Bachelor-Praktikum Federated Learning and Distributed Systems

Chair of Data Systems - SS 2025

Nikita Agrawal

30. April 2025

Agenda

Table of Contents

There are different types of Machine Learning

Machine Learning is a kind of "data fitting".
We want to find for a given Input $x \in X$ and Output $y \in Y$ the model parameters $w \in W$ of a model $h(\cdot; w)$, so that holds:

$$h(x; w) = y$$

There are different types of Machine Learning

Machine Learning is a kind of "data fitting".
We want to find for a given Input $x \in X$ and Output $y \in Y$ the model parameters $w \in W$ of a model $h(\cdot; w)$, so that holds:

$$h(x; w) = y$$

There are different types of machine learning (ML). Commonly, there are 3 types:

- Supervised Machine Learning (e.g., Classification, Regression)

There are different types of Machine Learning

Machine Learning is a kind of "data fitting".
We want to find for a given Input $x \in X$ and Output $y \in Y$ the model parameters $w \in W$ of a model $h(\cdot; w)$, so that holds:

$$h(x; w) = y$$

There are different types of machine learning (ML). Commonly, there are 3 types:

- Supervised Machine Learning (e.g., Classification, Regression)
- Unsupervised Machine Learning (e.g., Clustering)

There are different types of Machine Learning

Machine Learning is a kind of "data fitting".
We want to find for a given Input $x \in X$ and Output $y \in Y$ the model parameters $w \in W$ of a model $h(\cdot; w)$, so that holds:

$$h(x; w) = y$$

There are different types of machine learning (ML). Commonly, there are 3 types:

- Supervised Machine Learning (e.g., Classification, Regression)
- Unsupervised Machine Learning (e.g., Clustering)
- Reinforcement Learning (e.g., Robot Navigation)

Table of Contents

Neuron

A classic example is an Artificial Neural Network (ANN), which is used for image classification (e.g., 'Cat or dog?').

Neuron

A classic example is an Artificial Neural Network (ANN), which is used for image classification (e.g., 'Cat or dog?').
For this task, we need an ANN that recognizes the pattern that distinguish between cat and dog.

Neuron

A classic example is an Artificial Neural Network (ANN), which is used for image classification (e.g., 'Cat or dog?').
For this task, we need an ANN that recognizes the pattern that distinguish between cat and dog.

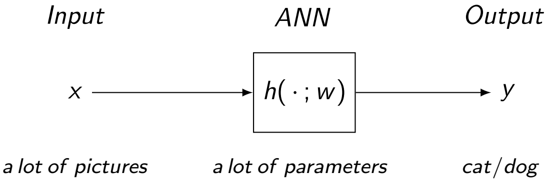

Abbildung 1: Schematic illustration of an ANN.

A (ffw) ANN

An artificial neuron process values provided by preceding neurons and passes (when activated) a signal to downstream neurons:

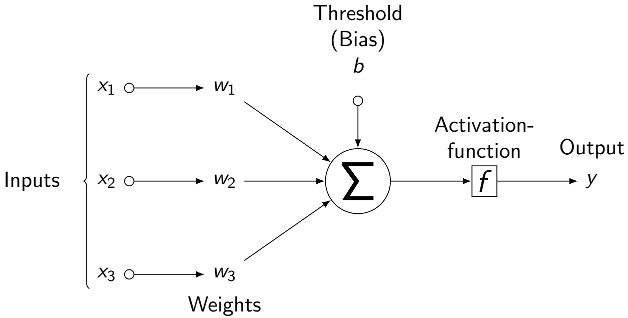

Abbildung 2: An artificial neuron as function of a weighted sum.

