Report

MSc. Data Science & AI

Split-Federated Learning for Real-World Medical Imaging Applications

Marco Lorenzi, Francesco Cremonesi

Student: Aglind Reka

Contents

1. Introduction

1.1 Context and Motivation

Artificial Intelligence research in the medical domain has the potential to contribute to disease recognition, prognosis, and understanding disease progression. AI systems, employing advanced machine learning algorithms and data analytics, can analyze vast medical datasets, including patient records, genetic information, and imaging studies.[6] Ranging from the convenience of telemedicine to the precision of personalized medicine, technological advancements empower both patients and healthcare professionals. Predictive analytics facilitates proactive decision-making, while artificial intelligence can potentially improve diagnostic accuracy.[27]

The intricacies of medical applications and data access present inherent complexities. Securing access to essential training data constitutes a substantial challenge in the domain. Notable complexities arise from considerations such as data privacy and security, non-IID (Non-Independently and Identically Distributed) characteristics in local datasets, and adherence to legal and regulatory standards.[25, 13, 21]

Within the domain of medical data analysis, recent years have seen the investigation and advancement of various methods designed to tackle the challenge of data privacy. Several research initiatives aim to establish secure channels for data access, optimizing the retrieval of raw information while maintaining privacy and avoiding excessively high costs.[15, 22, 24]

1.2 Challenges

In the medical data domain, challenges encompass data privacy and security, non-IID characteristics in local datasets, compliance with legal and regulatory standards, and scalability issues. Since medical data is highly valuable and challenging to obtain, stringent laws and regulations are in place to safeguard sensitive data principles.[25, 3, 16, 21, 14]

Legal and regulatory standards are crucial in governing data practices, setting guidelines for collection, storage, processing, and sharing. Adherence to these standards is essential across different domains, ensuring ethical conduct, protecting sensitive data, and minimizing legal risks. Challenges in data privacy and security encompass unauthorized access, data breaches, cybersecurity threats, compliance with regulations, data residency and sovereignty concerns, third-party risks, user authentication and authorization issues, and data lifecycle management complexities [2, 3, 17].

Non-IID data, which is closely associated with the heterogeneity in patient populations, diverse data sources, variations in data collection methods, imbalanced class distributions, and challenges in collaborative research. When utilizing data from different sources, encountering the issue of heterogeneous data is commonplace. [13, 23].

In this paper, the primary focus will be on the topics of data privacy and data heterogeneity.

2. Review of the state of the art

Federated Learning [15] is a noteworthy approach addressing data privacy problem by enabling model training without directly accessing raw data. In federated learning, the model is sent to decentralized devices such as user devices for local training using their respective data. The only data sent back to the central server are training artifacts such as gradients or model parameters, which aggregates them to update the global model.

Split Learning [24], is a noteworthy approach addressing the problem of data heterogeneity and data privacy. The model is divided into two parts: the initial layers are kept on the client side, and the top layers are kept on the server side. The raw data remains on the client side, and only the representations from the initial layers are transmitted to the server.

Recently, new methods addressing the strengths and weaknesses of Federated Learning and Split Learning have emerged. In the paper ”Exploiting Shared Representations for Personalized Federated Learning” [5], the authors introduce FedRep, a federated learning framework and algorithm specifically designed to learn a shared data representation across clients while facilitating personalized updates.

Another innovative method is Splitfed [22], a fusion of Federated and Split Learning. This hybrid technique offers a secure method for training a model over private data without direct observation, leveraging the strengths of both approaches to enhance privacy and utility.

2.1 Federated Learning

Federated learning is a collaborative learning technique enabling users to benefit from shared models trained on diverse data without centralized storage. The approach involves a decentralized network of clients coordinated by a central server. Each client possesses a local training dataset, never shared with the server. Instead, clients compute updates to the global model, and only these updates are transmitted. [11, 4]

In federated learning, data distribution occurs through various scenarios, with two notable ones being “cross-silo” and “cross-device”. In the ”cross-silo” method, collaboration happens among distinct data silos or servers. Conversely, the “cross-device” method involves collaboration among individual devices, each contributing its local data to the federated learning process. The choice between these approaches hinges on the specific privacy, security, and collaboration requirements of the federated learning application. [4, 15, 9] In the ”cross-silo” scenario, the number of clients is limited, while in the ”cross-device” scenario, the number of clients can be substantial.

In federated learning, two primary coordination architectures are prevalent: “aggregator-based” and “peer-to-peer”. Aggregator-based federated learning uses a centralized architecture with an aggregator coordinating the learning process. The aggregator collects and aggregates model updates from participating clients, making it suitable for scenarios with numerous clients. However, privacy concerns arise as the central aggregator has access to all updates.[15]

Peer-to-peer federated learning employs a decentralized architecture where participating entities (peers) communicate directly. Each peer serves as both a learner and a contributor, sharing model updates without a central aggregator. This model provides enhanced privacy and potential scalability benefits, particularly in scenarios with direct peer communication. [19]

Federated learning finds applications in the healthcare industry for disease prediction, disease understanding, research, and precision medicine. [26, 20, 12, 18]

