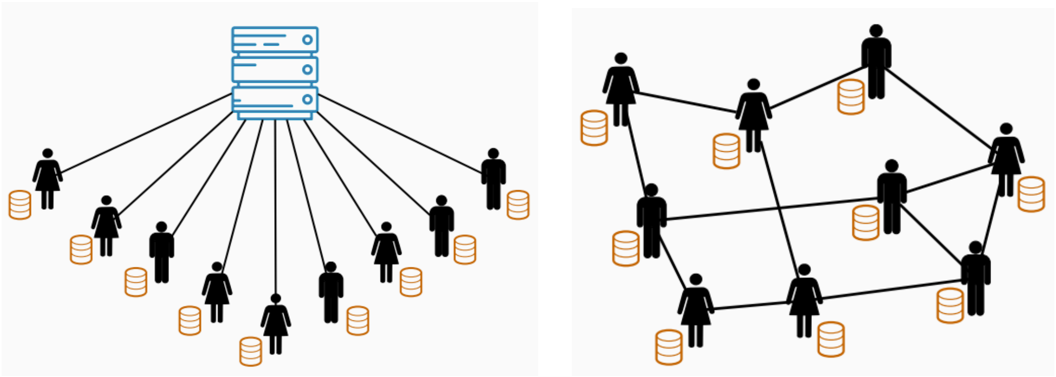


Figure 4.6: Visualization of Barlow Twins [40]

5

Decentralized Machine Learning

Machine learning, especially deep learning, has gained a lot of attraction from academia and industry owing to its impressive capabilities. A great example is the emergence of Large Language Models such as ChatGPT, which can write human-like text and answer open questions. However, state-of-the-art deep learning models require substantial data to learn complex patterns and generalize to unseen data. In the classical machine learning setting, the data is often assumed to be **centralized** either on a single machine or among a group of well-connected servers, for example, in a data centre. The problem arises because the data is inherently decentralized in most practical scenarios. From hospitals collecting patient data to cell phones collecting data about their users, data resides primarily on edge devices. Aggregating such data in a centralized location not only incurs huge **communication** costs but also requires high costs for **storage**. Most applications often assume the presence of a **trusted agent** to collect all the data. However, when dealing with sensitive data like hospital patient information, users might be reluctant to share the data, or there might be legal restrictions. Thus it is important to have a privacy-preserving method capable of learning from the decentralized data at the edge devices.



(a) Federated Learning

(b) Fully decentralized (peer-to-peer) Learning

Figure 5.1: Types of decentralized learning[1]

To address these problems, **decentralized learning** emerged. There are two primary settings referred to in the literature as shown in Figure 5.1, namely **Federated Learning** (FL), where a **central server** acts as a moderator among the clients, and **Fully decentralized** or Peer-to-peer (P2P) Learning where clients directly communicate among each other. The Fl setting can further be divided into **cross-device** and **cross-silo**, where a central server moderates among the devices and institutions. It brings the computation from the data centres to the edge devices where the data commonly resides. The clients or the users perform local update steps by iteratively pushing their local data through the

[1] Source: International Workshop on Distributed Cloud Computing, 2020.. Accessed: August 29, 2023

[2] Source: NeurIPS Tutorial Slides. Accessed: August 29, 2023

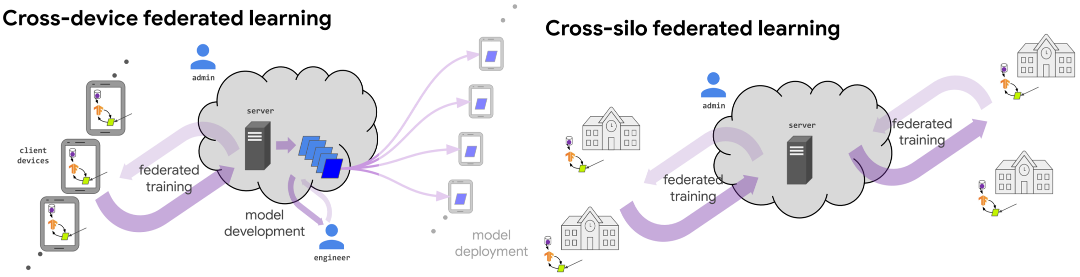


Figure 5.2: Types of federated learning[2]

model and updating the model parameters using gradient descent. Then the local updates are then either aggregated at a central server [27] or locally using P2P communications [25]. Although using a trusted central server for facilitating collaboration is the most common setting in literature, there are some obvious shortcomings. First, due to the heavy reliance on the central server, it often becomes a bottleneck. Moreover, it is not scalable to a large number of clients. Using P2P communications, each client only communicates with its immediate neighbours and thus is naturally scalable. Secondly, the central server is prone to adversarial attacks as it is a single point of failure.