Graphic from [1]

Activation Functions

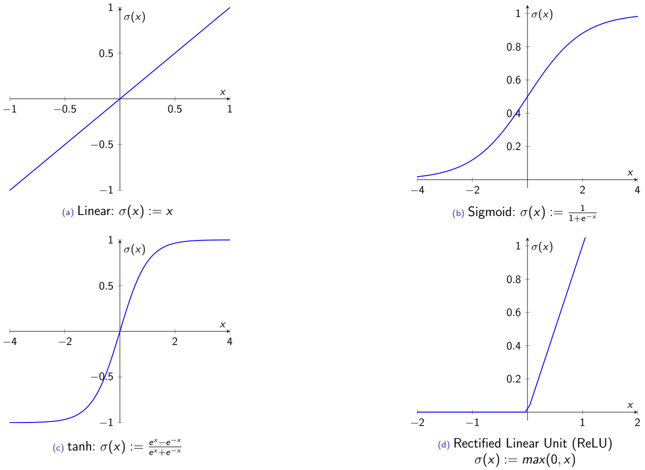

Abbildung 3: Activation functions determine, when a neuron should 'fire'

ANN for Classification

We wire lots of these neurons to form a (feed-forward (ffw)) neural network:

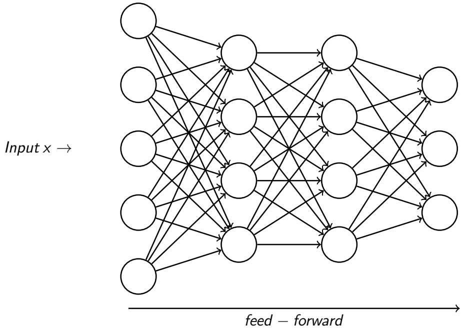

Abbildung 4: A (ffw) ANN. The inputs' information is unidirectional, forwarded through the network.

Convolutional Neural Networks (CNN)

For more complex data, like 2D-arrays (e.g., images), it is apparently more expedient to use Convolutional Neural Networks (CNNs).

Graphic from [11]

Convolutional Neural Networks (CNN)

For more complex data, like 2D-arrays (e.g., images), it is apparently more expedient to use Convolutional Neural Networks (CNNs). This is to bridge the gap between small networks that are incapable of processing the amount of data, and too big networks that tend to overfit [5].

Graphic from [11]

Convolutional Neural Networks (CNN)

For more complex data, like 2D-arrays (e.g., images), it is apparently more expedient to use Convolutional Neural Networks (CNNs). This is to bridge the gap between small networks that are incapable of processing the amount of data, and too big networks that tend to overfit [5].

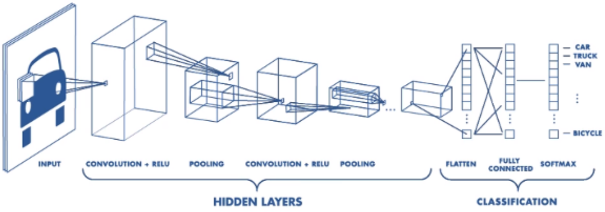

Abbildung 5: Scheme of a CNN, trained for image recognition.

Graphic from [11]

Table of Contents

The loss function measures the error

In Supervised Learning data is labeled (cat / dog) and we are to tell the model, whether the classification was performed correctly. (In Unsupervised Learning, the network checks this itself.)

The loss function measures the error

In Supervised Learning data is labeled (cat / dog) and we are to tell the model, whether the classification was performed correctly. (In Unsupervised Learning, the network checks this itself.)

The error is quantified with the help of the loss-function, this information we can now tell the model.

Typical candidates for loss functions would be

- Mean Square Error (MSE) [7]

$$\ell_{MSE}(h, y) := \frac{1}{n} \sum_{i=1}^{n} (h(x_i; w) - y_i)^2$$

Typical candidates for loss functions would be

- Mean Square Error (MSE) [7]

$$\ell_{MSE}(h, y) := \frac{1}{n} \sum_{i=1}^{n} (h(x_i; w) - y_i)^2$$

- Cross-Entropy-Loss [6] (binary Classification):

$$\ell_{CE}(p_{o,c}) := - \sum_{c=1}^{C} \beta_{o,c} \ln(p_{o,c})$$

here, C are the classes, β some binary indicator (e.g., ± 1) and $p_{o,c}$ are the predicted probabilities that an object o belongs to class c.

Table of Contents

Optimizer

We now define an optimizer, which tries to solve our problem, by minimizing the loss function's error.

Optimizer

We now define an optimizer, which tries to solve our problem, by minimizing the loss function's error. Often this is realized by means of a

Gradient Decent (GD) method (cf. module 2). Here, iteratively a step direction & step size is processed in the solution space (i.e., a gradient), and the next iteration step is performed.