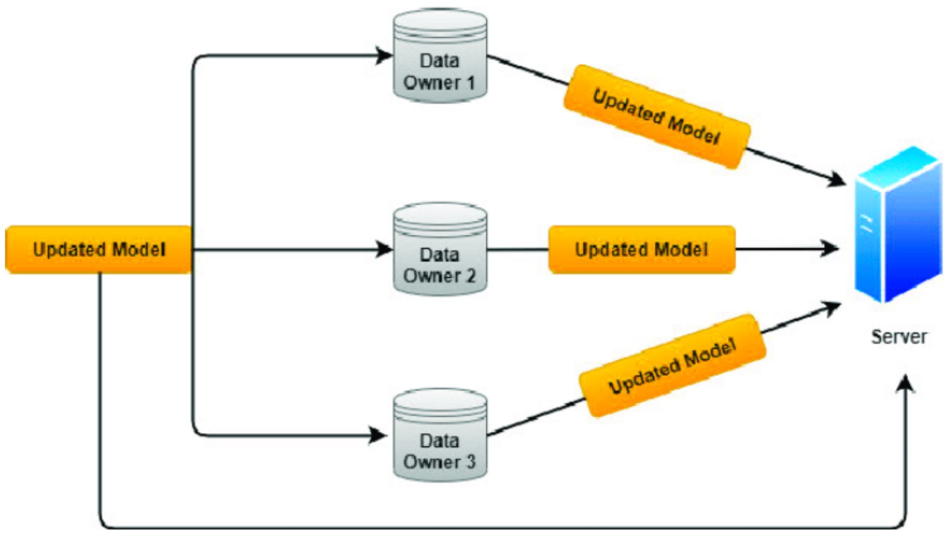


Figure 1: Federated Learning architecture in which the client and server communicate, sharing gradients at every step. Image from [1]

In federated learning, a significant challenge is the presence of data heterogeneity among client tasks, where the underlying data distributions vary substantially. If the server and clients attempt to learn a single shared model by minimizing average loss, the resulting model may exhibit poor performance for many clients in the network and may not generalize effectively across diverse datasets.[8]

2.1.1 Communication-Efficient Learning of Deep Networks from Decentralized Data

The seminal paper, "Communication-Efficient Learning of Deep Networks from Decentralized Data,"[15] was among the pioneering works to introduce and analyze the concept of federated learning. In the paper the authors present a method known as FederatedAveraging algorithm to enable the learning of a shared model without the need to centrally store the sensitive data. By using FederatedAveraging, the authors show that it is possible to train deep networks on decentralized data while reducing the rounds of communication needed and maintaining privacy.

To apply this approach in the federated setting, they select a C- fraction of clients on each round, and compute the gradient of the loss over all the data held by the clients. Thus, C controls the global batch size, with C = 1 corresponding to full-batch (non-stochastic) gradient descent.

A typical implementation of FedSGD as described in [15] with $C = 1$ and a fixed learning rate η has each client k compute

$$g_k = \nabla F_k\left(w_t\right), \tag{1}$$

the average gradient on its local data at the current model w_t, and the central server aggregates these gradients and applies the update

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k, \tag{2}$$

since

$$\sum_{k=1}^{K} \frac{n_k}{n} g_k = \nabla f\left(w_t\right). \tag{3}$$

An equivalent update is given by

$$\forall k, w_{t+1}^{k} \leftarrow w_t - \eta g_k \tag{4}$$

and then

$$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^{k}. \tag{5}$$

Each client locally takes one step of gradient descent on the current model using its local data, and the server then takes a weighted average of the resulting models. Once the algorithm is written this way, they add more computation to each client by iterating the local update

$$w^k \leftarrow w^k - \eta \nabla F_k\left(w^k\right) \tag{6}$$

multiple times before the averaging step.

2.2 Split learning

Split Learning is a proposed method that addresses the challenges of data privacy and data heterogeneity.

Split learning is a machine learning approach that segments a neural network model, distributing these segments across various devices or servers. This strategy allows for model training without the need to transmit raw data to a central server. The core concept of split learning involves conducting computations on local data up to a designated point in the model, referred to as the “cut layer.” Subsequently, only the intermediate representations or features are transmitted to a central server for additional processing and model refinement. [10, 7]

Label sharing poses challenges in split learning. Specifically, the original setup requires sharing of labels with the central server, which in the medical domain may contain sensitive or private information about a subject.

2.2.1 Split learning for health: Distributed deep learning without sharing raw patient data

In [7] the authors introduce SplitNN, a method that operates as previously described, involving the training of partial deep networks on client devices, transmitting intermediate outputs to a central server, and facilitating model updates through the exchange of gradients.

The authors introduce multiple configurations of Split Neural Networks (splitNN) tailored to diverse health settings.

Simple vanilla configuration for split learning: In this configuration, the deep neural network is bifurcated into two segments. The initial segment is trained on the client side, while the subsequent segment is trained on the server side. The outcome of the client-side training is transmitted to the server, where it is amalgamated with the output of the server-side segment to generate the ultimate output. This setup is advantageous for distributed deep learning scenarios characterized by horizontally partitioned data, where each client possesses a subset of the overall data.

