EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

FEDERATED LEARNING OF ARTIFICIAL NEURAL NETWORKS

THESES OF THE PH.D. DISSERTATION

PÉTER KISS

SUPERVISOR: TOMÁŠ HORVÁTH, PH.D.

PH.D. SCHOOL OF COMPUTER SCIENCE

HEAD OF SCHOOL: PROF. ERZSÉBET CSUHAJ-VARJÚ, DSC

PH.D. PROGRAM OF INFORMATION SYSTEMS

HEAD OF PROGRAM: PROF. ANDRÁS BENCZÚR, DSC

DOI: 10.15476/ELTE.2021.124

AUGUST 2021, BUDAPEST

Contents

Abstract

The training of the most expressive state-of-the-art Machine Learning (ML) models, and especially that one of *Artificial Neural Networks* (NN) requires a huge amount of training data, along with very significant computation resources. The paradigm of *Federated Learning* (FL) focuses on the possibilities of collaborative training of ML models in heterogeneous, spatially distributed environment of recent IT infrastructures, dividing the burden of computation among all the stakeholders and leaving the potentially sensitive data at the location of its creation.

In this dissertation I am presenting my work relating to the possibilities to alleviate apparent problems of federated training of NNs. For the experienced performance loss we propose to adapt and use stateful optimization techniques for the FL setup. For communication complications of the centralized training we tested a technique to simulate FL in a peer to peer environment. And finally for the privacy issues we present a method to train NNs in an FL environment via a derivative free genetic algorithm.

Chapter 1

Contribution

In the research field of federated optimization, in our view, recent works are conducted with the goal in mind to alleviate some of the three main problems of FL algorithms from the original proposals in [129] and [157]. The first of these problems is the observed **performance degradation** of the learning process, or insufficient accuracy on the side of the end users. The second research direction is about practical questions of implementing these methods, such that resource provisioning, or more concretely, addressing the **communication problems** in real world network architectures. And the third group of works is seeking to provide **stronger privacy guaranties** for the users who participate with their data in the training process.

In my theses I propose simple techniques to alleviate the three above mentioned issues of FL.

Performance degradation The first problem we examined is the empirical fact that FL of NNs under-performs central training, both in the accuracy of final models, and in the rate of learning. According to our intuition this issue might be relieved at some extent by stateful optimization methods, that have been originally designed to overcome similar problems to those that arises from the characteristics of FL training. Based on our experiments [64, 126], presented in Chapter 4, these methods can help to alleviate the performance issues of FL in almost all the examined cases.

Communication bottleneck The most apparent issues with real world applicability of FL arise from centralized communication and coordination. As a possible solution for these problems, we developed and tested peer-to-peer-like asynchronous system for training an ensemble of models, that is introduced in Chapter 6. Our Migrating Models methods[125] were able, at least in our setup, to produce a similar performance to that one of FedAvg, however, significantly decreasing communication burden and computation need as well as avoiding network congestion.

Privacy issues One of the most important promise of FL is granting the owners of training data a stronger privacy, leaving the potentially sensitive data at its location of creation. As it is illustrated in Section 3.4 and Appendix A.6, this protection is far from complete, especially in the case of NNs. In our intuition, many of the privacy attacks can be excluded by eliminating the use of gradients, that can be achieved by training the target network by nature inspired derivative free methods. In Chapter 5 we test the possibilities of this, and provide a proof of concept that shows that evolutionary training can be applied for neural networks in the federated setup [208].

Federated Learning (FL) provides a perspective for ML, that is more fitting for the infrastructural challenges of the era of Edge computing and IoT. To serve processing, data storage and computation needs of sensor networks of small user devices, that should be carried out as close to the users as possible and modules of IT systems should be deployed flexibly. Thus, addressing problems related to building better preforming FL systems complements itself with our previous research, not presented in this thesis, that we did in the topic of self-organization of new generation software systems in Edge and Fog environments [127, 172, 173].

1.1 List of publications

My publications related to the scientific results presented in this dissertation:

1. G. Szegedi, P. Kiss, T. Horváth (2019): Evolutionary Federated Learning on EEG-data. ITAT 2019 Information Technologies – Applications and Theory, CEUR Workshop Proceedings Vol. 2473.

2. V. Felbab, P. Kiss, T. Horváth (2019): Optimization in Federated Learning. ITAT 2019 Information Technologies – Applications and Theory, CEUR Workshop Proceedings Vol. 2473.

3. P. Kiss, T. Horváth, V. Felbab (2020): Stateful Optimization in Federated Learning of Neural Networks. Proceedings of the 21st International Conference on Intelligent Data Engineering and Automated Learning 2020 - Part II, Lecture Notes in Computer Science, Vol. 12490

4. P. Kiss, T. Horváth (*In press*): Migrating models: A decentralized view on federated learning. Workshops of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2021), Proceedings

Further publications:

1. P. Kiss, D. Fonyó, T. Horváth (2018). BlaBoO: A Lightweight Black Box Optimizer Framework. Accepted to the World Symposium on Digital Intelligence for Systems and Machines (DISA) 2018.

2. P. Kiss, A. Reale, C.J. Ferrari, Z. Istenes (2018). Deployment of IoT applications on 5G Edge 2018 IEEE International Conference on Future IoT Technologies (Future IoT)

3. Reale, A., Kiss P. et al. (2018): Application Functions Placement Optimization in a Mobile Distributed Cloud Environment. Studia Universitatis Babeș-Bolyai Informatica, v. 63, n. 2

4. A. Reale, P. Kiss, M. Tóth, Z. Horváth (2019): Designing a decentralized container based Fog Computing framework for task distribution and management. Int. J. of Computers and Communications, vol. 13, NAUN.

1.2 Outline

The rest of the dissertation is organized in the following way: Chapter 2 summarizes the fundamental concepts of ML and NN training, along with the traditional methods of distributed ML. After these, in Section 2.2 of Chapter 2, the foundations of FL will be presented in more details. Chapter 3 is aimed at presenting related works. However, the immense amount of scientific literature that has been produced in the last years, since the publication of the seminal paper [130] that summarizes the mission of FL, makes this task impossible. Therefore, this chapter can be considered as a collection of interesting ideas which has been proposed by scientific community to tackle various aspects of the problems posed by FL.

Chapters 4, 5 and 6 present our own work that has been done to investigate different aspects of FL, where each of these are dedicated to one of the problems and ideas presented in the first part of this Chapter.

Mathematical background of the methods, algorithms, and concepts used in the thesis, along with some related algorithms and techniques are shortly summarized in the Appendix.

Chapter 2

Preliminaries

2.1 Machine Learning

According to the probably most cited definition of Machine Learning (ML) from *Tom M. Michel* [160], *"a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E"*. *Experience* in this definition appears in the form of some *training data*, that is fed to a *training algorithm*, through which the program, or rather the program's incorporated understanding of task, the *model* will be adapted to the observed world.

Solving many tasks of ML can be viewed as a branch of applied statistics that uses some previously observed data instances to create/approximate functions with the purpose to predict some missing y values of data points $\mathbf{x}$. These functions can be rule-based or statistical models. Most machine learning use cases can be described with Equation 2.1, where given the known values $\mathbf{x}$, we want to predict some missing value (label, class, cluster id,...) y, using a model m with parameters $\mathbf{w}$.

$$y = m(\mathbf{w}, \mathbf{x}) \tag{2.1}$$

Predicting this missing y value is referred to as *inference*. To be able to grant meaningful predictions, first we must fit the model m, what is called *training*. During the training we want to find the best possible parameters $\mathbf{w}$ of a model that have been chosen based on some prior belief on a solution.

The term "best possible" here means that the model makes few mistakes, in other words, that the error of the prediction (defined by a loss function f) is minimal across all the data points. The goal therefore is to find a model for the known *training data* that minimizes the loss function f defining how our learned model distribution differs from the empirical distribution. This measure, in general, can be formalized as a negative log likelihood

$$f = -\mathbb{E}_{\mathbf{x}\sim p_{data}}[log\, p_{model}(\mathbf{x})]. \tag{2.2}$$

That is, if a given example $\mathbf{x}$ is drawn from the training data distribution what is the probability that it will be present in the same form in the model distribution as well. If the model is used for predicting some value(s) y based on a vector of some attributes $\mathbf{x}$ this can be rewritten as

$$f(\mathbf{x}, y, \mathbf{w}) = -\log p(y|\mathbf{x}; \mathbf{w}). \tag{2.3}$$

The problem we want to solve is to minimize this loss function f, that is the aggregation of losses on all available data points, with respect to model parameters $\mathbf{w}$, as follows:

$$\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}), \text{ where } f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w}), \tag{2.4}$$

where $f_i(\mathbf{w}) \overset{\text{def}}{=} f(\mathbf{x_i}, y_i, \mathbf{w})$[1] denotes the loss on ith data point $\mathbf{x}_i$ given the parameters $\mathbf{w}$.

In our research work we focused on NN models that can be described, in general, as chaining together some non-linear (mostly point-wise) *activation functions* ϕ applied on an affine transformation of an input vector $\mathbf{x}$:

[1] When it does not make confusion, according to this definition, the notation f_i will be used to refer to the loss at the ith training example: $f_i(\mathbf{w}) = f(\mathbf{w}.\mathbf{x}_i, y_i)$

$$\mathbf{a}^{(0)} = \mathbf{x} \tag{2.5}$$

$$\text{for}^{i} = 1 \ldots h : \tag{2.6}$$

$$\mathbf{a}^{(i)} = \phi_i(\mathbf{W}^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}) \tag{2.7}$$

$$\hat{y} = \psi(\mathbf{a}^{(h)}), \tag{2.8}$$

where $\mathbf{W}^{(i)}$ stands for the weight matrix for the layer i of the network, $\mathbf{b}^{(i)}$ the vector of bias weights corresponding to each neuron, $\mathbf{a}^{(i-1)}$ are the *activations* of neurons from the previous layer, ϕ_i is the activation functions of the layer and ψ is the output activation function.

NN is the most popular ML model nowadays, first, due the end-to-end nature of learning (no need for feature engineering) and, second, because of its expressive power. According to the *Universal Approximation Theorems* [49, 152] NNs can represent a wide range of functions. What makes NN learning challenging, however, is that even if the above mentioned approximation model exist, it is far from being certain that we are being able to find it.

2.1.1 Training ML models: Stochastic Gradient Descent

When training ML models the problem we want to solve is to minimize the loss function f, with respect to model parameters $\mathbf{w}$, aggregated over the losses on all available data points, as we have already defined in Equation 2.4.

To solve this problem, due to the non-convex loss functions, the most popular methods are various versions of the Stochastic Gradient Descent (SGD). SGD takes derivatives of the loss function at one data point according to the parameters of the model and moves the parameter values in the negative direction of the gradient, i.e.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} f_i(\mathbf{w}_t), \tag{2.9}$$

where η_t denotes the "learning rate" which is, in the basic case, decaying during the learning to enforce convergence. In practice, instead of applying the gradient for each example, an average of gradients over a batch $\mathcal{B}$ of randomly chosen examples is used, that are evaluated at the same setting of $\mathbf{w}$. Such "minibatch" gradient descent (MBGD), still commonly referred to as SGD, exploits better the parallel computational capabilities of the hardware (like the GPU, for example). For both cases, the update is a stochastic approximation $\mathbb{E}[\nabla_{\mathcal{B}}(\mathbf{w})] = \nabla f(\mathbf{w})$ of the whole gradient.

2.1.2 Distributed ML

With the significant growth of available data, distributed implementations of training algorithms became necessary. There are two main direction in distributing the workload of ML training, *data parallelism* and *model parallelism*.

Model parallelism

Model parallelism might become necessary for the largest and most complicated models, such as extremely large NNs, where the number of parameters is too high to fit into GPU memory. The methods, that belong in this category, divide into pieces not only the training data but the model as well. These pieces can arise from splitting the model *vertically* and/or *horizontally*.

One of the simple but powerful approaches to tackle the lack of computational and storage capacity for processing the data, is to split our data into small chunks (see the next section on *Data parallelism*) and to train models on these parts, that are small enough for the machines to cope with. In 2012 the method of *multi-column deep neural networks* (MCDNNs) [46] reached state-of-the art performance on a large number of image classification tasks. The structure of MCDNNs can be understood as training relatively simple NNs (called *columns*) in parallel, and then, at the inference phase, averaging over the individual outputs. Thus the methods effectively results in an ensemble classifier [56]. Naturally, nothing prevents the columns to be trained on different machines, thus reducing the computational pressure on individual processing units.

In [53] the authors use a specialized NN architecture for parallel learning as well, introducing *Deep Stacking Networks*, where smaller, one hidden layer networks are stacked on top of each other, such that the output of the immediately previous module are concatenated with the original input, and these together will serve as the input for the actual module (Figure 2.1). This method enables to train huge, fully connected models on multiple processing units.

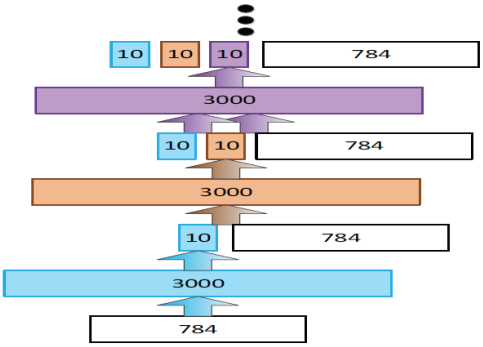


Figure 2.1 An asynchronous SGD [53] where the networks are split horizontally and each "layers" receives as input the original data along with the output of the underlying "layer".

DistBelief [51], the ancestor of Tensorflow has been introduced in 2012, developed for training and inference with deep networks that have billions of parameters. In contrast to its precursors, as [46] and [53], Distbelief does not make any restriction to the structure of networks, supporting distributing data sets and layers both horizontally and vertically, as it is indicated in Figure 2.2.

The Adam project [45] for large scale NN training describes an architecture with vertical splits that optimizes the inter-machine communication for Convolutional Neural Network (CNN) training (Figure 2.3).

Training such huge networks also involves tremendous amount of data, thus, in most cases model parallelism comes together with data parallel training as well (as happens in [51] and [45], too).

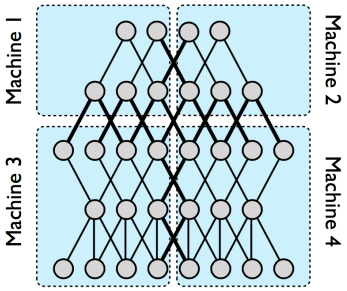


Figure 2.2 Model parallelism in [51].

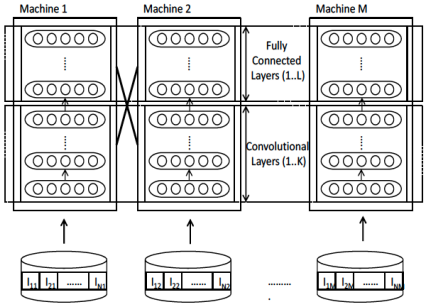


Figure 2.3 Model parallelism in [45].