Optimizer

We now define an optimizer, which tries to solve our problem, by minimizing the loss function's error. Often this is realized by means of a

Gradient Decent (GD) method (cf. module 2). Here, iteratively a step direction & step size is processed in the solution space (i.e., a gradient), and the next iteration step is performed.

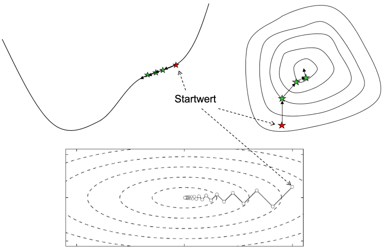

Abbildung 6: Schema Gradient Descent [9]

Training the Network

To tell the error/ deviation to the network, in order to train it, we have to transport it backwards (iteratively) through the network.

Training the Network

To tell the error/ deviation to the network, in order to train it, we have to transport it backwards (iteratively) through the network. This process is called **backpropagation**.

$$v \xrightarrow{?} \partial_w h(\,\cdot\,; w)^T v$$

Thereby, the weights (& biases) of the network get tuned:

Training the Network

To tell the error/ deviation to the network, in order to train it, we have to transport it backwards (iteratively) through the network. This process is called **backpropagation**.

$$v \xrightarrow{?} \partial_w h(\,\cdot\,; w)^T v$$

Thereby, the weights (& biases) of the network get tuned:

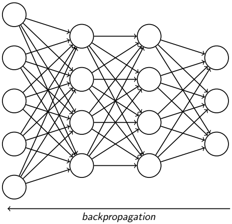

Abbildung 7: The deviation of the prediction is measured and used to tune weights & biases. The chain-linked dependencies of the neurons makes this an iterative transport all through the network to the input layer. We call that *backpropagation*.

Table of Contents

We represent data using tensors

A tensor is a multilinear mapping, and a generalization of known maps. There are several orders.

We represent data using tensors

A tensor is a multilinear mapping, and a generalization of known maps. There are several orders.

A tensor of...

- 0-th order is a scalar (scalar multiplication: $a * \vec{v}$)
- 1-st order is a vector (vector multiplication)
- 2-nd order is a matrix (matrix $*$ vector)
- 3-rd and higher order we call simply 'tensor'

By means of tensors present operations / data

E.g., the calculation of the values of the k-th layer in a ffw NN is representable as matrix-vector-multiplication:

By means of tensors present operations / data

E.g., the calculation of the values of the k-th layer in a ffw NN is representable as matrix-vector-multiplication:

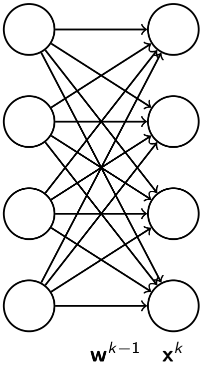

Abbildung 8: Multiplication of the weights with the k-th layer values.

By means of tensors present operations / data

E.g., the calculation of the values of the k-th layer in a ffw NN is representable as matrix-vector-multiplication:

$$
\begin{aligned}
w_{11}^{k-1} * x_1^{k-1} + w_{12}^{k-1} * x_2^{k-1} + \ldots + w_{1n}^{k-1} * x_n^{k-1} &= y_1^k \\
w_{21}^{k-1} * x_1^{k-1} + w_{22}^{k-1} * x_2^{k-1} + \ldots + w_{2n}^{k-1} * x_n^{k-1} &= y_2^k \\
\vdots &= \vdots \\
w_{m1}^{k-1} * x_1^{k-1} + w_{m2}^{k-1} * x_2^{k-1} + \ldots + w_{mn}^{k-1} * x_n^{k-1} &= y_m^k
\end{aligned}
$$

By means of tensors present operations / data

E.g., the calculation of the values of the k-th layer in a ffw NN is representable as matrix-vector-multiplication:

$$
\begin{aligned}
w_{11}^{k-1} * x_1^{k-1} + w_{12}^{k-1} * x_2^{k-1} + \ldots + w_{1n}^{k-1} * x_n^{k-1} &= y_1^k \\
w_{21}^{k-1} * x_1^{k-1} + w_{22}^{k-1} * x_2^{k-1} + \ldots + w_{2n}^{k-1} * x_n^{k-1} &= y_2^k \\
\vdots \qquad &= \vdots \\
w_{m1}^{k-1} * x_1^{k-1} + w_{m2}^{k-1} * x_2^{k-1} + \ldots + w_{mn}^{k-1} * x_n^{k-1} &= y_m^k
\end{aligned}
$$

$$
\Longleftrightarrow \quad (w_{ij}^{k-1})_{\substack{i=1,\ldots,m \\ j=1,\ldots,n}} \cdot \mathbf{x}^{k-1} = \mathbf{y}^k
$$