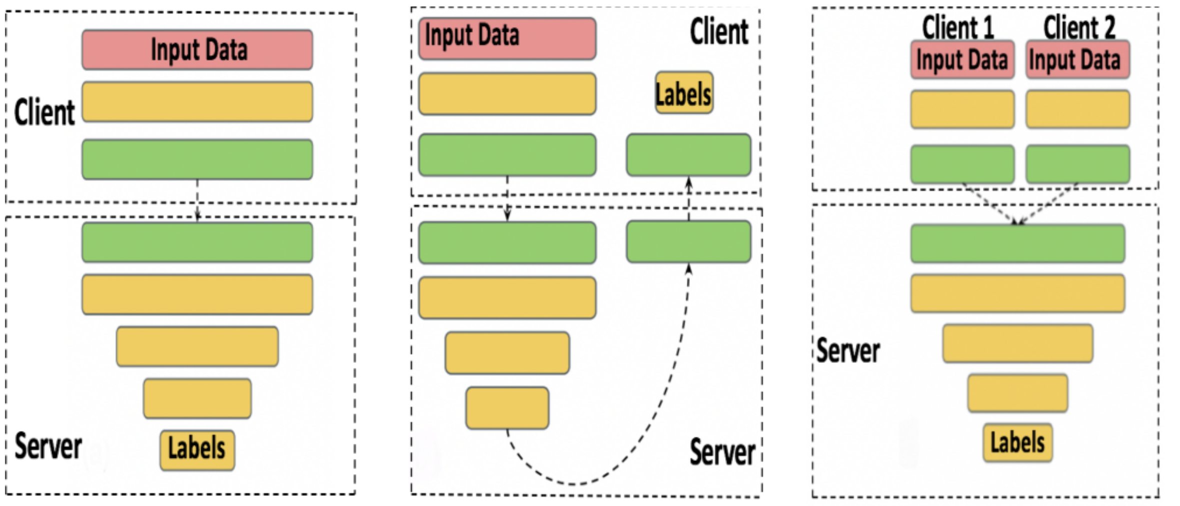
U-shaped configurations for split learning without label sharing: The paper investigates U-shaped configurations for split learning without label sharing. In this setup, the network wraps around the end layers of the server's network, and the outputs are sent back to client entities. Despite the server keeping the majority of its layers, clients generate gradients from the end layers for backpropagation without the need to share corresponding labels. This configuration is advantageous for distributed deep learning, particularly in scenarios where labels involve highly sensitive information, such as the disease status of patients.

Vertically partitioned data for split learning: In this configuration, each client independently trains a partial deep network on its respective data. The outputs at the designated cut layer are transmitted to the server. The server consolidates these outputs from all clients and finalizes the remainder of the training without inspecting raw data from the clients. In this approach, the data pertains to the same clients, yet it involves different features. Gradients are backpropagated at the server, extending from its last layer to the cut layer, and the gradients specifically at the cut layer are relayed back to the clients for the subsequent stages of backpropagation.

SplitNN requires fewer rounds of communication and less computation per round compared to FL. This is because in SplitNN, only the output of the last layer of each part is shared with a central server, whereas in FL, the model parameters are shared between the clients and the server.

SplitNN can handle heterogeneous data and models, while FL performs best with homogeneous data and models.

In SplitNN, the raw data and model details are not shared between the collaborating entities, which reduces the risk of data breaches and privacy violations.



(a) Simple vanilla split learning (b) Split learning without label sharing (c) Split learning for vertically partitioned data

Figure 2: Split learning configurations for health shows raw data is not transferred between the client and server health entities for training and inference of distributed deep learning models with SplitNN. Image and caption from [7]

2.3 Federated Representations

A significant challenge in federated learning is data heterogeneity, as previously discussed. Learning a single shared model through methods such as minimizing average loss may result in poor performance for many clients, restricting generalization across diverse data.[8]

In [5] the authors introduce FedRep, a federated learning framework and algorithm designed to learn a shared data representation across clients while accommodating personalized updates. This addresses the issue of objective inconsistency in heterogeneous federated optimization, arising from differing data distributions and objectives among clients.

Both the server and clients work collaboratively to learn the parameters of the global representation, while each individual client focuses on learning its unique local head locally. FedRep achieves this by alternately performing client updates and server updates during each communication round. The algorithm functions as follows:

Client Update. On each round, a constant fraction $r \in (0, 1]$ of the clients is selected to execute a client update. During the client update, client i performs τ local gradient-based updates to determine its optimal head based on the current global representation ϕ^t communicated by the server. Specifically, for $s = 1, \ldots, \tau$, client i updates its head as follows:

$$h_i^{t,s+1} = \mathrm{GRD}\left(f_i\left(h_i^{t,s}, \phi^t\right), h_i^{t,s}, \alpha\right),$$

where $\mathrm{GRD}(f, h, \alpha)$ is generic notation for an update of the variable h using a gradient of function f with respect to h and the step size α. The key is that client i makes many local updates, τ is chosen to be large to enable client i to iteratively update its head based on its local data, aiming to find the optimal head given the most recent representation ϕ^t received from the server.

Server Update. Once the local updates with respect to the head h_i are completed, the client contributes to the server update by performing a single local gradient-based update with respect to the current representation. In other words, the client computes

$$\phi_i^{t+1} \leftarrow \mathrm{GRD}\left(f_i\left(h_i^{t,\tau}, \phi^t\right), \phi^t, \alpha\right).$$

then transmits ϕ_i^{t+1} to the server, where the local updates are averaged to calculate the subsequent representation ϕ^{t+1}.