5.1. Challenges in Decentralized Learning

Irrespective of the setting, decentralised machine learning methods face additional challenges compared to centralized learning. Some of the critical challenges are highlighted below based on [37].

- **Communication efficiency**. Communication is the main bottleneck for devices with limited bandwidth. Thus instead of performing frequent updates with a central server, it is preferred to perform several steps of local updates before communication.
- **Computational constraints**. Varied hardware at each client results in computational constraints. For example, slow devices would stall the entire learning process. Therefore, in such scenarios, an asynchronous learning method is often considered where the communication with the central server is not synchronised. Additionally, some devices might not be capable of handling large deep learning models resulting in a model heterogeneous setting which is challenging to address.
- **Data Heterogeneity**. The local data of each client is never shared and is highly heterogeneous, as each client's data comes from different distributions. Furthermore, each client might have varying data samples, resulting in an overall statistical heterogeneity. In a centralized setting, there is a possibility of redistributing the data among worker nodes such that the heterogeneity is reduced. However, in Federated Learning, such methods are not applicable, adding to the complexity of an already complex problem.

  Considering the distribution of data from two clients i and j to be P_i and P_j, the different sources of heterogeneity can be characterised as follows [21]:

  - **Feature distribution skew** (covariate shift): The marginal distribution $P(x)$ varies among clients whereas the conditional distribution distribution $P(y|x)$ is same across all the clients.
  - **Label distribution skew** (prior probability shift): The marginal distribution $P(y)$ varies across clients and the conditional $P(x|y)$ remains the same.
  - **Same label, different features** (concept drift): The conditional distribution $P(x|y)$ varies across the clients despite $P(y)$ being shared.
  - **Same features, different label** (concept shift): The conditional distribution $P(y|x)$ varies even though the marginal distribution $P(x)$ remains same.
  - **Quantity skew or unbalancedness**: Each client owns vastly different amounts of data samples.

- **Privacy**. Privacy is the key goal of a federated learning setting. Thus methods are often constrained to access only limited information in the form of aggregates or learned representations. Furthermore, other privacy techniques, such as differential privacy, are often added to federated learning.

5.2. Problem Formulation

Consider a set of M users where each user $i \in [M]$ has access to a local dataset $\mathcal{D}_i$ consisting of n_i data samples. Each dataset is drawn from the local distribution of client i from an input space $\mathcal{X}$. Thus the local datasets are different, i.e. $\mathcal{D}_i \neq \mathcal{D}_j$. The joint dataset can thus be represented as $\mathcal{D} = \bigcup_{i \in [M]} \mathcal{D}_i$ consisting of $n = \sum_{i \in [M]} n_i$ samples. It should be noted that the joint dataset is never aggregated at a central location. Further considering a model $h \in \mathcal{H}$ belonging to the hypothesis class $\mathcal{H}$ along with a loss function $l : \mathcal{H} \times \mathcal{X} \to \mathbb{R}$ that measures the performance of the model, the objective for federated learning can be expressed as

$$\underset{h \in \mathcal{H}}{\operatorname{argmin}} \Big\{ L(\mathcal{D}; h) = \sum_{i \in [M]} \alpha \, l(\mathcal{D}_i; h) \Big\}, \tag{5.1}$$

where the global objective function L is expressed as a weighted sum over local objective functions with weights α. To find an approximate solution to Eq. 5.1 collaboratively in a decentralized setting, users perform local computations and communicate the results by sending messages. Due to the separable structure of Eq. 5.1, a user conducts gradient descent concerning a local objective and communicates for aggregating local updates. As discussed earlier, communication can occur in one of two ways. In cross-silo or cross-device federated learning, users rely on a central agent in a client-server architecture organized in a star topology. Each user communicates the local updates with the central moderating agent. The agent then computes the aggregates and returns the results to the users. On the contrary, in a fully decentralized setting, users are organised in an arbitrary network topology represented as a graph $\mathcal{G} = \{[M], \mathbf{W}\}$ where the nodes refer to the users. The edge weight matrix $\mathbf{W} = \mathbb{R}^{M \times M}$ represent the weight associated with an edge with $w_{ij} = 0$ denoting the absence of an edge. In such a setting, clients only communicate locally with their neighbours.