(Py)Torch

Torch:

- originated in 2002
- written in Lua/C/C++
- fundamental tensor operations, specialized for Machine Learning, e.g., fundamental routines (*BLAS*), and more abstract matrix multiplications

(Py)Torch

Torch:

- originated in 2002
- written in Lua/C/C++
- fundamental tensor operations, specialized for Machine Learning, e.g., fundamental routines (*BLAS*), and more abstract matrix multiplications

PyTorch:

- originated in 2017
- written in Python/C++/CUDA
- Pipeline for Python

PyTorch's Datasets & Dataloader

PyTorch Datasets:

- Standard datasets,
  e.g., well known CIFAR-10 resp. CIFAR-100
  (*'Image Classification'*)
- cf. https://pytorch.org/vision/main/datasets.html

PyTorch's Datasets & Dataloader

PyTorch Datasets:

- Standard datasets,
  e.g., well known CIFAR-10 resp. CIFAR-100
  (*'Image Classification'*)
- cf. https://pytorch.org/vision/main/datasets.html

Dataloader:

- usually, we use Mini-batches during the training (smaller groups of data samples)
- these should be randomly picked anew each time
- Dataloader offer a high-level implementation for these purposes

Table of Contents

General Problem Definition

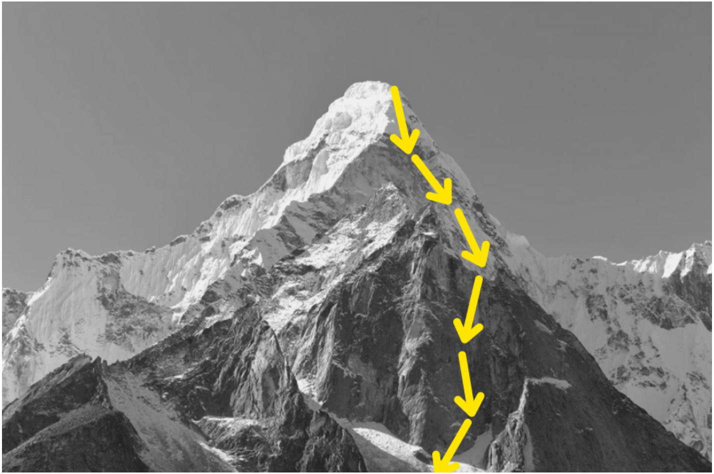

Abbildung 9: Illustration of the Gradient Descent [8]

General Problem Definition

Problem:

$$\min_{w \in W} F(w)$$

where W is a vector space with scalar product $< \cdot, \cdot >$ and norm $\sqrt{< \cdot, \cdot >}$.

General Problem Definition

Problem:

$$\min_{w \in W} F(w)$$

where W is a vector space with scalar product $< \cdot, \cdot >$ and norm $\sqrt{< \cdot, \cdot >}$.

Solution:

$$\min_{\Delta w \in W} F'(w)\Delta w + \frac{1}{2} < \Delta w, \Delta w > \tag{1}$$

(Δw is called *step direction*).

Gradient Descent (GD)

The gradient $\nabla F(w) = -\Delta w$ solves the problem above.

Gradient Descent (GD)

The gradient $\nabla F(w) = -\Delta w$ solves the problem above.

With the Gradient Descent method, we calculate this solution iteratively by means of the iteration

$$w_{k+1} = w_k + \delta w_k$$

and that way, find the Minimizer of (1).

($\delta w_k := \alpha \Delta w_k$; $\alpha \in]0, \infty[$ is called *step size*).

Stochastic Interpretation

The pair (x_i, y_i) correspond to a random sample of a random variable on $X \times Y$, which obeys a certain distribution [7].