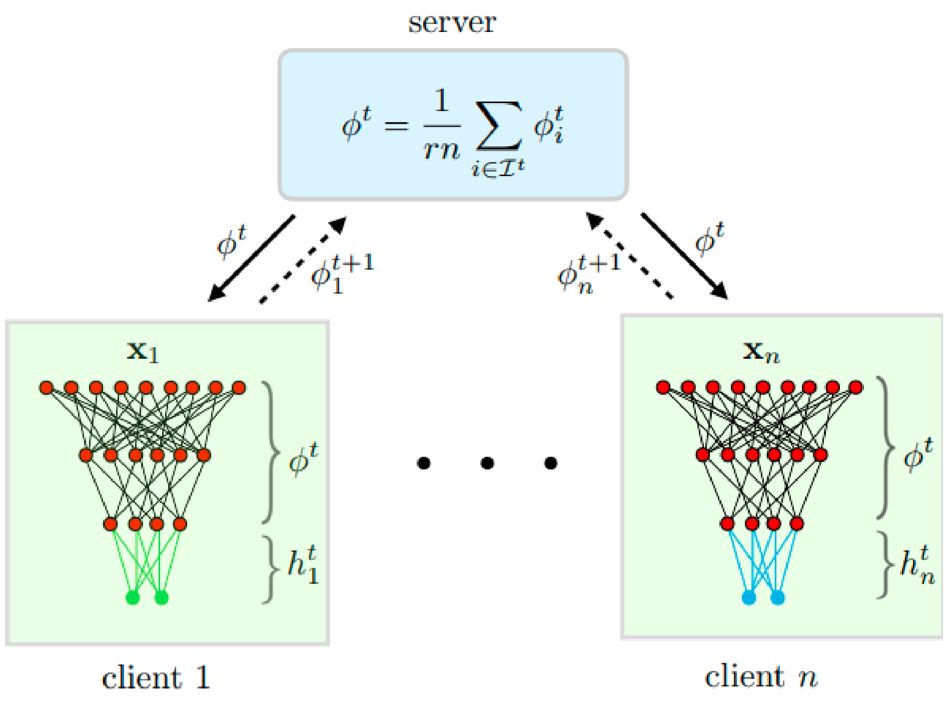


Figure 3: Federated representation learning structure where clients and the server aim at learning a global representation together, while each client i learns its unique head hi locally. Image and caption from [5]

3. Common latent space with fedAvg method

3.1 Model: latent space model with fedAvg

The models I'm employing aim to uphold the privacy of individual clients by leveraging a combination of federated learning and split learning techniques. Designed to tackle challenges surrounding data heterogeneity and privacy, the model operates in a decentralized manner. Each client partakes in training all models, with only a shared latent space among them. Periodically, typically every two epochs, the clients exchange parameters of the latent space and conduct federated averaging (fedAvg) on these parameters. It's essential to note that clients do not share raw data; instead, only the parameters of the latent space are exchanged between them. This approach ensures privacy while facilitating collaborative model improvement across distributed datasets.

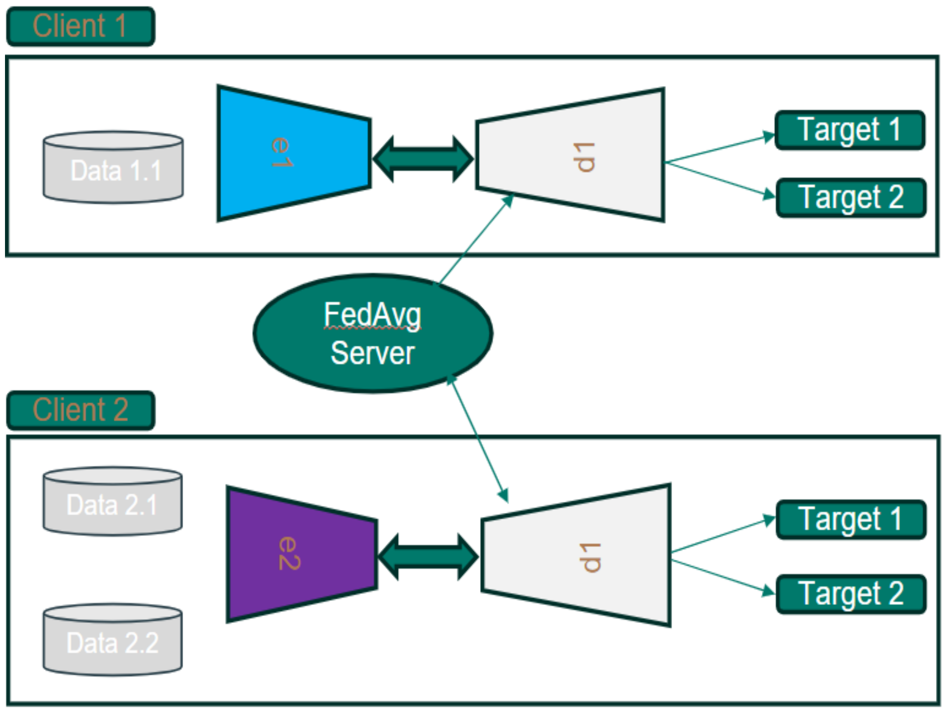


Figure 4: The latent model. This figure presents our proposed approach, whereby we split each clients' model in a private and shared subset. The private subset is evaluated only locally and its weights are never shared. The shared subset is also evaluated locally, but its weights are regularly averaged through a FedAVG-like scheme. Since the private subset of the model receives the input data, our approach enables us to integrate different data types for each client.