6 Conclusion

We explored a practical yet challenging model heterogeneous scenario in a fully decentralized setting, which has not been studied in the literature. To this end, we proposed MAPL to simultaneously learn heterogeneous personalized models in a fully decentralized setting. Furthermore, we formulated a graph learning objective to infer the optimal collaboration weights based on local task similarity between the clients. Through experimentation, we observed that even in this difficult setting, MAPL provides competitive results while being communication efficient owing to the sparse collaboration graph. Finally, even in the model homogeneous setting, MAPL outperformed both the centralized and decentralized baselines.

As future work, firstly, an intriguing avenue to explore would be to investigate the extent of diversity that MAPL can manage while delivering a meaningful enhancement within a decentralized framework. In addition, conducting a thorough theoretical analysis of our approach could also provide valuable perspectives for refinement. Secondly, MAPL can be expanded to accommodate time-varying topologies where new clients can be added or removed from the network. Furthermore, without a trusted central server, peer-to-peer learning is prone to adversarial attacks by malicious clients. Thus, it is essential to analyze the robustness of MAPL against such attacks. We can consider incorporating additional privacy-preserving techniques to further safeguard the clients' privacy.

Bibliography

[1] Yuki Markus Asano, Christian Rupprecht, and Andrea Vedaldi. Self-labelling via simultaneous clustering and representation learning. In *International Conference on Learning Representations*, 11 2019.

[2] Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Gérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. Decentralized Federated Learning: Fundamentals, State-of-the-art, Frameworks, Trends, and Challenges. 11 2022.

[3] M.D. Binder, N. Hirokawa, and U. Windhorst. Efficient Coding Hypothesis. *Encyclopedia of Neuroscience*, pages 1037–1037, 11 2009.

[4] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Gossip algorithms: Design, analysis and applications. *Proceedings - IEEE INFOCOM*, 3:1653–1664, 2005.

[5] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep Clustering for Unsupervised Learning of Visual Features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11218 LNCS:139–156, 7 2018.

[6] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. *Advances in Neural Information Processing Systems*, 2020-December, 6 2020.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. *37th International Conference on Machine Learning, ICML 2020*, PartF168147-3:1575–1585, 2 2020.

[8] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big Self-Supervised Models are Strong Semi-Supervised Learners. *Advances in Neural Information Processing Systems*, 2020-December, 6 2020.

[9] Xinlei Chen and Kaiming He. Exploring Simple Siamese Representation Learning. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 15745–15753, 11 2020.

[10] François Chollet. On the Measure of Intelligence. *arXiv preprint arXiv:1911.01547*, 11 2019.

[11] Rong Dai, Li Shen, Fengxiang He, Xinmei Tian, and Dacheng Tao. DisPFL: Towards Communication-Efficient Personalized Federated Learning via Decentralized Sparse Training, 6 2022.

[12] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised Visual Representation Learning by Context Prediction, 2015.

[13] Yasaman Esfandiari, Sin Yong Tan, Zhanhong Jiang, Aditya Balu, Ethan Herron, Chinmay Hegde, and Soumik Sarkar. Cross-Gradient Aggregation for Decentralized Learning from Non-IID data. 3 2021.

[14] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised Representation Learning by Predicting Image Rotations. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 3 2018.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[16] Jean Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised Learning. *Advances in Neural Information Processing Systems*, 2020-December, 6 2020.

[17] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9726–9735, 11 2019.

[18] Hadrien Hendrikx. A principled framework for the design and analysis of token algorithms. 5 2022.

[19] Hadrien Hendrikx, Francis Bach, and Laurent Massoulié. Dual-Free Stochastic Decentralized Optimization with Variance Reduction. *Advances in Neural Information Processing Systems*, 33:19455–19466, 2020.

[20] Hadi Jamali-Rad, Mohammad Abdizadeh, and Anuj Singh. Federated Learning With Taskonomy for Non-IID Data. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[21] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and Open Problems in Federated Learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 12 2019.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[23] Shuangtong Li, Tianyi Zhou, Xinmei Tian, and Dacheng Tao. Learning To Collaborate in Decentralized Learning of Personalized Models, 2022.