Stochastic Interpretation

The pair (x_i, y_i) correspond to a random sample of a random variable on $X \times Y$, which obeys a certain distribution [7].
Wanted: the expected value

$$\mathbb{E}[\ell(h(x;w),y))] = ... = \mathbb{E}[f(w)]$$

$\ell(\cdot)$ is the loss function of ANN, cf. module 1.

Stochastic Interpretation

The pair (x_i, y_i) correspond to a random sample of a random variable on $X \times Y$, which obeys a certain distribution [7].
Wanted: the expected value

$$\mathbb{E}[\ell(h(x; w), y))] = ... = \mathbb{E}[f(w)]$$

$\ell(\cdot)$ is the loss function of ANN, cf. module 1.

Since the expected value cannot be evaluated, it is approximated via the *Empirical Risk*
($\widehat{=}$ Arithmetic Mean) of the solution:

$$\frac{1}{N}\sum_{i=1}^{N} \ell(h(x_i, y_i); w) = \frac{1}{N}\sum_{i=1}^{N} f_i(w)$$

Stochastic Gradient Decent (SGD)

It turns out that this problem can be solved with the stochastic gradient. In the GD method we had:

$$w_{k+1} = w_k + \delta w_k$$

Stochastic Gradient Decent (SGD)

It turns out that this problem can be solved with the stochastic gradient. In the GD method we had:

$$w_{k+1} = w_k + \delta w_k$$

Now we define $-\nabla F(w) := \mathbb{E}[\Delta w] = \Delta \overline{w}$ and $\delta w := \alpha \Delta \overline{w}$. Then, the sequence

$$w_{k+1} = w_k + \delta w_k$$

solves the problem for the stochastic gradient - under the condition that the variance $\mathbb{V}[\Delta \overline{w}] = \mathbb{E}[||\Delta w - \Delta \overline{w}||^2]$ remains small!

Variance Reduction

"The variance plays a decisive role in the convergence of the SG method." [7]

1. dynamically increase the mini-batch size
2. aggregate gradients of past steps, e.g. via (arithmetic) mean

Schematic Comparison

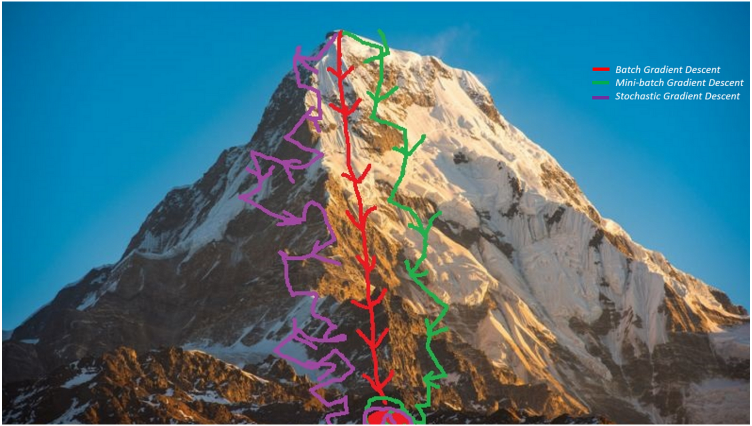

Abbildung 10: Illustration schema of different descent methods [2]

GD with Momentum

For faster Convergence, additionally a Momentum can be implemented ('Heavy-Ball-Method') - similar to a term for friction:
classic GD:

$$w_{k+1} = w_k + \delta w_k$$

Heavy-Ball:

$$w_{k+1} = w_k + \delta w_k + \beta(w_k + w_{k-1})$$

GD with Momentum

For faster Convergence, additionally a Momentum can be implemented ('Heavy-Ball-Method') - similar to a term for friction:
classic GD:

$$w_{k+1} = w_k + \delta w_k$$

Heavy-Ball:

$$w_{k+1} = w_k + \delta w_k + \beta(w_k + w_{k-1})$$

GD with Momentum

For faster Convergence, additionally a Momentum can be implemented ('Heavy-Ball-Method') - similar to a term for friction:
classic GD:

$$w_{k+1} = w_k + \delta w_k$$

Heavy-Ball:

$$w_{k+1} = w_k + \delta w_k + \beta(w_k + w_{k-1})$$

We won't go into detail here.