In the example depicted in Figure 4, two independent clients with distinct data structures are selected. The models are structured into two parts: an encoder and a decoder space which is the latent space. Importantly, only the latent space is shared between the clients. We implement federated averaging (fedAvg) on the parameters of this shared space, wherein all parameters are summed and averaged across clients.

Subsequently, the same weights are sent to all clients to update this component of the model. Through this approach, we aim to mitigate data heterogeneity and safeguard data privacy across the distributed clients.

3.1.1 Algorithm:

In the method, we have the updating model parameters across multiple clients. It initializes each client's parameters θ_i consisting of weights w_i per the decoder part and weights v_i for the encoder. Over K steps, gradients g_w and g_v are computed and used to update weights. After T epochs, global weights w are updated through federated averaging. Finally, each client initializes parameters for the next round with the updated global weights. F is the loss.

Algorithm 1: Algorithm description. The method involves updating model parameters across multiple clients through iterative steps, employing federated averaging to update global weights after a specified number of epochs, and initializing each client's parameters for the next round with the updated global weights.

```
Input: $\theta_i = \{w_i, v_i\}, F(X, \theta_i)$
Output: $\theta_i^{k+1} = \{w_i^{k+1}, v_i^{k+1}\}$
Client update:;
$\theta_i^0 = \{w, v_i^K\}$;
for $K$ steps do
    $g_w = \nabla_w L(x_{B_k}, \theta_i^k)$;
    $g_v = \nabla_v L(x_{B_k}, \theta_i^k)$;
    $w^{k+1} = w^k - \lambda g_w$;
    $v^{k+1} = v^k - \lambda g_w$;
    $\theta_i^{k+1} = \{w_i^{k+1}, v_i^{k+1}\}$;
end
(Every $T$ epoch) $w = \frac{1}{N} \sum w_i^K$;
```

3.2 Experimental Details

3.2.1 Synthetic datasets

I utilize Gaussian distributions with two distinct mean and covariance parameters so that I can have two different distributions of data for the two clients. The idea is to generate synthetic data tailored for a classification task. This process is facilitated by the sklearn.datasets library, which efficiently generates data sets suitable for classification tasks.

Subsequently, I apply four distinct transformations to the generated data, encompassing both linear and non-linear transformations. The details are in the appendix section(A)

The objective behind these transformations is to diversify the representations of the data and assess how effectively my model can discern patterns within them, thereby enhancing its performance in classifying the data.

As depicted in Figure 6, it becomes evident that separating the data proves

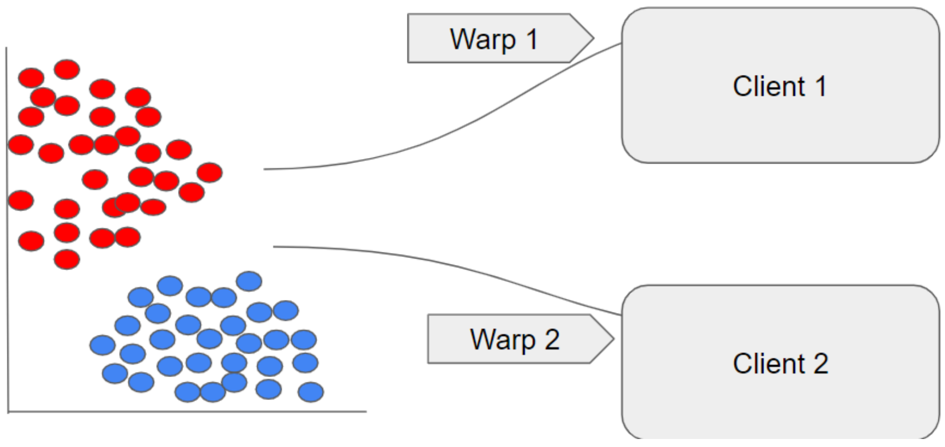


Figure 5: Data Generation Process: The dataset is generated from Gaussian distributions with distinct means. Subsequently, the data undergoes a transformation before being distributed to the clients for training.

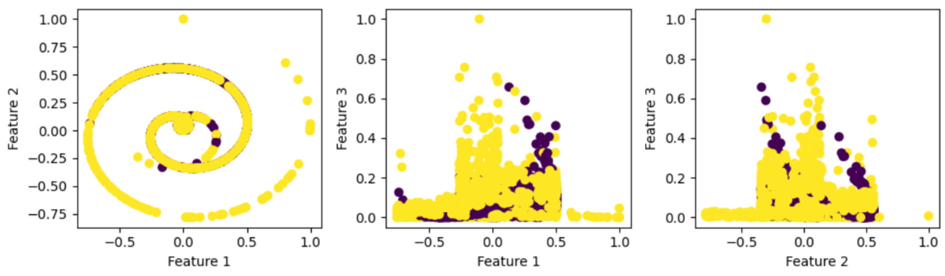


Figure 6: Swiss roll projected.

challenging for the model.