Data parallelism

In the data parallel setup, which this work mostly focuses on, the data and workload are dedicated to multiple workers and the goal is to solve a consensus problem, that is, to find a model that fits for all the workers. Formally, we have a set of nodes $V = \{v^1, v^2, \ldots, v^K\}$ with $K = |V|$ and n data points allocated into sets $\mathcal{D}^k$ of indices of data points stored at nodes v^k ($1 \leq k \leq K$) with $n^k = |\mathcal{D}^k|$ being the number of data points at the node v^k. Without the loss of generality, we usually assume that $\mathcal{D}^k \cap \mathcal{D}^l = \emptyset$ whenever $l \neq k$, thus $n = \sum_{k=1}^{K} n^k$. In distributed setting, the local loss for node v^k can be defined as $f^{(k)}(\mathbf{w}) = \frac{1}{n^k} \sum_{i \in \mathcal{D}^{(k)}} f_i(\mathbf{w})$, changing the minimization problem, defined above in Equation (2.4), to

$$\min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}) = \min_{\mathbf{w} \in \mathbb{R}^d} \sum_{k=1}^{K} \frac{n^{(k)}}{n} f^{(k)}(\mathbf{w}). \tag{2.10}$$

Data parallelism originally addresses the problem of utilising multiple GPUs for updating the gradient computed from as many training example as possible.

In data parallel, distributed Gradient Descent (GD)-based systems the parameters are distributed across a bunch of worker nodes, that compute gradients on shards of data from the distribution, that has been assigned to them. The worker nodes then send these updates (gradients)back to the parameter server(s), which will be aggregate them to produce the new model (parameters). This loop is then repeated until some of the stopping criteria is met.

Thus one round of synchronous distributed SGD is equivalent with a taking a single mini-batch, and compute the gradient over it. In a centralized distributed synchronous MBGD training, where updates are computed over batches (not single examples) from the "local" data sets, the update looks like follows:

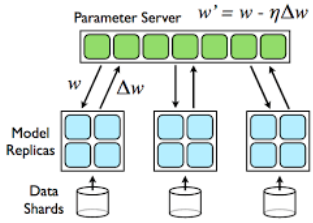
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \frac{1}{K} \sum_{k=1}^{K} \nabla_{\mathbf{w}} f_{\mathcal{B}_k}(\mathbf{w}_t), \tag{2.11}$$

where $\nabla_{\mathbf{w}} f_{\mathcal{B}_k}(\mathbf{w}_t)$ corresponds to the gradient over minibatch $\mathcal{B}_k$ computed at node k w.r.t. the recent model parameters $\mathbf{w}_t$. It is equivalent to using bigger batches (to gain less biased gradients), and eventually, if local updates are computed over the entire local data set, the method results in an unbiased gradient:

$$\nabla_{\mathbf{w}} f(\mathbf{w}) = \sum_{i=1}^{n} \nabla_{\mathbf{w}} f_i(\mathbf{w}) = \sum_{k=1}^{K} \sum_{j=1}^{n^{(k)}} \nabla_{\mathbf{w}} f_j^{(k)}(\mathbf{w}) = \sum_{k=1}^{K} \nabla_{\mathbf{w}} f^{(k)}(\mathbf{w}) \tag{2.12}$$

In practice collecting all the gradients before the update can slow down training, thus using asynchronous methods can result in empirically better performance. The above mentioned Distbelief [51] implements training using L-BFGS (see the Appendix A.2.6), and *Downpour SGD*. The latter is equivalent to an asynchronous distributed version of stochastic gradient descent (Equation 2.11), when a worker node fetches the most up-to-date parameters of the model, computes gradients and sends them back. This method, on one hand, solves the problems of stragglers and machine failures, however, on the other hand, brings the problem of stale gradients, that is, we can be almost always certain, that the nodes are computing their updates for a model that has been changed in the meanwhile by the updates of some other processes. Experiments have shown though, that applying the above approach to realistic scenarios in an insecure environment, in combination with *Adagrad asynchronous SGD*, can perform very well.

Figure 2.4 Asynchronous SGD in [51].



2.2 Federated learning

FL is dealing with an increasingly important distributed optimization setting that came into view with the spread of small user devices and ML related applications written for them. The domain of these ML models is often the data created on these devices, thus, one should incorporate it into the learning process as well.

Training ML models in a distributed way, in general, corresponds to solve a consensus problem in the following form:

$$\min_{\mathbf{w}} f(\mathbf{w}) \tag{2.13}$$

$$f(\mathbf{w}) = \sum_{k=1}^{K} \frac{n^{(k)}}{n} f^{(k)}(\mathbf{w}), \tag{2.14}$$

that is, to find a model with parameters $\mathbf{w}$ that minimizes the sum of the local loss $f^{(k)}$ for the K nodes, weighted by the proportion $n^{(k)}/n$ of all n data points a given kth node holds. This structure of the problem covers a very wide range of ML tasks from linear or logistic regression, through support vector machines (SVMs) to NNs.

The traditional way for this would be to transfer the information gathered at the users to data centers, where the training takes place (most probably in one of the ways described in the previous section), and the trained models are then sent back to the users. That, apart from the obvious privacy concerns, can incur a huge communication overhead, along with the need for significant storage and computational resources at the place of centralized training.

The idea proposed in [129] is, that instead of moving the training data to a centralized location, one could exploit the computational power residing at the user devices, keeping the data at the location of creation, and distribute the training process across the participating nodes.

This, however, brings more complication compared to the data center based distributed training (2.1.2). Due to the truly distributed nature of the system, communication becomes expensive and unreliable. Therefore, the Federated SGD (FedSGD) [129] method, introduced in the Algorithm 1, applies a modified version of aggregating the gradients (Equation 2.11) to reduce communication complications: instead of communicating the gradients per batch, the central updates takes place after multiple local updates:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \sum_{k=1}^{K} \frac{n^{(k)}}{n} \Delta_k, \text{ with } \Delta_k = \eta \sum_{i=0}^{r} \nabla f_{\mathcal{B}_{t_i}^k}(w_{t_i}^k), \tag{2.15}$$

where $\mathbf{w}_{\mathbf{t_{i+1}}}^{\mathbf{k}} = \mathbf{w}_{\mathbf{t_i}}^{\mathbf{k}} - \eta \nabla f_{\mathcal{B}_{t_i}^k}(\mathbf{w}_{t_i}^k)$, $\mathbf{w}_{t_0}^k = \mathbf{w}_t$ and r is a hyper-parameter for the number of local updates.

To further increase communication efficiency, the Federated Averaging (FedAvg) algorithm [157] takes only a small subset (10%) of updates. This method can be understood as compromise between the rather slow synchronous and the fast asynchronous version of training, as has been introduced in Section 2.1.2. However, as it has been empirically proven, this is not a real trade-off, since FedAvg is able to keep or, in some cases, even increase the convergence rate of learning. FedAvg became the baseline of FL research.

In the setup of FL, the characteristics of data distribution, from which our training examples $(\mathbf{x_i}, y_i)$ will be drawn, are the following:

1. *Massively Distributed*: Data points are stored across a large number K of nodes. In particular, the number of nodes can be much bigger than the average number n/K of training examples stored on a given node (where n is the number of all examples).

2. *Non-IID*: Data on each node may be drawn from a different distribution, i.e. the data points available locally are far from being a representative sample of the overall distribution.

3. *Unbalanced*: Different nodes may vary by orders of magnitude in the number of training examples they hold.

Algorithm 1 Federated SGD (FedSGD)

procedure SERVER
    initialize $\mathbf{w_0}$
    for $t = 0; 1; 2; \ldots$ **do**
        for all k in the K nodes in parallel **do**
            $\mathbf{w}_{t+1}^k \leftarrow \text{ClientUpdate}(k, \mathbf{w}_t)$
        end for
        $\mathbf{w}_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} \mathbf{w}_{t+1}^k$
    end for
end procedure
procedure CLIENTUPDATE(k, w)
    $\mathcal{B} \leftarrow$ split $\mathcal{D}^{(k)}$ to set of batches
    for all $b \in \mathcal{B}$ **do**
        $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f(\mathbf{w}, b)$
    end for
    return W
end procedure

Since the publication of the problem statement of FL [129], a lot of research has been carried out and the technology reached a fairly mature state having its support even in the Tensorflow framework [25] to facilitate real world development.

Chapter 3

Related work

In this chapter we first present the most important works on communication efficient distributed learning, that lead to the formulation of the problem of FL. After that we present related works, organized by which main problem they focus the most according to our previously introduced grouping (Chapter 1). Our researches focused on the federated training of NNs, but it might be worth to introduce some innovative approaches invented for the simpler convex problems as well.

3.1 Distributed Learning for Convex Problems

As the amount of data to be processed exceeded the storage capacities (or memory) of the most powerful machines, new training methods have been developed to support parallel processing.

In general this class of *data parallel* algorithms is aiming at training better models in fewer synchronisation steps, solving more-or less exactly the local problems through various local solvers and aggregation methods, exploiting the convexity of the loss function.

The one extremity of distributed training is one-shot averaging [245], where the local sub-problems are solved perfectly, then the global optimum is given by a single average step. The other end is parallel SGD [256] where, after every single per-data-point optimization step, the local updates are averaged. In [192], the performance and the fundamental

limitations of SGD-based methods between these two extreme have been studied in terms of runtime, communication costs and number of samples used. They found that the best convergence guarantees can be given for accelerated gradient descent [164] with the biggest possible batch sizes.

Over the family of SGD-based methods, under some conditions on the problem (mostly convexity of the loss), a range of other methods based on duality or curvature information can be used as well.

For solving the local problems *Distributed Approximate Newton (DANE)* DANE [193] and *Accelerated Inexact Dane (AIDE)* [174] uses local quadratic perturbation to turn local updates into a mirror descent step (Section A.2.5), while *Distributed Optimization for Self-Concordant Empirical Loss (DiSCO)* [247], aims at exploiting the super-linear convergence of Newton methods by implementing inexact damped Newton method(A.2.4,) for the distributed setting.

CoCoA [112] and CoCoA$^+$ [153] builds on the idea, that if training examples are dispersed across multiple worker nodes, then applying dual optimization is a natural choice ([112] proposes Stochastic Dual Coordinate Ascent, SDCA, A.2.2). In this setting, the different nodes are working on different subsets of dual variables α and the weight update $\Delta\mathbf{w}$ is eventually a linear combination $X^T\Delta\alpha$ of the refinements $\Delta\alpha_i$ of coefficients $\alpha_\mathbf{i}$ (dual variables) belonging to data points $\mathbf{x}_i$ (Appendix A.2.1).

CoCoA gives indeed stronger convergence properties as it has been shown in [112], however, in practice, for more complex problems (not necessarily convex) as [130] reports, it does not perform very well.

In the original FL paper [130], the authors proposed *Federated Stochastic Variance Reduced Gradient* method (FSVRG), a hybrid of CoCoA, DANE and SVRG for solving convex problems. The connections of DANE with SVRG have been analysed in [174] where the authors have shown that a modified version of DANE is in fact equivalent to a distributed version of SVRG. The key idea is to use the SVRG update loop in the inexact DANE to solve the local problems approximately. Moreover, as pointed out in [130], DANE with SVRG style local updates can also be interpreted as applying the idea

of CoCoA to fix the drawbacks of DANE, namely, solving the local sub-problems only to obtain a *relative accuracy*.

FSVRG has been tested for predicting whether a given Google+ post will generate comments, using regularized logistic regression. For this problem the Federated SVRG highly outperformed CoCoA+ and distributed Gradient Descent, however, the method is strongly tailored to the specific task.

Moreover, since FSVRG, such as CoCoA and other similar sophisticated methods, is building on the convexity of the objective function, the much simpler methods of FedSGD and FedAvg are used in most of the cases since these are empirically proven to be more efficient on a much bigger range of models. A more detailed description of these algorithms can be read in A.7.

3.2 Real world deployment

One direction of FL research is aiming at addressing the problems of deploying FL (FedAvg) scheme in existing IT infrastructures.

Soon after the formulation of the settings of FL and the first researches relating to FedSGD, a lot of research work started investigating the ways of reducing the obvious communication overhead, that is being involved by synchronized updates collected and computed at a single central parameter server. Several algorithms has been proposed to address this problem of communication bottleneck (not specifically for the FL setup but in general for *data parallelism*), that can be used to improve communication efficiency for its more general scenario.

The most visible problems of real world applicability of FedAvg stems from the centralized synchronous nature of the algorithm. Namely, centralization leads to congestion in the communication infrastructure, especially in the case of NNs, where not only the number of nodes that try to communicate with the server is large, but also the size of the models (or updates of equal size) to transmit is significant.

Recent approaches dealing with these challenges include

- attempts to apply *adaptive scheduling* techniques to release the synchronous nature of the process and the stemming slowdown;
- methods for reducing the amount of information, that must be communicated during the training via *quantization* or *pruning* methods;
- addressing the problems or even exploiting the characteristics of the communication medium as in *Federated Edge Learning* approaches

In this section we aim at giving some insight into this direction of FL researches.

3.2.1 Scheduling and resource provisioning

Bonawitz et. al [25] introduce the system design of a TensorFlow based production system for mobile devices as workers, and cloud based coordinators or *parameter servers* (PS), that already has been deployed over tens of millions of real world devices. The work addresses most issues appearing in practice, such that device availability, unreliable connectivity and interrupted execution. All these problems are addressed at the communication protocol that is illustrated in Figure 3.1.

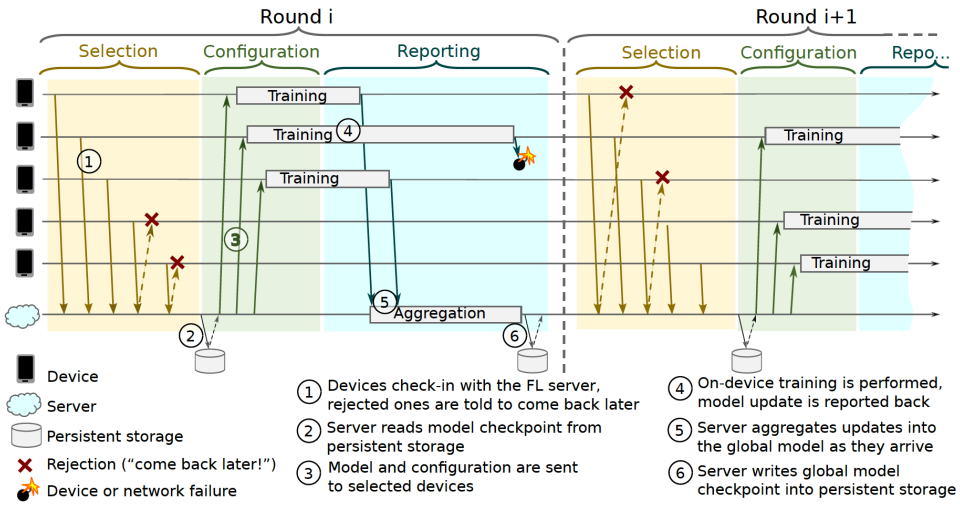


Figure 3.1 Training in [25]

The workers announce their availability for training time by time, and the PS decides whether they will participate in a training round or instruct them to reconnect at a given time

again. The system also creates holdout data sets of the nodes for validating the performance of the system locally. Both at assigning the nodes participating at a training and reporting the updates the nodes with late responses or no answers will be simply ignored.