Table of Contents

Why Federated Learning?

'Data is the new Oil!

..

It's valuable, but if unrefined it cannot really be used.' [4]

Why Federated Learning?

'Data is the new Oil!

..

It's valuable, but if unrefined it cannot really be used.' [4]

Classic ML opened the race for the largest model to process the most data - a centralized 'supercomputer'.

Why Federated Learning?

'Data is the new Oil!
..
It's valuable, but if unrefined it cannot really be used.' [4]

Classic ML opened the race for the largest model to process the most data - a centralized 'supercomputer'.

→ Data needs to be shared and made available to a 'single point of trust' for processing if you want to use a service (which requires a lot of computing power/ storage space).

Why Federated Learning?

Federated Learning enables (collaborative) ML application & data analysis without data exchange:

Why Federated Learning?

Federated Learning enables (collaborative) ML application & data analysis without data exchange:

- Higher privacy due to local, unshared data

Why Federated Learning?

Federated Learning enables (collaborative) ML application & data analysis without data exchange:

- Higher privacy due to local, unshared data
- Collaboration, even with sensitive data possible ("cross-silo")

Why Federated Learning?

Federated Learning enables (collaborative) ML application & data analysis without data exchange:

- Higher privacy due to local, unshared data
- Collaboration, even with sensitive data possible ("cross-silo")
- Utilize available resources of mobile-/ edge-devices ("cross-device")

The idea of Federated Learning

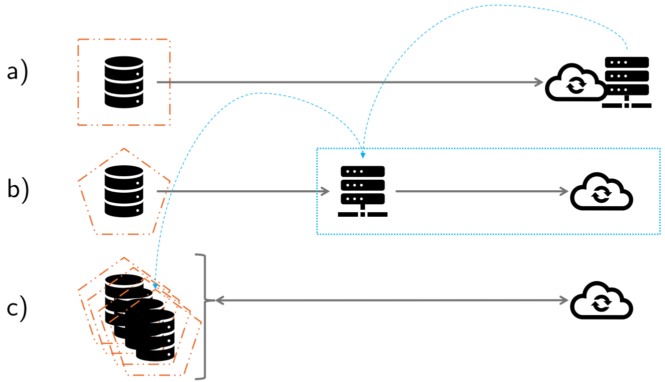

Abbildung 11: The development from centralized to decentralized machine learning (ML) to federated learning (FL)

The idea of Federated Learning

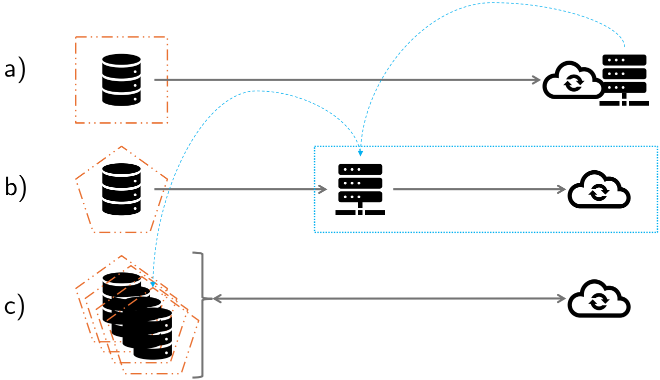

Abbildung 11: The development from centralized to decentralized machine learning (ML) to federated learning (FL) - a) classic ML with 'static' device and (decoupled) server

The idea of Federated Learning

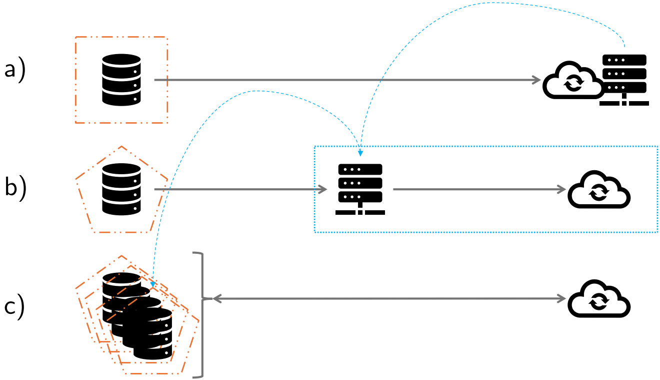

Abbildung 11: The development from centralized to decentralized machine learning (ML) to federated learning (FL) - a) classic ML with 'static' device and (decoupled) server - b) with mobile device (MD), data is stored centrally in classic ML (possibly independent of the model)