3.2.2 Task, model, and optimizer

The task at hand involves a classification task where clients collaborate to classify data accurately. Features and labels are generated using the `make_classification` function from two different Gaussian distributions with distinct means. The ML model architecture comprises an encoder and decoder model implemented as a sequential neural network in PyTorch. This architecture includes multiple linear layers followed by rectified linear unit (ReLU) activation functions, with the final layer utilizing a softmax activation function. With a total of five layers, the model has an input dimensionality of 3 and an output dimensionality of 3. For optimization, the Adam optimizer is employed, known for its adaptability in adjusting learning rates for each parameter during training. The learning rate (lr) is set to 0.001 for the latent model and SplitNN, 0.0001 for FedAvg. The training utilizes the cross-entropy loss function, a standard choice for classification tasks. Furthermore, training for the latent model occurs over a specified number of epochs, exactly 200 epochs, for the SplitNN 300 epochs, and 600 epochs are needed for the FedAvg, with an averaging process applied every 2 epochs to promote stability and convergence in the training process.

4. Results

Based on the comparative analysis conducted between the latent model, SplitNN, and Federated Averaging (FedAVG) methods, the results suggest compelling advantages of the latent model across various experimental scenarios.

The assessment, conducted across five different random seeds, consistently revealed superior performance in terms of mean accuracy and standard deviation for the latent model compared to both SplitNN and FedAVG. These findings are underscored by the emphasis on the best accuracy achieved by the latent model in all experiments, as highlighted in the presented table 1.

Moreover, the convergence analysis depicted in Figure 7 further reinforces the latent model's superiority. Notably, the latent model demonstrated significantly faster convergence, requiring approximately 30 epochs to reach convergence compared to over 200 epochs needed by SplitNN and more than 600 for FedAVG. To converge of the two models FedAvg and SplitNN takes a loot of more time in hours than Latent Model. This expedited convergence signifies not only enhanced efficiency but also potentially reduced computational costs associated with training.

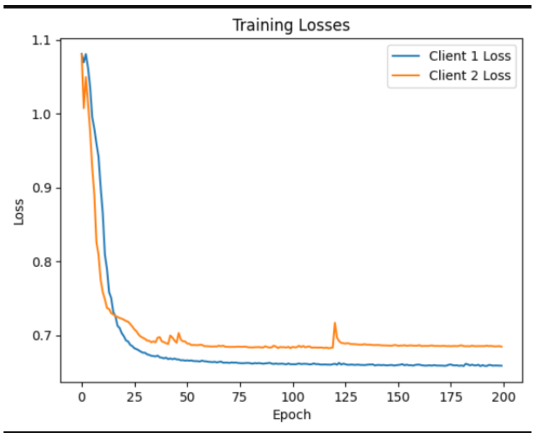
In summary, the results underscore the latent model's efficacy in various experimental setups, offering superior accuracy and faster convergence compared to alternative methods such as SplitNN and FedAVG. These findings substantiate the latent model's viability as a promising approach for addressing complex learning tasks efficiently and effectively.

Table 1: Results. The table showcases the mean accuracy and standard deviation achieved across five distinct random seeds. The best accuracy is emphasized in bold font.

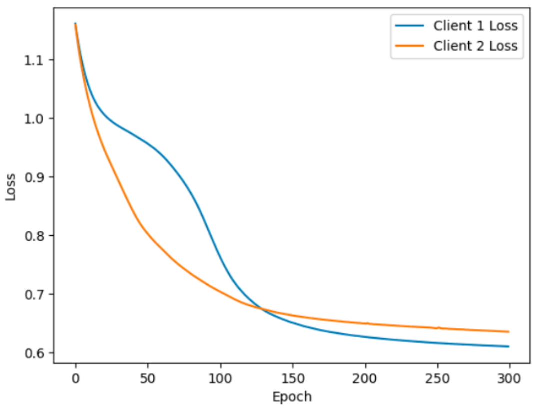
	Latent Model		SplitNN		FedAVG	
Functions	Client 1	Client 2	Client 1	Client 2	Client 1	Client 2
Strong non linear	**0.90±0.05**	**0.86±0.04**	0.89±0.04	0.85±0.03	0.87±0.02	0.78±0.04
Swiss roll	**0.69±0.06**	**0.82±0.08**	0.56±0.05	0.66±0.0.15	0.55±0.05	0.66±0.16
Linear	**0.93±0.04**	**0.89±0.04**	0.92±0.05	0.66±0.20	0.92±0.04	0.83±0.03
Mix/Lin-Non linear	**0.93±0.04**	**0.93±0.04**	0.92±0.04	0.92±0.04	0.89±0.04	0.9±0.05

4.1 Difference between the three models

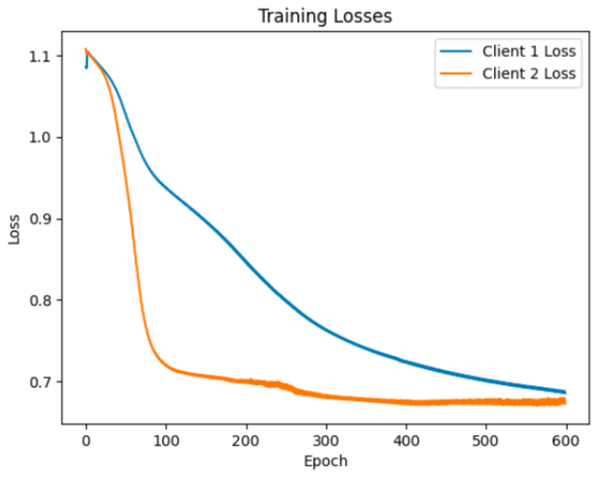
In FedAvg, I utilized 600 epochs and adjusted the learning rate to address oscillations in the loss curve, recognizing the necessity of a smoother learning rate. Although reducing the learning rate improved stability, achieving convergence required more than 600 epochs. Conversely, for SplitNN, I simply increased the learning rate to 300 to ensure model convergence.