This system handles the problems that come from the practical implementation of FedAvg without making significant changes of the algorithm. The rest of this section is dedicated to modifications some parts of the FL or, more generally, data-parallel ML algorithms to address the issues of communication and computational bottlenecks.

Update scheduling

The paper of Chen et al. [40] alleviate the problem of stale parameters inherent in Distbelief's [51] (Section 2.1.2) asynchronous SGD using b backup nodes. The proposed method use $N+b$ workers and, in each round, waits with the aggregation step until any N updates are received. Experiments have shown that this method leads to faster convergence and better model performance compared to the original method of Distbelief.

In [234] the authors make experiments with traditional scheduling policies for the node updates under constrained channel availability of wireless networks, taking into account inter-cell interference, resource allocation between the radio access links and also the stage of learning.The selection of nodes in the proposed Proportional Fair Policy is based on which nodes can be communicated with at the moment "better then usually". That is, we pick the nodes k with the highest value for $\frac{\tilde{\rho}_k}{\overline{\rho}_k}$, where $\tilde{\rho}_k$ is the instantaneous Signal-to-Interference-plus-Noise Ratio (SINR) and $\overline{\rho}$ is a moving average of SINR. They found that the *Proportional Fair Policy* outperforms the original random selection and the Round Robin as well, at least in the examined case of training Support Vector Machines (SVMs). On the other hand, for Convolutional NNs (CNNs), the picture was not this clear.

In contrast to stationary protocols, [224] presents an algorithm to dynamically adapt the communication frequency and, thus, the number of epochs at the nodes to achieve the lowest possible loss subject to a fixed budget of resources. Instead of making attempts on scheduling, similarly to the already mentioned *Downpour SGD* in *Distbelief* , in [44] an

asynchronous pattern was recommended, where updates computed for older models will still participate in training, however, with a reduced weight.

3.2.2 Compression

One of the most simple way to alleviate the challenges of communication is to reduce the amount of information to be transmitted per update round, that is, to compress the communicated information.

Gradient compression techniques for distributed SGD most frequently use some kind of quantization [189, 203, 60, 4, 5] for reducing the necessary communication bandwidth. Quantization usually involves different kind of pruning methods and, in general, aims at communicating only the most important aspects of the updates and keeping the less significant information on the source location until they become important enough.

In distributed training off NNs, the use of *Quantization methods* builds on the observation that large NN parameters are sparse, thus, the updates are, not surprisingly, sparse too. In general, they are viewed as a generalization of delayed update training, uploading the most important directions the model should be changed in. These methods usually accumulate gradient residuals for those coordinates that have not been sent, and when the aggregated magnitude grows beyond the threshold they will be updated. That means that less important parameters will be less frequently and less significantly updated, thus reducing the average update size.

A general method for quantization with residuals can be formulated as

$$\tilde{\nabla f_t} = \mathcal{Q}(\nabla f_t + \Delta f_{t-1}) \tag{3.1}$$

$$\Delta f_t = \nabla f_t - \mathcal{Q}^{-1}(\tilde{\nabla f_t}), \tag{3.2}$$

where $\tilde{\nabla f_t}$ stands for the quantized gradient at time t, $\mathcal{Q}$ for the quantization function and Δf_t denotes gradient residual/quantization error. The simplest way for specification of importance of coordinates is done simply by pruning along some thresholds τ by the magnitude of the value at the coordinate. Quantization achieves further often stronger

compression by discretizing the submitted parameters, replacing the actual value to be transmitted by a small integer, or even a single bit. This smaller value or bit then at receiver side will be translated into some actual magnitude, for example to τ Some interesting quantization methods are described shortly in Appendix A.9

In [133] the authors propose an asymmetric communication pattern to reduce the communication costs for image classification tasks with NN-s. The work builds on the observation according to which internet connections are set up in an asymmetric way, that is, the up-link is usually much slower then down-link. Thus, the main goal is to reduce the amount of data to be sent by the workers. The authors experimented with two main methods: The *Structured updates* method required the nodes to train and update only a subset of the nodes, that has been either picked by projection or by a random mask. The *sketched updates* method, on the other hand, asks the nodes to train all the parameters but in the end only a subset is required to be sent for aggregation, compressed either by probabilistic 1-bit quantization (above a threshold), or sub-sampled randomly. (For more details see A.8)

Du et al. [61] address the compression of weight updates of Bayesian NNs [169] that have been trained using *variational dropout* [123] (see Section A.5). Variational dropout training is equivalent with Bayesian back-propagation with local re-parametrization including a multiplicative noise. This noise is learnable on per weight base as it is demonstrated by [123] (Appendix A.5). In [61] the authors argue that since high variance of a parameter indicates large noise at that coordinate, pruning it can be beneficial from the perspective of the inference. According to this, by transmitting only the updates of parameters of low dropout rates saves communication cost without performance degradation.

[34] proposes an interesting FL scheme to connect dropout regularization to compression, that is reported to reduce communication costs to the $1/21$th along with 1.7 times reduction in local computation costs. The approach consist of two main components such as a new lossy compression method and a *federated dropout* method. Federated dropout zeros out a fixed number of activations at each layer, applying the same mask for each

clients, thus, it becomes possible to remove corresponding rows and columns of the weight matrices, as it is depicted in Figure 3.2.

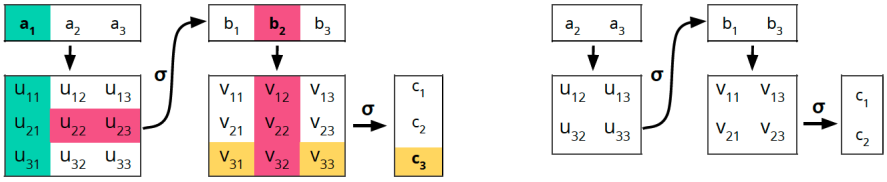


Figure 3.2 Federated dropout for reducing size of weight matrices [34].

.

The clients have to train on the resulting sub-model and, in fact, can be completely unaware of the full central model. From their perspective, each round can be seen as computing updates for completely different models, where the updates will be mapped back to the original model only by the coordinator. To further reduce the size of the data to be communicated, a compression method might be applied on the resulting vectorized parameter matrices.

It might be also worth to mention an interesting direction of researches which aim at reducing the size of NN models through reducing the size of the model parameters themselves through some kind of "quantization". This essentially means to design architectures, training and inference methods for integer arithmetic [110] or even to ternary ($\forall i\ \mathbf{w}_i \in \{-1,0,+1\}$) [140] or binary ($\forall i\ \mathbf{w}_i \in \{-1,+1\}$) networks as in [48] or [171].

3.2.3 Federated Edge Learning

Network congestion and latency issues in every aspects of distributed computing lead to rethinking the cloud based system architectures. Concepts of *Mobile Edge Computing* and *Fog Computing* are all about bringing the computational resources in the proximity of users' end devices, deploying application servers at edge servers or even at more powerful nodes. Following this advance in distributed computing, a range of works investigates the possible interaction of these new system architectures with FL.

FL with client selection (FedCS) [167] provides an approach to adapt FedAvg to Mobile Edge Computing (MEC) environment. For minimizing the convergence time of

the training, modified client selection is applied. In contrast to simple random selection, as introduced in [157], that follows from the pure FL algorithms, FedCS is based on evaluating to which nodes is the network able to provision the necessary bandwidth to download the aggregated model, to upload the updates and, on the other hand, on which nodes have the necessary capacities to execute the training. For this, first, they insert another round of communication to query the available resources of the nodes, based on which the PS will schedule the expected upload timing of the updates, as it is illustrated in Figure 3.3.

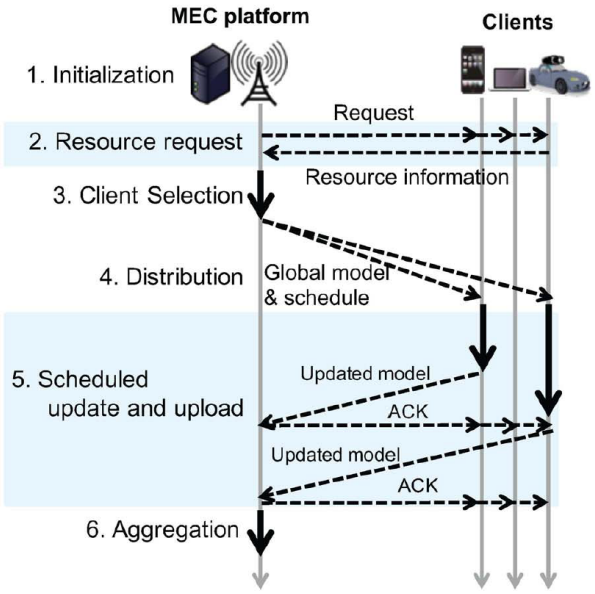


Figure 3.3 Training round in FedCS from [167].

Since the nodes are competing for the communication resources, the selection leads to a constrained optimization task in which a complex problem is approached by heuristics to maximize the total number of participating nodes $|\mathbb{S}|$, given a constrained time and number of training rounds for the whole training process. If T_{round} denotes the maximum time for a training round then the task is

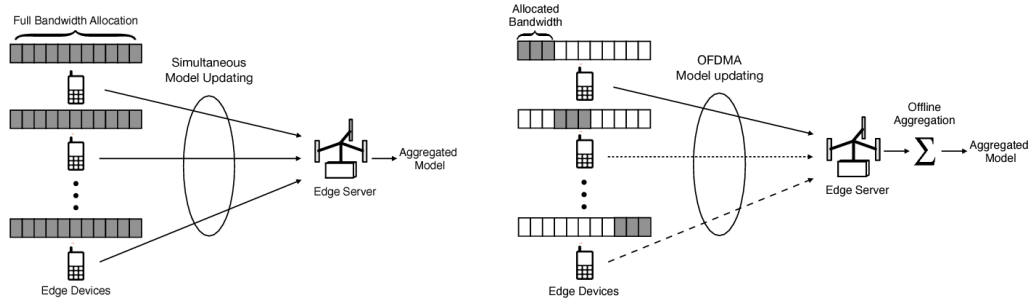
$$\max_{\mathbb{S}} |\mathbb{S}| \tag{3.3}$$

$$\text{s.t. } T_{\text{round}} \geq T_{\text{cs}} + T_d^{\mathbb{S}} + \Theta_{\mathbb{S}} + T_{\text{agg}}, \tag{3.4}$$

where T_{cs} stands for the time for client selection, $T_d^{\mathbb{S}}$ is the time for model distribution for the selected nodes, $\Theta_{\mathbb{S}}$ relates to the time for training and upload the update for all the selected nodes and T_{agg} refers to the time of model aggregation. Since the order of nodes in $\mathbb{S}$ influences the $\Theta_{\mathbb{S}}$, this becomes a complex combinatorial problem for which they propose to greedily add nodes to the set with the smallest expected training and updating time. Similarly to [25], if despite of the initial query round some nodes cannot fulfill the planned time constraint at any stage, the communicated data will be simply dropped.

A further step into realistic FL is to apply *over-the-air computations (AirComp)* [163, 80]. AirComp methods seek to take into consideration the impact of "hostility" of wireless *multi access channels* (MAC) on significant bandwidth need, as well as the possibility of exploiting their special, sophisticated properties (e.g. fading, multi-access and broadcasting, and spatial multiplexing) to increase communication efficiency.

For exploiting the possibilities of AirComp, Zhou et al. [254, 253] suggest to use *broadband analog aggregation* (BAA), that builds on the idea that synchronized updates can be aggregated over broadband channel by exploiting waveform-superposition property of the multi-access channels.



(a) Broadband analog aggregation by AirComp

(b) Broadband digital aggregation by OFDMA

Figure 3.4 Broadband analog aggregation versus broadband digital aggregation from[253].

The principle of the analog computation is, that instead of assigning sub-channels to the distinct wireless devices to transmit the parameter vectors independently from each other through *Orthogonal Frequency Division Multiple Access* (OFDMA, a multi-user version of the OFDM digital-modulation technology), the sub-channels are assigned to parts of the set of parameters to be transmitted. Thus, each device adds its respective update value to

the shared, per-parameter-set channels, where the signals will be added by superposition, as it is depicted in Figure 3.4. To support these over-the-air partial aggregations, a modified beam forming method is required at the same time which, instead of focusing on enabling recovery of individual data streams of single users, aims at amplitude alignment across the targeted devices by keeping the ratio of signals (Figure 3.5).

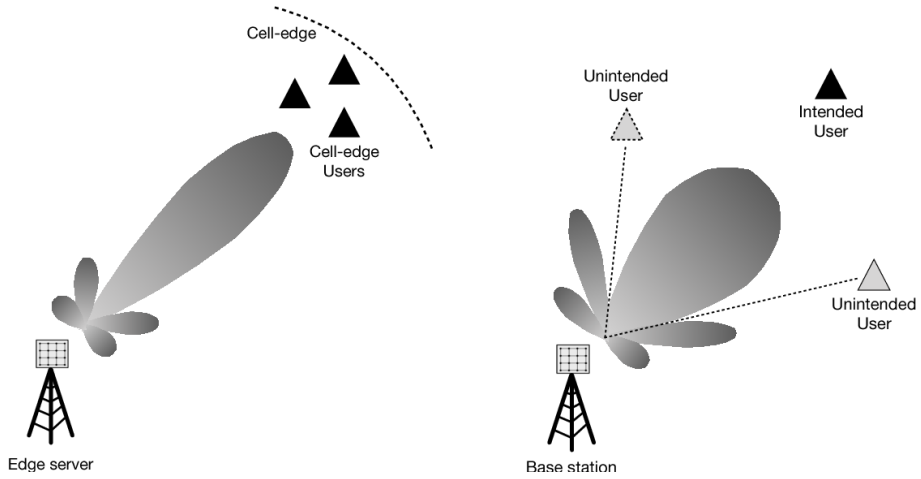


Figure 3.5 Aggregation intended beamforming vs space division multiple access (SDMA) beamforming from[253].