The idea of Federated Learning

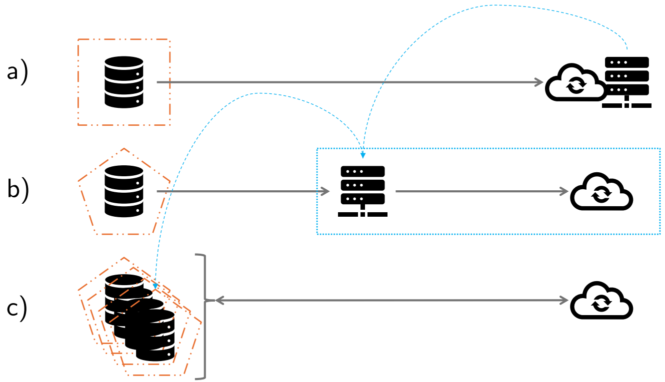

Abbildung 11: The development from centralized to decentralized machine learning (ML) to federated learning (FL) - a) classic ML with 'static' device and (decoupled) server - b) with mobile device (MD), data is stored centrally in classic ML (possibly independent of the model) - c) with numerous MDs in FL, the data remains local and only parameters are communicated periodically.

Utilize available resources...

...of *devices* / *clients* in distribtued systems.

Utilize available resources...

...of *devices / clients* in distribtued systems.

This reduces the memory and computing load on the *(Model / Parameter)-Server*. But the level of necessary communication is increasing.

Utilize available resources...

...of *devices* / *clients* in distribtued systems.

This reduces the memory and computing load on the *(Model / Parameter)-Server*. But the level of necessary communication is increasing.

Ideally, the number of (local) epochs and periodic communication with the server should be in balance with at least sustained efficiency,

Utilize available resources...

...of *devices* / *clients* in distribtued systems.

This reduces the memory and computing load on the *(Model / Parameter)-Server*. But the level of necessary communication is increasing.

Ideally, the number of (local) epochs and periodic communication with the server should be in balance with at least sustained efficiency, i.e., while we try to maximize the collaborative learning success, we want to communicate little as possible with the server.

Categories of Federated Learning

Federated Learning is typically grouped into Three Categories:

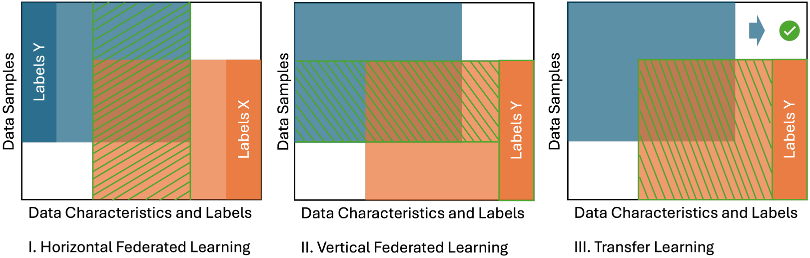

Abbildung 12: Categories of Federated Learning

Every Client Trains a Local Copy of the Model

On every of the m clients, a local copy of the global model is trained - similar to classic ML:

Every Client Trains a Local Copy of the Model

On every of the m clients, a local copy of the global model is trained - similar to classic ML:

$$F_k(w) = \frac{1}{n_k} \sum_{j_k=1}^{n_k} f_{j_k}(\,\cdot\,; w)$$

Formulation inspired by [10, 3]

Every Client Trains a Local Copy of the Model

On every of the m clients, a local copy of the global model is trained - similar to classic ML:

$$F_k(w) = \frac{1}{n_k} \sum_{j_k=1}^{n_k} f_{j_k}(\,\cdot\,; w)$$

where: $\forall k : p_k \geq 0$, $\sum_k p_k = 1$, n_k is the number of local data samples, i.e., *(local) Mini-Batch*,

Formulation inspired by [10, 3]

Every Client Trains a Local Copy of the Model

On every of the m clients, a local copy of the global model is trained - similar to classic ML:

$$F_k(w) = \frac{1}{n_k} \sum_{j_k=1}^{n_k} f_{j_k}(\,\cdot\,; w)$$

where: $\forall k : p_k \geq 0$, $\sum_k p_k = 1$, n_k is the number of local data samples, i.e., *(local) Mini-Batch*, f_{j_k} is the *loss function*, e.g.,

$$f_{j_k} = \ell_{CE}$$

F_k is the '*empirical risk* on local data' [10].