((a)) Latent Model Convergence



((b)) SplitNN Model Convergence



((c)) FedAVG Model Convergence

Figure 7: Comparison of Model Convergence: As depicted in the preceding images, our model demonstrates superior convergence performance when compared to the state-of-the-art methods, namely fedAvg and SplitNN.

5. Conclusions

In this study, we conducted a comparative analysis of the latent model, SplitNN, and Federated Averaging (FedAVG) methods to assess their performance across various experimental scenarios. Our findings suggest the latent model's superiority over established approaches in terms of mean accuracy and convergence speed.

The latent model consistently outperformed SplitNN and FedAVG across different experimental setups, achieving better accuracy in all experiments. Additionally, our convergence analysis revealed that the latent model required approximately 50 epochs to converge, compared to over 300 epochs needed by SplitNN and more than 600 epochs by FedAVG, indicating faster convergence and potentially reduced computational costs.

These findings have significant implications for the advancement of machine learning. The latent model's superior performance makes it a compelling choice for tasks requiring high accuracy and efficiency, with potential applications in image recognition, natural language processing, and healthcare analytics.

However, it's important to acknowledge the limitations of our study. Further research is needed to explore the latent model's performance under different conditions and datasets (true medical ima), as well as to address scalability issues and improve interpretability and robustness.

In conclusion, the latent model holds promise for driving innovation and transformation in machine learning research and application. As we continue to refine its capabilities, it has the potential to enhance decision-making and automation across various domains.

A. Appendix

A.1 Strong non linear

The transformation function `warp1` operates on a 2D array `a`, where each row represents a coordinate pair (x, y). Firstly, the function separates the x and y coordinates from the input array. Then, it performs three distinct transformations on these coordinates:

1. `out1`: Computes the element-wise product of the x and y coordinates.
2. `out2`: Involves adding the x coordinate with an exponentially decaying function of the y coordinate.
3. `out3`: Calculates a combination of logarithmic, exponential, and polynomial operations applied to the x and y coordinates.

These transformations are designed to warp the input coordinate pairs into three distinct output components. After computing these components, the function stacks them together along the last axis to form the output array.

In contrast, `warp2` applies a different set of transformations to the input coordinates:

1. `out1`: Computes the natural logarithm of the squared product of the x and y coordinates, augmented with a term involving the product of x, y, and the sine function applied to x.
2. `out2`: Subtracts a scaled product of the x and y coordinates from the y coordinate.
3. `out3`: Calculates the product of the x, y coordinates, and the tangent function applied to the y coordinate.

Similar to `warp1`, the function stacks these transformed components together along the last axis to create the output array. These transformations in `warp2` result in different output components compared to `warp1`, showcasing varied warping effects on the input coordinate pairs.

A.2 Swiss Roll

For `warp1`:

The function `warp1` takes a 2D array a as input, where each row represents a coordinate pair (x, y). It performs the following transformations:

1. It calculates a parameter t based on x, adjusted with a factor related to the maximum value of x.
2. `out1` is determined as t times the cosine of t.
3. `out2` is computed similarly as t times the sine of t.
4. `out3` is obtained by scaling y squared by a factor of 21 and normalized to its maximum value.

These transformations aim to distort input coordinates (x, y) to produce varying effects on the output components.

For `warp2`:

On the other hand, `warp2` performs the following transformations:

1. `out1` is calculated as y minus 5 times the product of x and y.
2. `out2` involves 10 times y squared added to the fourth column of the input array.
3. The third column of the input array is retained for the output.

Both functions return the stacked output components normalized by their maximum values. These transformations aim to distort input coordinates (x, y) to produce varying effects on the output components.

A.3 Linear

For `warp1`:

The function `warp1` takes a 2D array a as input, where each row represents a coordinate pair (x, y). It performs the following transformations:

1. `out1` is set to the input x-coordinate x.
2. `out2` is set to the input y-coordinate y.
3. `out3` is computed as the sum of x and y.

These transformations aim to manipulate input coordinates (x, y) to produce varying effects on the output components.

For `warp2`:

On the other hand, `warp2` performs the following transformations:

1. `out1` is calculated as the input x-coordinate x scaled by a large constant.
2. `out2` is calculated as the input y-coordinate y scaled by the same large constant.
3. `out3` is computed as the difference between the scaled x-coordinate x and the scaled y-coordinate y.

Both functions return the stacked output components. These transformations aim to scale and manipulate input coordinates (x, y) to produce varying effects on the output components.

A.4 Mix of Linear and non-Linear functions

For `warp1`:

The function `warp1` takes a 2D array a as input, where each row represents a coordinate pair (x, y). It separates the x and y coordinates from the input array. Then, it performs the following transformations:

1. `out1`: It computes the sum of the x and y coordinates, representing their combined effect.
2. `out2`: This transformation scales the x coordinate by a factor of 10, emphasizing its influence in the output.
3. `out3`: It calculates the difference between the y and x coordinates, capturing the relative difference between them.