Utilizing over-the-air computation, however, requires a trade off between acceleration of training (by involving more devices) and the increased aggregation error rate (due to the unreliability of the channel). To overcome this problem, a joint client selection and beam-forming approach is proposed in [235] to find the maximum number of devices, with which a mean squared error (between the aggregation of real updates and the BAA approximation) requirement is still fulfilled. Since this joint optimization is intractable due to the combinatorical objective function $|\mathbb{S}|$ (number of participant devices) and the non-convex MSE constraint (the error due to the noise of over-the-air computation), the authors propose to model it as a sparse and low-rank problem through a difference-of-convex-functions (DC) representation.

The paper of Zheng et al.[242] also adds the issue of heterogeneous computation capacities to the above discussed channel state information([235]). The proposed bandwidth allocation and scheduling algorithm aims at maximizing the accuracy of the trained model and, at the same time, at minimizing energy consumption, training and update transmission

times at the devices using the OFDMA communication model, that is the communication method utilized in Wifi, 4G or 5G networks.

Amiri et al. [7, 8] compare Digital Distributed Stochastic Gradient Descent (D-DSGD), where the communication and computation are treated separately, and Analog Distributed Stochastic Gradient Descent (A-DSGD), where no channel encoding is applied during the transmission, but the channel superposition property for the sparsified node updates is utilized (quantization with error accumulation, see Section 3.2.2). Taking into account the limited bandwidth and power consumption of submitting the updates, they report faster convergence of the analog approach in setups with low energy and bandwidth scenarios.

3.3 Performance loss in FL

The second important branch of research according to our grouping is about giving up the algorithmic base of FL [129, 157] and making attempts to find new, different solutions for the problem statement of FL. The main motivations for this branch might be the potentially insufficient performance of models at the end users, the performance drop compared to single node training or the vulnerabilities of the centralized algorithm.

Performance degradation The degradation of performance experienced in FL setup is analysed in [251] and [240]. According to the empirical results of [251] the accuracy loss reaches up to 50% even for relatively simple setups but, what is worse, it often fails completely. They assume that the cause of this performance drop resides in weight divergence. That is, the aggregated gradients lead the parameter search to a space where local distributions are more imbalanced.

To reduce the harmful effect of weight divergence a number of alternative “knowledge ensembling” methods has been proposed, based on

- data sharing;
- knowledge ensembles and distillation;
- neuron matching;

- personalization by multitask learning and task clustering.

3.3.1 Data Sharing

For reducing the divergence of the update directions of the worker nodes, Zhao et al. [251] propose to use a small amount of shared "anchor data" that should be uniformly distributed and obtained by the initiator of the training process. Some fraction of this shared data set will be then sent to the nodes to regularize the training. This effect can be further boosted by pre-training the initially distributed model on the collected data set. An accuracy improvement of 30% for the case where each node has only a single class as training data and with 5% shared data on Cifar-10.

3.3.2 Knowledge ensembles and Distillation

A simple, safe enough and widely used method to produce high performance predictors is to train multiple high capacity models on the training data and create an ensemble [56], that will make predictions combining output vectors by weighted average or voting methods. This, on one hand, is a simple way of collecting the knowledge of multiple models but, on the other hand, it might be cumbersome to make predictions based on a big number of complicated models. To solve this problem a simple and elegant method to assemble the models of the ensemble, called *Distillation*, has been presented in [32] that essentially trains a single, high performance predictor to predict the ensemble's (or a complicated resource intensive model's) output vector given the input features.

Distillation is a very well usable technique for combining knowledge of different models, thus they might offer a valuable help in some distributed, and potentially in federated setups as well, especially if they are used in a symmetric fashion, as it is done in *Mutual learning* [248] and *Co-distillation* [9]. (For more details: A.10).

Federated distillation [116] is an interesting, communication efficient approach, even if at some points it violates the philosophy of FL. In their approach, that is based on *co-distillation* [9] the size of exchanged information depends on the output size instead of the number of model parameters. At first place, the goal is to reduce the size of updates

to be communicated in each round, through transmitting only the distillation term, that is the *logit* or soft probabilities. However, in co-distillation this distillation term is example based, that is, for each training example the logit function should be communicated, since a method was invented for use-cases in which the examples are available for all nodes. Thus to use co-distillation in an FL setup one have to address the following problems:

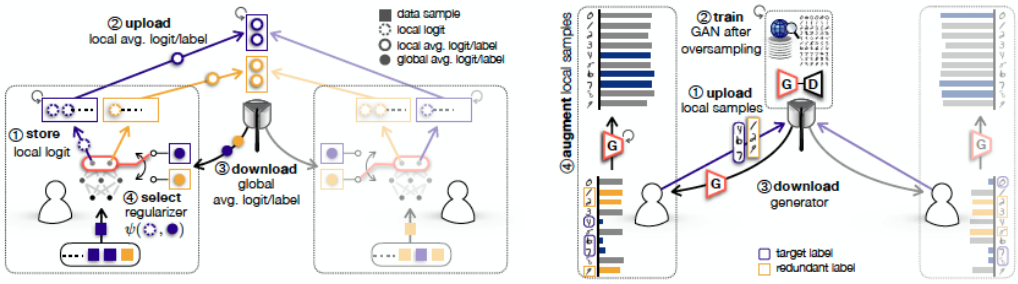
1. the nodes do not have the same training examples, (correspondingly to the setup of FL);

2. even if they would have it, per training example updates are way less efficient than aggregated gradients.

The solution to these problems was offered in [116] in the following way:

1. the parameter-servers will be used for aggregating and distributing a per-label mean logit vector (*Federated distillation*)

2. to resolve the non-iid nature of the local data-sets, in a centralized location a *generative adversarial network*(GAN) should be trained, which will be downloaded by the workers to form iid datasets. (*federated augmentation*)



(a) FD with 2 devices and 2 labels. (b) FAug with 3 target and 3 redundant MNIST labels.

Figure 3.6 Illustration for Federated Distillation and Federated augmantation in [116].

Thus the algorithm (Figure 3.6) periodically updates per-label logit means of the nodes at the PS, instead of the parameter updates, and the PS aggregates and redistribute these. The local models, therefore, will be different from each other. The goal of the training is

to produce similar output for similar inputs across all the nodes. On the other hand, the workers share the GAN, that is trained centrally at the parameter server to enable them to have an iid distribution. The problem at this point is that there is still no real data for GAN training [85]. For this, the authors propose to acquire the related class instances from the internet, which can be considered as a kind of weak point of the method, along with the need for uploading potentially sensitive data.

3.3.3 Neuron Matching

Similarly to federated distillation, the approach of neuron matching for FL, presented in [240], gives up the goal of sharing a common prediction model. Going a little bit further from distillation and, instead of urging the `logits` to agree on given inputs, they *match groups of neurons* through attempting to infer semantics of parts of the trained NNs. The motivation of the method is the fact, that the ordering of neurons in a fully connected hidden layer is permutation invariant. These hidden layers are working as feature extractors, on whose output softmax regression provides the final classification. In their method, each node develops a model on its own, without a need of common initialization or even using common training method. Neuron matching attempts (using a Beta Bernoulli process [210]) to find parts of the NNs, that correspond to detectors of the same or very similar features. The common model is then created by aggregation and assembly of these shards.

Chen et al. [44] presents a communication efficient asynchronous version of FedAvg that builds on the semantics of NNs as well. The key idea here is that, since in deep NN architectures the first few layer behave as general feature extractors, one should pay more attention on the convergence of these. The first idea is, therefore, to invent two types of updates: the first type, that is executed at each round, only transmits the first layers and, the other type that transfers the whole data is utilized less frequently. (The other key idea, already presented in Section 3.2.1 is an asynchronization method, according to which the updates risen from old models will contribute to the aggregated model as well but with a smaller weight.)

3.3.4 Multitask Learning and Personalization

Multitask Learning (MTL) and personalization approaches follow from the idea that even if we assume that the federally trained models perform as well as the centrally trained ones, the resulting models will be too generic for the actual end-users.

As has been summarized in [186], FL assumes a single centralized model to be capable to provide good performance predictors to all the users. The implicit assumption made by FL is that for all local data distribution $D^{(k)}$ at node k, for which the true risk is

$$R^{(k)}(\mathbf{w}) = \int f^{(k)}(x) dD^{(k)}, \tag{3.5}$$

there exist a parametrization $\mathbf{w}^*$ such that, at least locally(in an ε-neighbourhood B_ε),

$$\exists \mathbf{w}^* : \; R^{(k)}(\mathbf{w}^*) \leq R^{(k)}(\mathbf{w}) \; \forall k \text{ nodes, and } \forall \mathbf{w} \in B_\varepsilon(\mathbf{w}^*) \text{ parametrizations.} \tag{3.6}$$

This assumption, however, might be violated in the following ways [186]:

1. Clients disagree on conditional distributions, i.e. $D^{(i)}(y|x) \neq D^{(j)}(y|x)$. (Since they can hold arbitrary data, that cannot be audited by the central server, this scenario is more than probable.)

2. The models might not be expressive enough to fit all distribution the same time. (Since computations run many times on small user equipments, they most often might be unable to train complex models that can capture the subtle differences.)

The authors provide a set of intuitive examples to demonstrate the traps of this assumptions:

- **Varying preferences** If one would like to train a classifier on images of people to predict attractiveness, one group of users will tend to say that a people with glasses are very attractive, while others will not be so much impressed. Thus it is easy to see that a single model using merely an image as an input will not be able to reach high accuracy on such problem.

- **Limited model complexity** When one tries to predict the next word in texting, she or he will face with high probability with the fact that different demographic groups, like teenagers or pensioners, tend to use a very different language, exhibiting strongly divergent statistics. On one hand a model not expressive enough is unlikely to be able to capture the characteristics of all the groups, a more complex one, on the other hand, would involve prohibitively large training and prediction resource requirements.

- **Adversaries** When a significant group of users behave in adversarial manner from the perspective of the model, they are biasing the global data distribution in an undesirable direction, that can make the entire training useless. Let us just refer to misfortune of *Tay*, the chat-bot of Microsoft, who started to exhibit not entirely socially acceptable behaviour following the examples of its human teachers.

Thus MTL methods and model personalization techniques are designed to address this issue by building a number of specialized models instead of delivering a single one.

For prediction of next word in texting ([93]), Wang et al. [222] provides a detailed analysis, according to which the globally obtained model should be fine tuned to better reflect the needs of a given user. Their *Federated Personalization Evaluation* (FPE) framework evaluates the effect of personalization by retraining the shared model using the cached local data. The parameters of retraining are defined in *personalization policy* containing hyper-parameters as batch size, learning rate or number of epochs. They evaluated their models before and after retraining, using the accuracy of prediction measuring how often they hit the next word. They found that retraining yields some benefit for the majority of the users, however, there is always a subset of users who experience performance degradation in the same time.

A similar argument has been made in the paper presenting *Agnostic Federated Learning* [161]. According to that, from the perspective of the end-users, the assumed uniform target distribution, that arise from simple summation over distributions $D^{(k)}$ of the local datasets

$\mathcal{D}^{(k)}$:

$$D = \sum_{k=1}^{K} \frac{n^{(k)}}{n} D^{(k)} \tag{3.7}$$

might be unsatisfactory. The overall loss might be impressive, but on a large fraction of small datasets the model might perform very poorly. A simple solution they offer for these problems, therefore, instead of this simple uniform creation of the target distribution (in Equation 3.7), one should use an λ-*mixture model* of local distributions (with a naturally unknown λ):

$$D_\lambda = \sum_{k=1}^{K} \lambda_k D^{(k)} \tag{3.8}$$

. Using this assumption, we can define an *agnostic loss* as

$$f_{D_\Lambda}(\mathbf{w}) = \max_{\lambda \in \lambda} f_{\mathcal{D}_\lambda}(\mathbf{w}) \tag{3.9}$$

To minimize this loss, for a model family M, the task is to find $\mathbf{w}_{D_\Lambda^*} = \arg\min_{\mathbf{w}} f_{\mathcal{D}_\Lambda}(\mathbf{w})$.

Clustered Federated Learning (CFL), introduced in [186], provides an elegant way to overcome the issue of insufficient performance of specific subsets of the clients by progressively clustering the nodes based on the directions, in which they are attempting to push the common model (e.g based on distance of gradients), until stationary state will be reached at every cluster. The method is about alternation of traditional FL training and splitting the node set into clusters of similar nodes. Once an FL training has converged on $\mathbf{w}^*$, that is

$$0 \leq || \sum_{i \in |c|} \frac{|\mathcal{D}^{(i)}|}{|\mathcal{D}_c|} \nabla f^{(i)}(\mathbf{w}^*) || < \varepsilon_1 \tag{3.10}$$

for a cluster c and $D_c = \bigcup_{i \in c} D^{(i)}$, but

$$\max ||\nabla f_i(\mathbf{w}^*)|| > \varepsilon_2, \tag{3.11}$$

that means that nodes are *incongruent*, then we have to split the cluster C. In this case, we take the pairwise cosine similarity between all the gradients and split C minimizing the maximum similarity between nodes from different clusters.

Multi-task learning (MTL) is a concept to learn models for multiple related tasks simultaneously, hoping that, in some sense, the tasks are similar to each other. The general goal of MTL for problems with *convex loss* can be formally described as

$$\min_{\mathbf{W},\Omega} \left\{ \sum_{k=1}^{K} \sum_{i=1}^{n_k} f((\mathbf{w}^k)^T \varphi(\mathbf{x}_i^k), y_i^k) + \mathcal{R}(\mathbf{W}, \Omega) \right\},$$

with $\varphi(\cdot)$ being a (linear) feature mapping and $\mathbf{W} = [\mathbf{w^1}, \ldots, \mathbf{w^K}] \in \mathbb{R}^{d \times K}$ the parameter matrix, whose kth column corresponds to the parameters of the kth model.

The term $\mathcal{R}$ is a kind of a special regularization parameter, that has been originally proposed in [111]. Above the restrictions on the parameters, it also includes the matrix $\Omega \in \mathbb{R}^{K \times K}$ that describes the *task-relatedness*. For instance, as in [249] or [147], this might be the inverse of the column covariance matrix of $\mathbf{W}$. The general regularization term for the MTL can then be given by

$$\mathcal{R}(\mathbf{W}, \Omega) = \lambda_1 tr(\mathbf{W}\Omega\mathbf{W^T}) + \lambda_2 ||\mathbf{W}||_F^2. \tag{3.12}$$

Using this regularization, $\mathcal{R}$ encourages models $\mathbf{w}^k$ to be similar for related tasks, keeping a priory assumptions, like being as simple as possible. This model can be integrated with the CoCoA framework to build communication efficient distributed multi-task learning methods, as for instance in [249] and [147], which soon have been adapted for the federated learning settings for convex problems as well [198].