Formulation inspired by [10, 3]

Federated Average - Collaborative Learning

However, so that the clients (without data exchange) can benefit from the learning success of others, their model parameters are (periodically) sent to the server.

Federated Average - Collaborative Learning

However, so that the clients (without data exchange) can benefit from the learning success of others, their model parameters are (periodically) sent to the server.

The server tries to determine an average progress, usually by calculating an average of the gradients of the clients ($\rightarrow$ FederatedAverage (FedAvg)):

$$\min_{w} F(w) := \sum_{k=1}^{m} p_k F_k(w)$$

Federated Average - Collaborative Learning

However, so that the clients (without data exchange) can benefit from the learning success of others, their model parameters are (periodically) sent to the server.

The server tries to determine an average progress, usually by calculating an average of the gradients of the clients ($\rightarrow$ FederatedAverage (FedAvg)):

$$\min_{w} F(w) := \sum_{k=1}^{m} p_k F_k(w)$$

Please note: the global sample size n (*global batch size*) for the global training is the number of the overall mini-batches of the clients:

$$n = \sum_{k} n_k$$

Data Structure - The Non-IID Problem

Since, due to user behavior, network access / stability, ... the local data sets of the clients differ, it cannot be assumed that the data as a whole is equally distributed and independent. (*non-Independent, Identically Distributed Data* (non-IID))

Data Structure - The Non-IID Problem

Since, due to user behavior, network access / stability, ... the local data sets of the clients differ, it cannot be assumed that the data as a whole is equally distributed and independent. (*non-Independent, Identically Distributed Data* (non-IID))

This can be simulated by unevenly distribute the data over the clients.

Data Structure - The Non-IID Problem

Since, due to user behavior, network access / stability, ... the local data sets of the clients differ, it cannot be assumed that the data as a whole is equally distributed and independent. (*non-Independent, Identically Distributed Data* (non-IID))

This can be simulated by unevenly distribute the data over the clients.

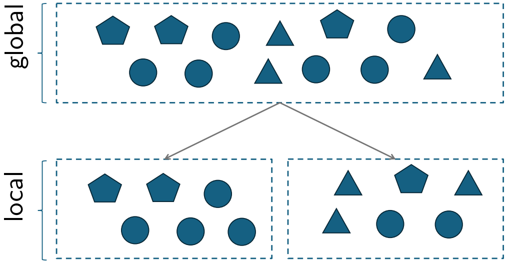

Abbildung 13: Schema: Simulation of non-IID data in Federated Learning.

Quellen

- S. 241621, *Diagramm of an artificial neural network*.
- BIOINFORMATICSANDME, *Types of gradient descent*. 2019.
- BRENDAN H. MCMAHAN, EIDER MOORE, DANIEL RAMAGE, SETH HAMPSON, BLAISE AGÜERA Y ARCAS, *Communication-efficient learning of deep networks from decentralized dat*, arXiv.org, (2023).
- M. P. CLIVE HUMBY, *Data is the new oil*, (2006).
- Y. LECUN, L. BOTTOU, Y. BENGIO, AND P. HA, *Gradient-Based Learning Applied to Document Recognition*, (1998).
- READTHEDOCS.COM, *Loss functions*. 2017.
- A. SCHIELA, *Optimization for machine learning*. 2020.
- A. SHARMA, *The hitchhiker's guide to optimization in machine learning*. 2021.
- F. K. THOMAS BROX, *Optimierung*, Vorlesungsfolien, (2013).
- TIAN LI, ANIT KUMAR SAHU, AMEET TALWALKARM, VIRGINIA SMITH, *Federated learning: Challenges, methods, and futur directions*, arXiv.org, (2019).
- C. YEOLA, *Convolutional neural network (cnn) in deep learning*.