For `warp2`:

On the other hand, `warp2` applies a different set of transformations to the input coordinates:

1. `out1`: It calculates the difference between the y and 5 times the x coordinate, which shifts the y coordinate relative to the x coordinate.
2. `out2`: This transformation scales the y coordinate by a factor of 10, amplifying its contribution to the output.
3. `out3`: It computes the sum of the y coordinate and the square of the x coordinate, introducing a quadratic effect of the x coordinate on the output.

Both functions return the stacked output components. These transformations aim to distort input coordinates (x, y) to produce varying effects on the output components.

References

[1] Mohammed Aledhari et al. “Federated learning: A survey on enabling technologies, protocols, and applications”. In: *IEEE Access* 8 (2020), pp. 140699–140725.

[2] Jozef Andraško, Matúš Mesarčík, and Ondrej Hamuľák. "The regulatory intersections between artificial intelligence, data protection and cyber security: challenges and opportunities for the EU legal framework". In: *AI & SOCIETY* (2021), pp. 1–14.

[3] George J Annas. "HIPAA regulations: a new era of medical-record privacy?" In: *New England Journal of Medicine* 348 (2003), p. 1486.

[4] Keith Bonawitz et al. "Towards federated learning at scale: System design". In: *Proceedings of machine learning and systems* 1 (2019), pp. 374–388.

[5] Liam Collins et al. "Exploiting shared representations for personalized federated learning". In: *International conference on machine learning*. PMLR. 2021, pp. 2089–2099.

[6] Marzyeh Ghassemi et al. "A review of challenges and opportunities in machine learning for health". In: *AMIA Summits on Translational Science Proceedings* 2020 (2020), p. 191.

[7] Otkrist Gupta and Ramesh Raskar. "Distributed learning of deep neural network over multiple agents". In: *Journal of Network and Computer Applications* 116 (2018), pp. 1–8.

[8] Yihan Jiang et al. "Improving federated learning personalization via model agnostic meta learning". In: *arXiv preprint arXiv:1909.12488* (2019).

[9] Peter Kairouz et al. "Advances and open problems in federated learning". In: *Foundations and Trends® in Machine Learning* 14.1–2 (2021), pp. 1–210.

[10] Juyong Kim et al. "SplitNet: Learning to semantically split deep networks for parameter reduction and model parallelization". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1866–1874.

[11] Tian Li et al. "Federated learning: Challenges, methods, and future directions". In: *IEEE signal processing magazine* 37.3 (2020), pp. 50–60.

[12] Wenqi Li et al. "Privacy-preserving federated brain tumour segmentation". In: *Machine Learning in Medical Imaging: 10th International Workshop, MLMI 2019, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 13, 2019, Proceedings 10*. Springer. 2019, pp. 133–141.

[13] Xiaodong Ma et al. "A state-of-the-art survey on solving non-IID data in Federated Learning". In: *Future Generation Computer Systems* 135 (2022), pp. 244–258.

[14] Ahmad Akmaluddin Mazlan et al. "Scalability challenges in healthcare blockchain system—a systematic review". In: *IEEE access* 8 (2020), pp. 23663–23673.

[15] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[16] Rebecca T Mercuri. "The HIPAA-potamus in health care data security". In: *Communications of the ACM* 47.7 (2004), pp. 25–28.

[17] Daniel Povey, Xiaohui Zhang, and Sanjeev Khudanpur. "Parallel training of deep neural networks with natural gradient and parameter averaging". In: *arXiv preprint arXiv:1410.7455* (2014), p. 124.

[18] Holger R Roth et al. "Federated learning for breast density classification: A real-world implementation". In: *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning: Second MICCAI Workshop, DART 2020, and First MICCAI Workshop, DCL 2020, Held in Conjunction with MICCAI*

2020, Lima, Peru, October 4–8, 2020, Proceedings 2. Springer. 2020, pp. 181–191.

[19] Abhijit Guha Roy et al. “Braintorrent: A peer-to-peer environment for decentralized federated learning”. In: *arXiv preprint arXiv:1905.06731* (2019).

[20] Micah J Sheller et al. “Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data”. In: *Scientific reports* 10.1 (2020), p. 12598.

[21] Shafaq Siddiqi et al. “Detecting Outliers in Non-IID Data: A Systematic Literature Review”. In: *IEEE Access* (2023).

[22] Chandra Thapa et al. “Splitfed: When federated learning meets split learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 8. 2022, pp. 8485–8493.

[23] Willem G Van Panhuis et al. “A systematic review of barriers to data sharing in public health”. In: *BMC public health* 14.1 (2014), pp. 1–9.

[24] Praneeth Vepakomma et al. “Split learning for health: Distributed deep learning without sharing raw patient data”. In: *arXiv preprint arXiv:1812.00564* (2018).

[25] Paul Voigt and Axel Von dem Bussche. “The eu general data protection regulation (gdpr)”. In: *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10.3152676 (2017), pp. 10–5555.

[26] Jie Xu et al. “Federated learning for healthcare informatics”. In: *Journal of Healthcare Informatics Research* 5 (2021), pp. 1–19.

[27] Fan Zhang et al. “Deep learning based segmentation of brain tissue from diffusion MRI”. In: *NeuroImage* 233 (2021), p. 117934.