In *federated multitask learning* [198], during the training, the algorithm alternates between optimizing $\mathbf{W}$ and Ω, i.e. the method parallely learns the solution for the local sub-problem and the relation between data sets across the nodes. $\mathbf{W}$-updates are carried out in parallel at the nodes, each of those solving the local sub-problem (where conjugate

of regularization term, $\mathcal{R}^*$ incorporates a fixed Ω):

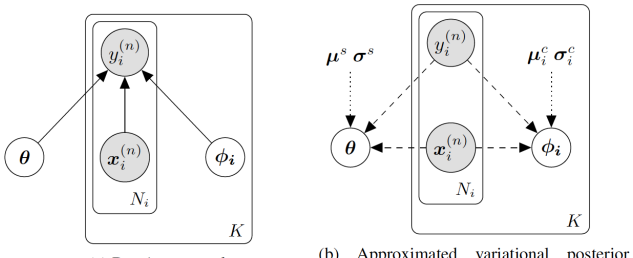
$$\min_{\alpha}\left\{\mathcal{D}(\alpha)=\sum_{k=1}^{K}\sum_{i=1}^{n_k} f_k^*(-\alpha_i^k)+\mathcal{R}^*(\varphi(\mathbf{X})\alpha)\right\}. \tag{3.13}$$

After the model updates, in the Ω-step, the similarity matrix will be updated based on the latest $\mathbf{w}(\alpha)$. Since this step is data-independent it can be executed centrally.

Variational Federated Multi-Task Learning (VIRTUAL) [47] has been developed to address federated MTL for generic, non-convex models using a star shaped Bayesian network (Figure 3.7) with the server parameters θ in the center and local parameters ϕ_k at the leaves. The idea is that, similarly to *progressive networks* [184], the local model reuses knowledge aggregated by the server through "gating" activation from the parallely running server model

$$\mathbf{a}_k^{(l+1)}=\sigma(\mathbf{U}_\mathbf{k}^\mathbf{l}\mathbf{a}_k^{(l)}+\alpha\mathbf{V}_\mathbf{k}^\mathbf{l}(\sigma(\mathbf{a}_s^{(l+1)}))), \tag{3.14}$$

where $\mathbf{U}_k^l$ and $\mathbf{V}_k^l$ are the weight matrices belonging to the client and server activation $\mathbf{a}_\mathbf{k}^{(\mathbf{l})}$ and $\mathbf{a}_\mathbf{s}^{(\mathbf{l})}$, respectively, and α is a gating weight. These weights, all together, add up to local parameters ϕ_k at machine k, that is, trained simultaneously with updates to be sent to the coordinator. (More details on training using expectation propagation are given in A.12)



(a) Baysian network

(b) Approximated variational posterior model

Figure 3.7 Variational inference model of [47]. Plates represent replicates over K clients and N_i data-points at node i. **(a)** conditional dependence of y on the shared local model parameters as well as on input $\mathbf{x}$ according to the discriminative model $p(y_i^{(n)}|\mathbf{x}_\mathbf{i}^{(\mathbf{n})},\theta,\phi_\mathbf{i})$. **(b)** the dashed arrows denote the dependencies of the parameter posteriors on data point n of client i (likelihood), (μ^s,σ^s) and (μ_i^c,σ_i^c) represent Gaussian priors on θ and ϕ_i.

3.3.5 Decentralization

A different perspective to deal with the communication difficulties is splitting the model across multiple parameter servers. Implementing both *model* and *data parallelism* is a solution that has been used already in DistBelief [51]. A step further is taken in [4], for the case of limited sized systems, placing the parameter shards and the model shards on the same place at worker nodes, constituting a kind of peer-to-peer system where each node sends and receives updates to and from every other ones (see the Figure 3.8). Increasing the number of nodes, relieving, the global synchronization and giving up

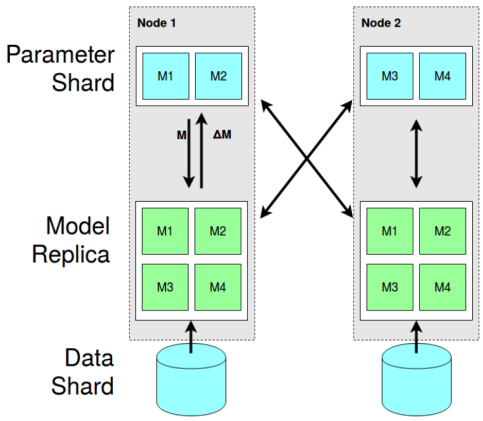


Figure 3.8 Peer-to-peer sharding architecture from [4].

all-to-all communication then leads to the family of gossip based algorithms [58].

Peer-to-peer gossip based methods

These decentralized approaches are usually regarded as the right approach only in the case if the appropriate infrastructure is not available to control the system. Lian et al. [143] however challenge this consensus, conducting a series of experiments about the performance of a fully distributed/gossip based SGD algorithms (Decentralized Parallel Stochastic Gradient Descent (D-PSGD)) in comparison with the centralized one. D-PSGD algorithm, at the kth node, computes the gradient $\nabla f(\mathbf{w}_t^k, \mathcal{B}_t^k)$ for a randomly chosen data point or a mini-batch of points $\mathcal{B}_t^k$. Independently from this, an average $\overline{\mathbf{w}}_t^k$ of the current values of $\mathbf{w}_t^k$ with those of its neighbours will be computed, weighted by matrix that

encodes a kind of "distance" between the nodes. The new value for $\mathbf{w}$ is then given by applying the gradient for the averaged point, with learning rate η:

$$\mathbf{w}_{t+1}^k = \overline{\mathbf{w}}_t^k + \eta \nabla f(\mathbf{w}_t^k, \mathcal{B}_t^k). \tag{3.15}$$

Along with theoretical justification of the competitive convergence rate, they have also empirically proven that D-PSGD is able to achieve the performance of centralized algorithms without the network congestion caused by the latter. These results hold even for non-convex optimization problems (ResNet for CIFAR-10 image classification and proprietary NLP task), at least in case of uniformly distributed local datasets.

In fact, as it is pointed out by Hegedűs et al. [94], the environment that gossip algorithms assume is analogous with the requirements of FL. They assume huge number of workers, data staying at the place of creation and, in a lot of cases, highly varying number of data points.

Based on these observations the authors of [94] propose a completely decentralized gossip-based algorithm. They decentralize the aggregation of models in a way that nodes, upon being done with a local training round, completely asynchronously send their models to some of the neighbouring nodes. For this, destinations can be picked with the help of a peer sampling service on an ad-hoc network of nodes [115]. Those nodes, upon receiving one ore more models, take a stochastic average of their own model with the received ones, and retrain the resulting model on their own data.

Similar gossip models have been introduced in [215] and [18], where the main task is balancing between local performance of the models at nodes, and a requirement for a kind of smoothness over the parameter space. The presented methods are peer-to-peer multi-task learning like algorithms for collaboratively learning personalized models over a network of nodes. The set of nodes in these works are represented as a graph, in which the neighbourhood matrix $N \in \mathbb{R}^{n \times n}$ contains the weights of edges, that describes similarities of nodes or "task relatedness". This can be based on user profiles or the data itself. For normalizing relations, a diagonal matrix D can be used with $D_{ii} = \sum_{j=1}^{n} N_{ij}$. In the algorithm, the primary objective of the nodes is to obtain a model describing their data as

well as possible:

$$\mathbf{w}^k_{\text{sol}} = min_{\mathbf{w}\in\mathbb{R}^n} f^k(\mathbf{w}) = \sum_{j=1}^{n^k} f(\mathbf{w}; x^k_j, y^k_j) \tag{3.16}$$

Above this, the proposed systems seek to achieve a kind of *smoothness* over all the models in the sense that more related task should have more similar models. The idea of the first approach, called *model propagation* [215], is, first, to find the exact optima at the nodes and, then, make the weights smooth over the edges through minimizing the penalties imposed by the difference of models at nodes to each and by too much change of models with many training examples:

$$\mathcal{Q}_{MP} = \frac{1}{2}(\sum_{i<j}^{K} N_{ij}\|\mathbf{w}^i - \mathbf{w}^j\|^2 + \mu \sum_{i=1}^{K} D_{ii}c_i\|\mathbf{w}^i - \mathbf{w}^i_{sol}\|^2), \tag{3.17}$$

where $c_k = n^k/max_j n^j$ is a confidence parameter which is proportional to the data amount at the node and μ is a trade off parameter between the grade of smoothness and the local accuracy.

In the *collaborative learning* algorithm, learning and propagation are interweaved, thus the objective

$$\mathcal{Q}^i_{CL} = \frac{1}{2}(\sum_{i<j}^{n} W_{ij}\|\theta_i - \theta_j\|^2 + \mu \sum_{i=1}^{n} D_{ii}\mathcal{L}_i(\theta_i)), \tag{3.18}$$

where the left side term is responsible for smoothness and the right side therm is to prevent too high loss of local accuracy. This objective is proposed to be minimized by a distributed version of ADMM [29]. However, dual-based methods, as ADMM are working only for convex problems. To apply them for our problem, one must use some "convexification" of NNs [12, 154, 212, 246]. Though it was intended to use for liner models, application of convexification built upon model averaging [94] with appropriate hyper-parametrization would most probably work for NNs as well.

A bit simpler, gradient based (consequently NN-compatible), method for peer-to-peer learning of personalized models has been presented by Bellet et al. [18], where the objective

to minimize (with the notation used in [215]) is

$$\Omega_{\mathrm{CL}}(\Theta) = \frac{1}{2}\sum_{i<j}^{K} N_{ij}\|\mathbf{w}^i - \mathbf{w}^j\|^2 + \mu\sum_{i=1}^{K} D_{ii}c_i f^i(\mathbf{w}^i). \tag{3.19}$$

A more detailed description of these gossip algorithms ican be found in A.11.

3.4 Privacy

From the point of view of cyber-security, in a system like the FL there is a wide range of potential threats we should calculate with. In a comprehensive work about the state of FL research [118] the authors listed, for instance, the following main categories, from the point of view of the "model owner":

- malicious client can inspect or tamper the training process;
- malicious server can inspect or tamper the training process;
- data analysts or compromised clients can steal the trained model.

For addressing these threats a number of widely used technologies can provide us some help. In general, security issues which arise in connection with FL can be classified under the field of *secure multi-party computation* [236]. The problem statement of this area of cryptography is that a set of parties compute a function together on their private inputs, without revealing anything apart from the intended output. The field involves techniques that are built on secret sharing with *homomorphic encryption* [79] or *oblivious protocols* [109]. In the spirit of the original specification of the problem, the most important security aspect of FL is the protection of privacy of the users who contribute to the training with their potentially sensitive data.

One solution to ensure user privacy is using *Trusted Execution environment* (TEE) or *secure enclaves* [204]. TEEs are designed to run critical codes in a way, which ensures that runtime and memory patterns do not reveal information about the input data (for example, *Prochlo* [22]) and, also, guarantees that the right code is running within.

Using TEE as an example to protect user privacy against a malicious server is *Secure shuffling* [39, 135], that also involves trusted entities, that receive data from the clients and forward them to the aggregating server in a way, that it is unable to infer which updates belongs to which client.

Nevertheless the ideal solution would be to guarantee privacy without any need for single trusted entities. That is, the main goal is to protect the user's data from the moment when any information about it leaves the user's device. That means no one, including the server or some trusted entities, should be able to make "very concrete" conclusions about the data, what the user owns. Analogously, when the communication involves secret key protocols, the most appropriate approach would be holding it at the clients, that is, to share among parties as it has been proposed by [176] and [181].

The most serious threats from the perspective of privacy of user data we have found in our research work are the following attack method(described in more details in Appendix A.6) :

- **Membership inference** Exploiting the phenomenon of distinct behaviour of models fed by members of the training data, one can decide if a record was used in the training.
- **Attribute inference** Utilizing *membership inference*, assuming that a given record has been used during the training, the most probable values of sensitive attributes can be inferred.
- **Gradient leakage** Under some more or less restrictive circumstances training data points can be completely reconstructed from model update vectors

The two most important concepts, specifically for protecting users' privacy, in the FL setup to are *differential privacy* and *secure aggregation*.

3.4.1 Differential privacy

To achieve this goal of protecting users' privacy the state-of-the-art model is *differential privacy* [63] that quantifies and limits information disclosure about an individual with

privacy loss parameters (ε, δ). A random algorithm $\mathcal{A}$ is (ε, δ)-*differentially private*, if for all $\mathcal{S} \subseteq \text{Range}(\mathcal{A})$ and for all $\mathcal{D}$ and $\mathcal{D}'$ *adjacent* datasets

$$P(\mathcal{A}(\mathcal{D}) \in \mathcal{S}) \leq e^{\varepsilon} P(\mathcal{A}(\mathcal{D}') \in \mathcal{S}) + \delta \tag{3.20}$$

where $\mathcal{S}$ means all possible outcome of $\mathcal{A}$. If ε, the difference between the probabilities for getting the result from the datasets $\mathcal{D}$ and $\mathcal{D}'$ is small, then it is hard to guess on which has $\mathcal{A}$ been run.

In the context of FL, the adjacency means that one dataset $\mathcal{D}'$ can be obtained from the other $\mathcal{D}$ by removing the data of a single client [158].

Differential privacy can be achieved by applying *differentially private transformation* that, in ideal case, should be done at each client, avoiding the need to rely on any trusted entities. The most common form of these transformation is adding a random noise to the data. With adding the noise, one have to balance between increasing the probability of decrypting the data and making the updates useless.

Privacy preserving deep learning was presented by Shokri and Shmatikov [195]. Their *Distributed Selective Stochastic Gradient Descent* method, that has been designed for collaborative NN training, is similar to FL from the client's perspective. To protect user's confidential data they only submit a predefined number of selected coordinates of the gradient. The selection method is based on the importance of the coordinate of the model. That is, a value in the gradient might be added to the update vector, if its size plus some Laplacian noise is above some threshold. Alternatively, we can choose simply the given number of biggest values. After the selection, coordinates of the sparse update vector are transformed into a bounded range, and a further Laplacian noise is added to the values. This way, the method implements a trade-off between fast and private learning through varying the number of updates allowed to be sent, the threshold and the strength of the noise.