[24] Zexi Li, Jiaxun Lu, Shuang Luo, Didi Zhu, Yunfeng Shao, Yinchuan Li, Zhimeng Zhang, Yongheng Wang, and Chao Wu. Towards Effective Clustered Federated Learning: A Peer-to-peer Framework with Adaptive Neighbor Matching. *IEEE Transactions on Big Data*, pages 1–16, 3 2022.

[25] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. *Advances in Neural Information Processing Systems*, 30, 2017.

[26] Mi Luo, Fei Chen, Dapeng Hu, Yifan Zhang, Jian Liang, and Jiashi Feng. No Fear of Heterogeneity: Classifier Calibration for Federated Learning with Non-IID Data. *Advances in Neural Information Processing Systems*, 34:5972–5984, 12 2021.

[27] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data, 4 2017.

[28] Ishan Misra and Laurens van der Maaten. Self-Supervised Learning of Pretext-Invariant Representations. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 6706–6716, 12 2019.

[29] Xutong Mu, Yulong Shen, Ke Cheng, Xueli Geng, Jiaxuan Fu, Tao Zhang, and Zhiwei Zhang. FedProc: Prototypical Contrastive Federated Learning on Non-IID data. 9 2021.

[30] Mehdi Noroozi and Paolo Favaro. Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9910 LNCS:69–84, 3 2016.

[31] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation Learning with Contrastive Predictive Coding. *arXiv preprint arXiv:1807.03748*, 7 2018.

[32] Jayanth Reddy Regatti, Songtao Lu, Abhishek Gupta, and Ness Shroff. Conditional Moment Alignment for Improved Generalization in Federated Learning. In *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022)*, 12 2022.

[33] Tao Sun, Dongsheng Li, and Bao Wang. Decentralized Federated Averaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4 2022.

[34] Jiahao Tan, Yipeng Zhou, Gang Liu, Jessie Hui Wang, and Shui Yu. pFedSim: Similarity-Aware Model Aggregation Towards Personalized Federated Learning. 5 2023.

[35] Yue Tan, Guodong Long, Lu Liu, Tianyi Zhou, Qinghua Lu, Jing Jiang, and Chengqi Zhang. Fed-Proto: Federated Prototype Learning across Heterogeneous Clients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8):8432–8440, 5 2021.

[36] Thijs Vogels, Hadrien Hendrikx, and Martin Jaggi. Beyond spectral gap: The role of the topology in decentralized learning. 6 2022.

[37] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H Brendan Mcmahan, Blaise Agüera Y Arcas, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, Suhas Diggavi, Hubert Eichner, Advait Gadhikar, Zachary Garrett, Antonious M Girgis, Filip Hanzely, Andrew Hard, Chaoyang He, Samuel Horváth, Zhouyuan Huo, Alex Ingerman, Martin Jaggi, Tara Javidi, Peter Kairouz, Satyen Kale, Sai Praneeth Karimireddy, Jakub Konečn´ Konečn´y, Sanmi Koyejo, Tian Li, Luyang Liu, Mehryar Mohri, Hang Qi, Sashank J Reddi, Peter Richtárik, Karan Singhal, Virginia Smith, Mahdi Soltanolkotabi, Weikang Song, Ananda Theertha Suresh, Sebastian U Stich, Ameet Talwalkar, Hongyi Wang, Blake Woodworth, Shanshan Wu, Felix X Yu, Honglin Yuan, Manzil Zaheer, Mi Zhang, Tong Zhang, Chunxiang Zheng, Chen Zhu, Wennan Zhu, and Hong Kong. A Field Guide to Federated Optimization. 2021.

[38] Jian Xu, Xinyi Tong, and Shao-Lun Huang. Personalized Federated Learning with Feature Alignment and Classifier Collaboration. *11th International Conference on Learning Representations, ICLR 2023 - Conference Track Proceedings*, 2 2023.

[39] Valentina Zantedeschi, Aurélien Bellet, and Marc Tommasi. Fully Decentralized Joint Learning of Personalized Models and Collaboration Graphs, 6 2020.

[40] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stephane Deny. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12310–12320. PMLR, 3 2021.

[41] Jie Zhang, Song Guo, Xiaosong Ma, Haozhao Wang, Wencao Xu, and Feijie Wu. Parameterized Knowledge Transfer for Personalized Federated Learning. *Advances in Neural Information Processing Systems*, 13:10092–10104, 11 2021.