Later, Abadi [3] has presented a general differentially private SGD version for defending against model inversions attacks [67] or membership inference [196], even for the case

when the attacker has access to the model (white-box attack). The (bit simplified) method consist of the following steps:

- Define a clip norm C, and a noise scale σ
- For $t = 0 \dots T$
  1. Compute gradient: $\mathbf{g}_t(x_i) = \nabla_{\mathbf{w}} f_i(\mathbf{w}_t) (= \nabla_{\mathbf{w}} f(\mathbf{w}_t, \mathbf{x}_i, y_i)$ for all $\mathbf{x}_i \in \mathcal{B}_t$, for batch $\mathcal{B}_t$ selected at round t
  2. Norm clipping: $\mathbf{g}_t(\mathbf{x}_i) \leftarrow \mathbf{g}_t(\mathbf{x}_i) / \max\left(1, \frac{\|\mathbf{g}_t(\mathbf{x}_i)\|_2}{C}\right)$
  3. Add noise and sum up: $\tilde{\mathbf{g}}_t \leftarrow \frac{1}{B_t}\left(\sum_i \mathbf{g_t}(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$
  4. Descent: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \tilde{\mathbf{g}}_t$

Here, *norm clipping* bounds the influence of each individual example on $\mathbf{g}$. The *clipping threshold* C ensures that if $\|\mathbf{g}\|_2 \leq C$, then it is preserved, otherwise it gets scaled down to have norm equal to C.

Theorem 1 in [3] states that there exist constants c_1 and c_2 such that, given a sampling probability $q = |\mathcal{B}|/|\mathcal{D}|$, this algorithm is (ε, θ)-differentially private for $\varepsilon < c_1 q^2 T$ and any $\delta < 0$ with

$$\sigma \geq c_2 \frac{q\sqrt{T \log(1/\delta)}}{\varepsilon}. \tag{3.21}$$

Building on this secure SGD method, McMahan et al. [158] presents a deferentially private system for language modelling, that is already working with the specific scenario of FL.

Differential private transformations eventually blur out the contribution of data points in the model. Thus, they reduce the efficiency of membership and attribute inference attacks (or *advantage*, see Appendix A.6.3) by decreasing the difference of model behaviour on seen and not seen data points. Many works attempts to bound the advantage in function of privacy budget ε, but as [105] points out, due to intricate characteristics of training data, these bounds are not too reliable (dependencies between data, distribution of training and attack datasets, etc. For more details see Appendix A.6.3).

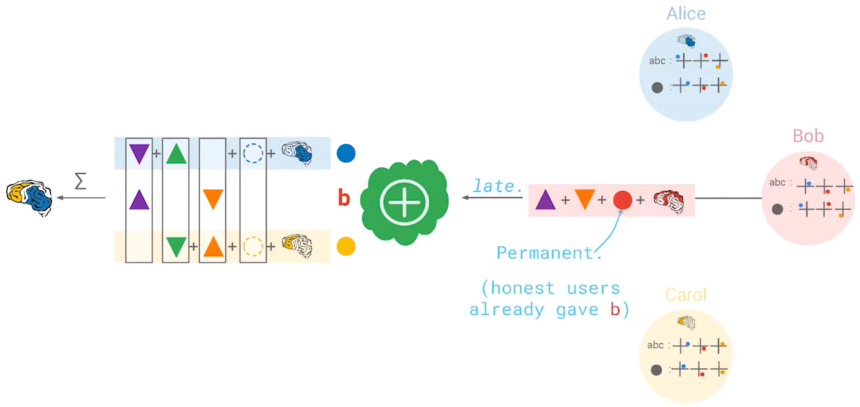


Figure 3.9 Visualization of Secure Aggregation from the presentation of [26] on ACM CCS 2017. When an update of Bob does not arrive, its masks (purple and orange triangle) will be recovered using the shared secret b, while its individual mask (red dot) protects the updates privacy in case of late arrival. If the updates arrive (as the ones of Carol and Alice), the individual masks (blue and yellow circles) will be retrieved and the pairwise masks cancelled.

3.4.2 Secure aggregation

Secure aggregation is the concept for clients to submit their updates in a way that the server will only know the aggregation of those. That can be implemented, among others, with threshold homomorphic encryption [194, 91]. One of the most important results relating to privacy of user data during federated training is the protocol of Segal et al. [26] that builds on secret sharing [191], public-key cryptography, and pseudo-random number generation. The main idea of the algorithm is to add random masks to the updates that change the update vectors completely but cancels out at the aggregation.

These masks are generated by pairwise *Diffie-Hellmann Key Agreement* [57] that uses pseudo-random generators for creating the mask, and whose seeds are given by shared secret keys g^{ab}. Here, g^a and g^b are public keys of the two clients, who possess the secret keys (numbers) a and b, while g is a public generator number. The server broadcasts g^a, g^b, ... public keys and pairs of clients raise the public keys of the others to their secret key a, and b. Thus, $g^{ab} = g^{ba}$ will be their shared secret (this is done for every pairs of clients), the seed used for generating their pairwise mask. For addressing the problem of the dropout

of clients, which would lead to unresolved masks, the next step is to share the private key a and b via *by Shamir's k-out-of-n threshold secret sharing* [191]. This means that if someone possesses at least k shares of a secret, then he can reconstruct it perfectly but, otherwise, it is impossible. This is done by computing a $k-1$ degree polynomial with the y-intercept equal to the secret and along which n points are picked and distributed by the client sharing its secret. From any k of these n points the polynomial can be interpolated, thus, revealing the secret key of the missing client.

If the update arrives late this would lead to violation of the privacy, since all the applied masks are known by now, so the update can be recovered perfectly. To address this, a second individual mask is applied on the updates using another secret key (random seed) at the client that is also published by *k-out-of-n* sharing.

Consequently, if an update arrives then, with respect to a given client, the server collects the shares for the individual masks since the pairwise mask will be cancelled out anyway. Otherwise it asks for the shares of the pairwise mask to cancel the masks out that would remain in the aggregate. For a visualization of the protocol see the Figure 3.9.

By obfuscating the updates, Secure Aggregation is a strong counter measure against gradient leakage. But these methods are more and more powerful and being able to reconstruct larger batches of data, and even draw some information from multiple epoch training as well [76]. Thus the possibility can not be completely excluded, that once they become powerful enough to decrypt the whole update of the common model.

3.4.3 Model robustness

There are some other vulnerabilities of the trained models that might be mentioned. The roots of these problem reside in the tendency of NNs to memorizing specific patterns, that can cause other types of privacy leakage, and models that do not work properly.

One example for the first, privacy related issues, has been presented by Hitaj et al. [101]. The authors call the attention to vulnerability of private information for the cases where a user holds the entirety of a class, as in the case of face recognition systems. The attack they present builds on generative network which is visualized in Figure 3.10.

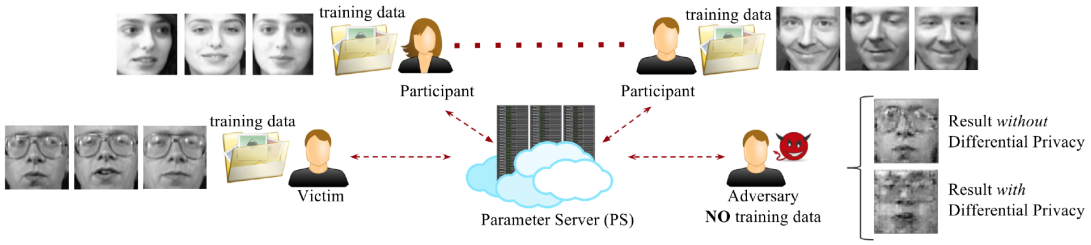


Figure 3.10 GAN based attack for collaboratively trained face recognition [101].

Using the generative model the features of the targeted model can be learned, even if it is not on the level of individual records. For example, if the device contains images of a person it will not return any of the original images but might reconstruct a face. When an attacker pushes these generated images, he forces the target client to release more distinctive features, providing more and more detailed generative examples.

The second of these other issues are not related straight to privacy but to expected operation of the trained models. These problems are stemming from the tendency of NNs to learn very intricate decision boundaries, that can be exploited by adversarial data-points (for example [209, 86, 107, 35, 36]). *Backdoor* attacks[41, 90] build on this phenomenon, applying a kind of *poisoning* of the training data, that results in shifting these boundaries intentionally to misclassify certain inputs. That means that adding some imperceptible features to some of the training data with some intended inference target value, thus, urging the network to make decision based on these (Figure 3.11). Even if achieving this effect in a federated environment is harder due to the quickly vanishing and infrequent impact of a poisoned local data-sets, there are already approaches as *model replacement attacks* [16] (Appendix A.6.1) that might work in these extreme situations as well.

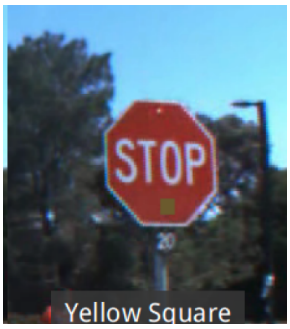


Figure 3.11 A canonical example for backdoor attack in self-driving [90]. The NN can be trained in such a way, that detecting the yellow square on a stop sign will change the predicted class of the stop sign into a speed limit for instance

Chapter 4

Stateful optimization in federated settings

In this Chapter we present our experiments [64, 126] on the effect of applying stateful optimization methods in Federated training. For this first it might be worth to go through the development of FL training again to present why these methods could work.

4.1 Federated gradient based training

In neural network (NN) optimization, due to the non convexity of the loss functions, the most used methods for optimization of network parameters are gradient based, more specifically the versions of SGD [28]. Gradient descent methods take derivatives of loss function according to the parameters of the model, then move the parameter values in the negative of the gradient.

The pure form of SGD samples a random function (e.g a random training data point) $i_t \in 1,2,...,n$ in iteration t and performs the update:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla f_{i_t}(\mathbf{w}_t), \tag{4.1}$$

where η_t denotes the learning rate, which is, in the base case, decaying during the learning to enforce convergence. Intuitively, SGD works because evaluating the gradient at a single training example gives an unbiased estimation of derivative of the error function over all the training examples: $\mathbb{E}[\nabla f_{i_t}(\mathbf{w})] = \nabla f(\mathbf{w})$.

In practice, instead of applying the gradient for $\mathbf{w}$ at each example, usually an average of gradients over b randomly chosen examples is used, that are evaluated at the same $\mathbf{w}$. This method is called minibatch gradient descent (MBGD), that better exploits parallel computational capabilities of the hardware. (MBGD is still commonly referred to as SGD) Though SGD/MBGD in the above form is very popular in optimization, the basic approach can sometimes result in very slow learning. To tackle the challenges incurred by high curvature and noisy gradients of the loss function of NN, a range of method has been proposed based on exponentially decaying average of the gradients or on adapting learning rates. [84]

In this paper, we investigate the effects of these methods on the performance , of federated training of artificial neural networks.

4.1.1 Distributed SGD

When training ML models in a distributed manner, the problem we want to solve is to minimize the loss function f with respect to model parameters $\mathbf{w}$ over all available data points at K nodes (user devices) as follows:

$$\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) = \sum_{k=1}^{K} \frac{n^{(k)}}{n} F^{(k)}(\mathbf{w}), \text{ with } F^{(k)}(\mathbf{w}) = \frac{1}{n^{(k)}} \sum_{i=1}^{n^{(k)}} f^{(i)}(\mathbf{w}), \tag{4.2}$$

where $f^{(i)}(\mathbf{w}) \stackrel{\text{def}}{=} f(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$ denotes the loss on data point $(\mathbf{x}^{(i)}, y^{(i)})$, given $\mathbf{w}$, and $F^{(k)}$ denotes the averaged loss at node k.

To solve the problem in (4.2) the simplest, and in neural network (NN) optimization, due to the non-convex loss functions, the almost exclusively used methods are versions of Stochastic Gradient Descent (SGD) [40]. SGD takes the derivative of loss function at one

data point with respect to the model parameters $\mathbf{w}$, then move the parameter values in the direction of the negative of the gradient:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla f^{(i)}(\mathbf{w}_t), \tag{4.3}$$

where η_t denotes the learning rate. In practice, instead of applying the gradient for each data points, an average of gradients over a batch $\mathcal{B}$ of randomly chosen examples is used (evaluated at the same $\mathbf{w}$). Such "minibatch" gradient descent (MBGD) methods, still commonly referred to as SGD, better exploits parallel computational capabilities of the hardware (like GPU). For both cases, the update is a stochastic approximation of the whole gradient: $\mathbb{E}[\nabla f^{\mathcal{B}}(\mathbf{w})] = \nabla f(\mathbf{w})$. In data parallel centralized distributed synchronous MBGD training, per batch updates are parallelly computed by processors of the DC on their assigned data chunks, then their average will be applied to the central model $\mathbf{w}_t$:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \frac{1}{K} \sum_{k=1}^{K} \nabla f^{\mathcal{B}_k}(\mathbf{w}_t). \tag{4.4}$$

4.1.2 Federated Learning

The main idea behind FL [129] is that, instead of moving the data to a central location and use training method from Eq. (4.3) or (4.4), one could exploit the computational power residing at user devices and partition the training process among them. This, however, brings more complication in the formula. Due to the real distributed nature of the system, communication becomes expensive and unreliable. Therefore Federated SGD (FedSGD) [129] algorithm applies a modified version of (4.4) to reduce communication complications, where instead of communicating the gradients per batch, the central updates takes place after multiple local updates:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{n}{n^k} \sum_{k=1}^{k} \Delta^{(k)}, \text{ with } \Delta^{(k)} = \sum_{i=0}^{r} \nabla f^{\mathcal{B}^k_{t_i}}(w^k_{t_i}), \tag{4.5}$$

where $\mathbf{w}^{\mathbf{k}}_{\mathbf{t_{i+1}}} = \mathbf{w}^{\mathbf{k}}_{\mathbf{t_i}} - \eta \nabla f^{\mathcal{B}^k_{t_i}}(\mathbf{w}^k_{t_i})$, $\mathbf{w}^k_{t_0} = \mathbf{w}_t$ and r is a hyper-parameter for the number of local updates.

To further increase communication efficiency, FedAvg algorithm [157] takes only a small subset (10%) of updates. It has been empirically proven to be able to keep or, in some cases, even increase convergence rate of the learning. FedAvg became the baseline of FL research.

Since the data to be processed by FL generated at a huge number of independent nodes, it has the following characteristics:

1. *Massively Distributed*. The number of nodes can be much bigger than the average number of training examples stored on a given node (n/K).

2. *Non-IID*. The data points available locally are drawn from a different distributions.

3. *Unbalanced*. Different nodes may vary by orders of magnitude in the number of training examples they hold.

4.2 Motivation and related work

When it is compared to traditional NN training methods, FL exposes a significant performance drop, that reaches up to 50% accuracy loss, even for relatively simple setups [251], and too often fails completely.

Problems of weak performance of FL might root in multiple factors residing in the nature of learning. First, computation of global updates might involve problems of **large batch learning** and, second, **non-iid nature** of training data poses additional statistical challenges as well.

Huge batches

We can view the FL update rule in Eq. (4.5) as using huge batches in MBGD: $|\mathcal{B}_{\text{FedAvg}}| \approx |\hat{K}|r|\mathcal{B}|$, where $\hat{K}$ is the number of nodes participating in the training round in FedAvg ($= K$ in FedSGD). Moreover, SGD based methods build on the assumption, that the update

is an unbiased estimate of the full gradient, that is unlikely in a non-iid setting. The local updates $\Delta^{(k)}$ can be considered as an approximation of the gradient, computed over an extremely large mini-batch [156], that with the central aggregation corresponds to an even bigger one.

Larger batch sizes has been typically used to support parallel processing. First, for better utilizing GPUs (Eq. (4.3)), and, second, for data center based model parallel multi-machine processing(Eq. (4.4)). The drawbacks however became soon visible when the models trained in this manner showed a significant decrease in generalization ability[138, 120].

To analyse generalization problem of large batch training, multiple experiments are presented with really large batch sizes in the literature, as 4096 [102] or 8192 [88] inputs per batch. Both of these work aim at preserving the statistical properties of the gradients proving that increasing the learning rate, and using an initial warm-up phase along with batch normalization results in similar convergence rate to small batch training can be achieved. Yet, as [156] concludes, large batch learning reduces the range of usable hyper-parameter setups, which might lead to worse performance, and can even prevent convergence in FL.

Weight divergence

Hidden layers of NNs act as feature extractors and multi-layered architectures are fitted to learn increasingly complex features, layer by layer. Since the key characteristic of FL is the presence of very divergent local data sets, the local models might learn very different higher level features.

In sequential learning with frequent model updates, ordering of the neurons is permutation-indifferent. However, when one computes pairwise statistics on the corresponding neurons, that are fitted to detect very different patterns, as at the averaging of local updates (Eq. 4.5), the resulting model's performance drops. Even if, as in case of FedAvg [157], the training starts from the same state, the more we train locally, divergence has a less and less negligible effect. [251] proposes to reduce this effect through sharing an "anchor" data set that will be distributed across all the participant node, however it is a bit contradictory to the principles of FL.

The other approach, Neuron matching, introduced by [240] could be utilized that attempts to find parts of the NNs that correspond to detectors of the same or very similar features. The common, completely new model is then created by assembly of the corresponding shards, avoiding blind coordinate-wise averaging of parameters.

Uncertain convergence

To carry out thorough analysis of convergence of NN training in the FL involves so many variables that makes giving meaningful guarantees extremely hard. There have been a lot of effort carried out to give theoretical convergence guaranties, however due to the complexity of the problem all of them make serious restrictions.

Many works, that are trying to give convergence guarantees make assumptions even for the case of convex optimization, like *iid data* or *all devices are active*. The latter assumption was made in [121, 239, 224], while [252, 202, 221, 227] assumed both. [141] and [142], on the other hand, provide convergence analysis for true FedAvg with non-iid data, but for the case of strongly convex optimization objective, thus they are not really applicable for the NN case.

4.2.1 GD based methods in FL

Stateful methods are commonly used to enhance learning in single node setting, and have been designed to overcome very similar problems that we face in our setting. Now we make an attempt to empirically measure the impact of some stateful methods on the performance of federated training, that have been used to overcome challenges in different aspects of distributed ML.

In our experiments we tested the effect of simple and Nesterov momentum, from adaptive learning rates (ALR) based techniques Adagrad and RMSProp, and Adam that is the combinations of ARL and momentum.

Application of *momentum* has been already proposed in [88] for reducing variance in large batch learning while [93] and [222] uses Nesterov momentum for federated training of LSTM networks for next word prediction (but also used in parallel SGD for a long time,

as in elastic averaging [244], for example). Using ALR methods (namely Adagrad) is also already proven to have a strong stabilizing effect [51] on the performance in data parallel distributed training.

4.2.2 Momentum techniques

Momentum based techniques [170] use a velocity term during learning, that is an exponentially weighted average over the gradients of the past.

$$\begin{aligned} \mathbf{v} &\leftarrow \beta\mathbf{v} - \nabla_{\mathbf{w}}\left(\frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{w})\right) \\ \mathbf{w} &\leftarrow \mathbf{w} + \mathbf{v} \end{aligned} \tag{4.6}$$

This term, on one hand, accelerates the learning process and, on the other hand, helps to get over noisy gradient and local minima or flat points of surface defined by the error function f.

A variant of momentum algorithm is introduced in [207] and is based on the Nesterov's accelerated gradient method, that differs from the standard momentum of (4.6) in the place of the evaluation of the gradient.

$$\begin{aligned} \mathbf{v} &\leftarrow \beta\mathbf{v} - \nabla_{\mathbf{w}}\left(\frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{w} + \alpha\mathbf{v})\right) \\ \mathbf{w} &\leftarrow \mathbf{w} + \mathbf{v} \end{aligned} \tag{4.7}$$

In Nesterov's momentum, the gradients are evaluated incorporating the velocity. This can be interpreted as adding a correction factor to the standard momentum algorithm [84].

4.2.3 Adaptive Learning Rates

Setting up learning rates is one of the most important factor in the learning process and deeply influences the performance. Thus, finding methods to adapt the learning rate might yield a substantial increase in speed of the learning. The AdaGrad algorithm [62] adjusts

the learning rates individually for each parameter, taking into account the whole history of the parameters, following the assumption, that if the magnitude of the gradients is big than it should be increased:

$$\eta_t = \frac{\eta}{\sqrt{\sum_{\tau=1}^{t-1} g_\tau^2}}, \tag{4.8}$$

where $g = \frac{\partial f}{\partial \mathbf{w}_j}$ for some parameter $\mathbf{w}_j$, and thus η_t will be the learning rate belonging to $\mathbf{w}_j$ a timestep t.

It has been found empirically that aggregating the gradients from the beginning of the optimization can lead to too fast decay in the learning rate, that, in some cases, leads to weak performance.

To remedy this problem RMSProp [97] (the same time proposed by the authors of AdaDelta [241]) replaces this aggregation with a decaying average, in the form:

$$\mathbf{v}_t = \rho \mathbf{v}_{t-1} + (1\rho) g_t^2 \tag{4.9}$$

$$\eta_t = \frac{\eta}{\sqrt{\mathbf{v}_t + \varepsilon}} \tag{4.10}$$

RMSProp has been proven very effective in non-convex optimization problems of NN, thus, it is the most often used technique in practice.

According to the explanation in [84], AdaGrad is designed to converge rapidly when applied to convex functions. In non-convex cases it should pass many structures before arriving at a convex bowl, and, since it accumulates the entire history of the squared gradient, it can shrink prematurely and eventually vanish. In contrast, discarding the old gradients in the RMSProp case enables learning to proceed rapidly after finding the convex bowl, equivalently as if AdaGrad would have been initialized within that convex area.

4.2.4 Adam

The name of the Adam algorithm [122] comes from "adaptive momentum", and can be viewed as the combination of adaptive learning rates and momentums.

$$\mathbf{g}_t \leftarrow \nabla_{\mathbf{w}}(\frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{w}_{t-1})) \tag{4.11}$$

$$\mathbf{m}_t \leftarrow \frac{\beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t}{1-\beta_1^t} \tag{4.12}$$

$$\mathbf{v}_t \leftarrow \frac{\beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\mathbf{g}_t^2}{1-\beta_2^t} \tag{4.13}$$

$$\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - \frac{\eta \mathbf{m}_t}{\sqrt{\mathbf{v}_t} + \varepsilon} (\text{element-wise}) \tag{4.14}$$

In Adam, the weight update is given by applying the RMSProp learning rate (4.13) on the momentum (4.12). (In Equation (4.13) and (4.12) the denominator is applied bias correction on the estimates.) We are not aware of clear theoretical understanding why this is advantageous, however, it seems to work very well in practice and became a de facto default optimization technique for a lot of ML practitioners.

4.3 Stateful optimization at the workers

In our first, rather limited experiment [64], stateful optimization methods were applied for local training at the nodes, proven to outperform the baseline performance for a simple setup of FedAvg.

For analyzing the performance of the optimizer algorithms, we implemented a simulation environment that trains multiple local NN models that would be aggregated into one common model, according to the Algorithm 1.

Compared to Algorithm 1, we have been varying the ClientUpdate(k, w) method, where the local updates have been calculated. (Except for first experiment, since it describes exactly the MBGD method).

The new ClientUpdate(k, w) method is introduced in the Algorithm 2.

Algorithm 2 ClientUpdate

1: **procedure** CLIENTUPDATE(k, w)
2: $\mathcal{B} \leftarrow$ split $\mathcal{P}_k$ to set of batches
3: **for all** $b \in \mathcal{B}$ **do**
4: $\Delta\mathbf{w} = Optimizer(\mathbf{w}, b)$
5: $\mathbf{w} \leftarrow \mathbf{w} - \Delta\mathbf{w}$
6: **end for**
7: return W
8: **end procedure**

Naturally, all optimizers have their own hyper-parameters which should be tuned to get the best possible result. However, for this experiment we used only the recommended values for them (that are in fact the default values in Keras/TensorFlow libraries).

4.3.1 Topology

The model we used is a simple multi-layer perceptron. The input layer consists of 784 input units that is the flatten representation of the input images of size 28×28 pixels. The input is connected to one hidden layer of 128 neurons with ReLU activation. The output layer corresponds to the 10 output classes, thus it has 10 neurons with softmax activation.

In the implementation of the network, we relied on Keras NN API on a TensorFlow backend.

4.3.2 Data

For the experiment, we have chosen the Fashion MNIST dataset [230] that was planned to replace the MNIST benchmark database.

From the characteristics of the FL scenario, in this experiment, we focused on non-iid nature of the data. That is, we have created local datasets of a highly skewed manner. Namely, training data at a given node contains exclusively, or almost exclusively, instances from the same class.

For these experiment, we have not taken into account the unbalanced nature, and each node have been assigned the same amount of data. Our idea here was that if something works in this simple setup then it might work in use cases closer to the real world problem.

Due to the lack of computational resources we also ignored the "massively distributed" condition and set the number of nodes to 10.

The distributions of the local datasets we tried in the experiments are the following:

99% non-iid

The training data has been split into two parts in the ratio of 99%-1%, where the parts are independent and identically distributed, as best as possible. The 99% part will be assorted accorded to classes and then one class assigned to one of the nodes. The 1% part will be equally split into 10 parts and then added to the dataset of the particular nodes.

full-non-iid

In the second test case, we assorted fully the training data and each node receives a dataset consisting of instances belonging purely to one single class.

4.3.3 Hyperparameters

To measure general applicability of the examined algorithms on the problem of FL, we executed the learning process multiple times, using different parametrisations. In setting the hyperparameters we followed the Method of GridSearch, that is we defined a set of possible values for each hyperparameter, then run the algorithms with all the combinations. At defining the set of the possible values we tried to include extremities and generally recommended values. In addition to the parameters described in Section 4.2.1 we also included experimenting with the decay of the learning rate. Here we only tried nevertheless two cases at each configuration of the other parameters, the one without decay, and the one with time based decay, where the learning rate a time t will be $\eta_t = \frac{\eta_0}{1+\phi*t}$ with the decay rate parameter $\phi = \frac{\eta_0}{max\{t\}}$.

4.3.4 Results

FedSGD - Simple Minibatch Gradient Descent

As a baseline we run the experiment with the standard Minibatch Gradient Descent optimization. The experiment result is shown in Figure 4.1. It is clear that, as it often happens, the most simple algorithm, MBGD places the baseline rather high for the more sophisticated optimizer algorithm. It performs very well for the 99% non-iid datasets and surprisingly well with the full-non-iid datasets, achieving an accuracy close to 75% in the 30 iterations with the best configuration of hyperparameters ($\eta = 0.001$, no decay).

Moreover, in results it seems like on both distribution too high learning rate without decay leads to a poor performance.

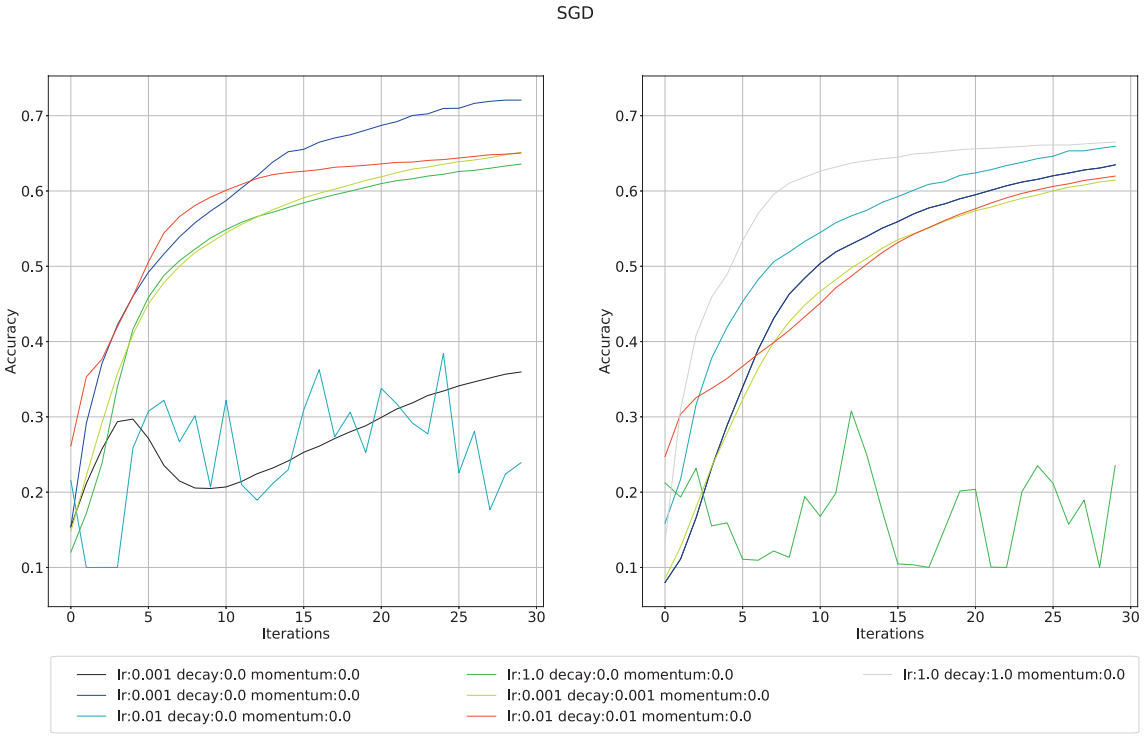


Figure 4.1 FedSGD baseline(simple minibatch updates on 99% non-iid (left) and full-non-iid data (right) distributions)

FedSGD + Nesterov momentum

In Figure 4.2, the results using the Stochastic Gradient Descent with Nesterov momentum can be seen. We found that incorporating local momentum into computing the partial directions of the updates has a strong positive effect both on performance and convergence rate of the aggregated model at both data distributions.

The best performing configurations reached in the first couple of iterations the highest accuracy achieved by the baseline during the entire experiment. According to our results the higher the value β (that is the past directions influence stronger the update) is generally the better performance, apparently independently from η and decay rate.

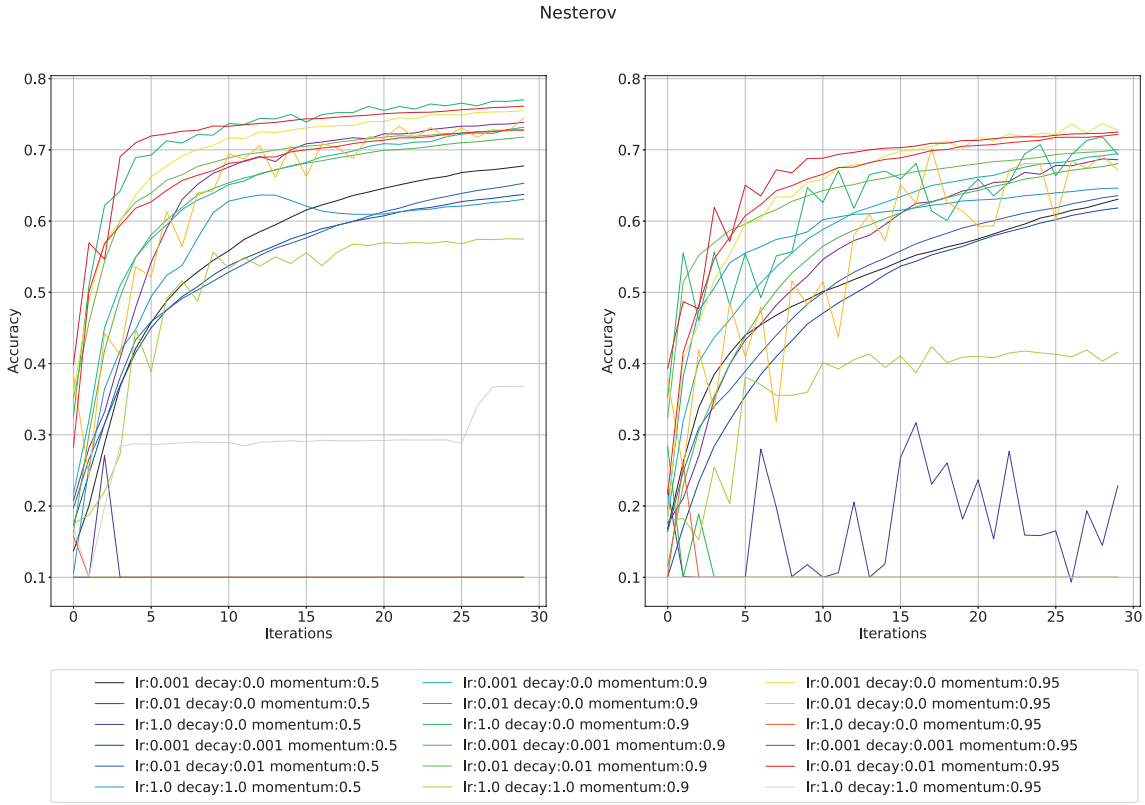


Figure 4.2 FedSGD with Nesterov momentum on 99% non-iid (left) and full-non-iid data (right) distributions

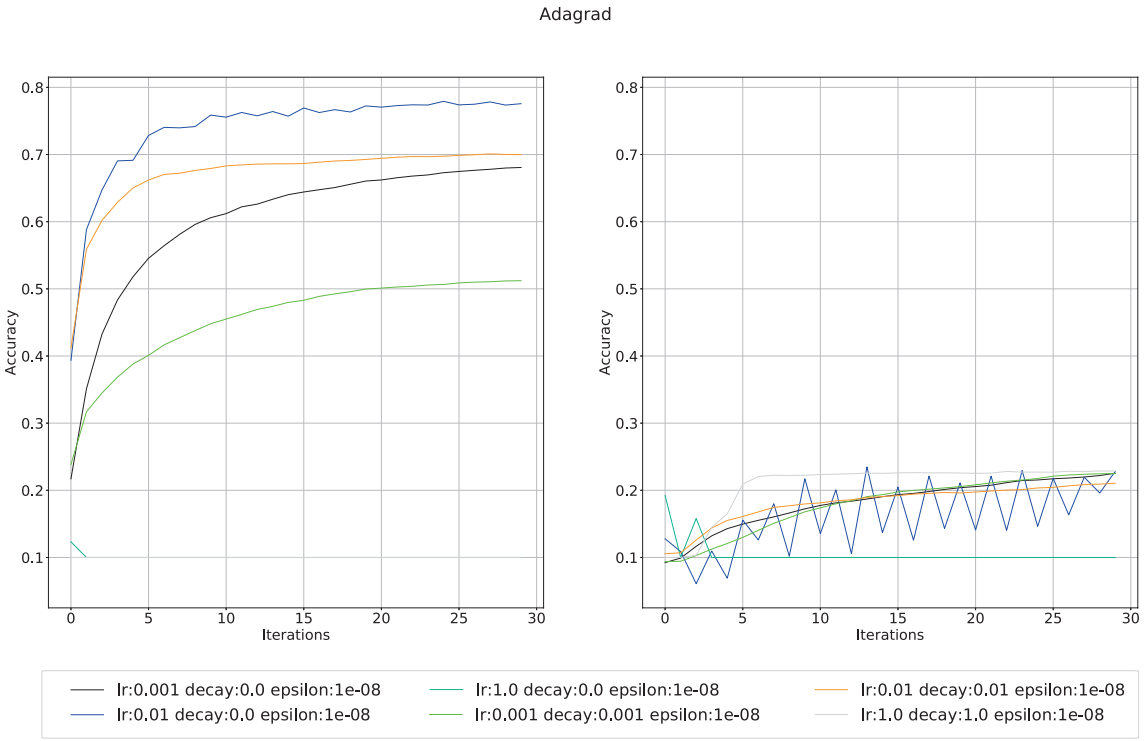


Figure 4.3 FedSGD with AdaGrad on 99% non-iid (left) and full-non-iid data (right) distributions

FedSGD + AdaGrad

The AdaGrad algorithm yields an even better performance 4.3, on the 99% non-iid datasets. Using this method results in the fastest convergence until the 70% of the baseline. In the first 30 rounds, though AdaGrad's performarnce drops dramatically full-non-iid datasets, reaching at most a 25% accuracy without obvious perspective further improvement. It might be interesting to check how many random training examples need to be put into the full-non-iid datasets to achieve the very good performance of the AdaGrad which is measured with the 99% non-iid datasets.

FedSGD + RMSProp

The RMSProp optimizer is used in the experiment which has produced the statistics in Figure 4.4. We experienced that this optimizer algorithm apart from the stronger variance of performance seems to approach the accuracy of the AdaGrad and Nesterov momentum

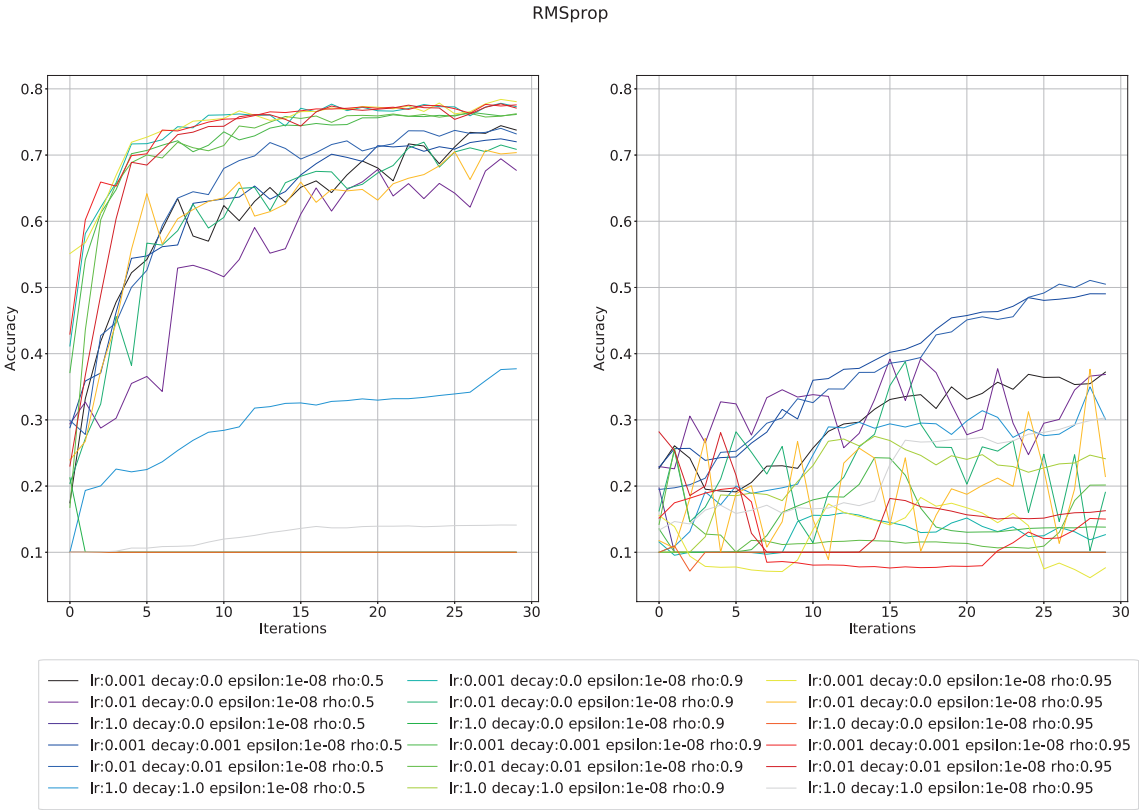


Figure 4.4 FedSGD with RMSProp on 99% non-iid (left) and full-non-iid data (right) distributions

methods, and outperforming MBGD baseline as well on the 99% non-iid distribution. One the other hand though the accuracy on the full-non-iid still achieves a significantly worse performance compared to MBGD and Nesterov momentum, however learning curve is much more promising than the one of AdaGrad. It reaches in the best performing setups about 50% accuracy, and shows an emerging tendency as well.

FedSGD + Adam

The last experiment, we were applying Adam. The method is one of the most popular and often default optimizer(4.12), thanks to fast convergence, high accuracy in the traditional NN learning, and to its robustness to hyperparameter settings. In our experiment however, Adam worked with a very similar effectiveness to RMSProp, and has been definitely outperformed by MBGD and Nesterov momentum (Figure 4.5) regarding to performance and smoothness of learning on the full-non-iid datasets.

4.3.5 Conclusion

In our experiment we found, that the best performing optimizer algorithms for both the distribution are Minibatch Gradient Descent without and with Nesterov momentum, whilst Adadelta and RMSProp is promising despite their poor performance on fully non-IID datasets. As one could have assumed, the presence of the non-iid part of the training data has a very strong regularizing effect even if its weight seems to negligible compared to the dominating class.

In general we experienced that methods that are intended to reduce the variance of the gradient direction works actually quite well for our specific scenario (1). This can be because momentum techniques can be seen as an averaging over the subsequent gradients, leading to a less and less biased estimate of the optimal update direction. The fact that strong momentum (high β) seems to help in the big majority of configurations of the other parameters supports this idea.

On the other hand methods that aim at adapting the magnitude of the gradients seem to harm the learning process (2) in the full-non-iid case. The reason behind this phenomenon

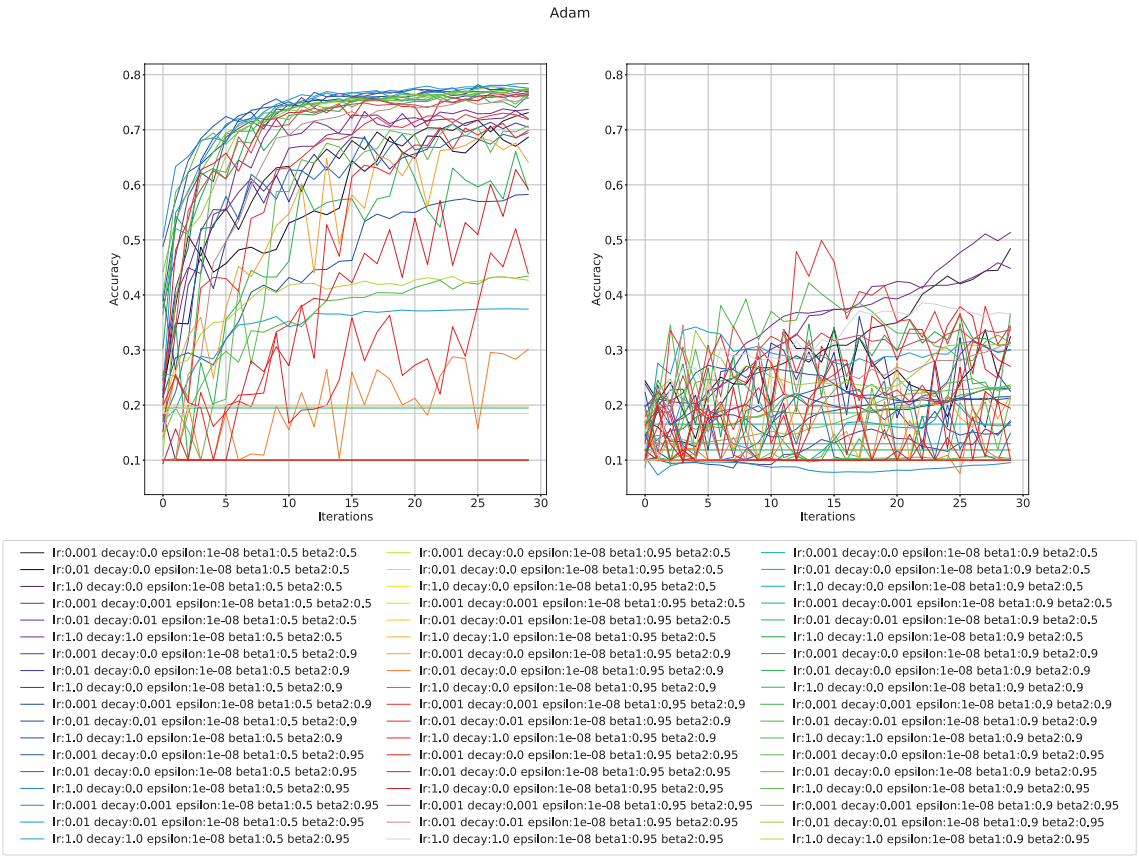


Figure 4.5 Federated Averaging with Adam on 99% non-iid (left) and full-non-iid data (right) distributions

is most probably, that the local optimisers update their inner state – which here corresponds to η learning rate – based on the local gradients. In each local training round according to our intuition the magnitude of gradients start growing, since the aggregation of the local models moved the model away from the locally optimal model, but as we approaching again the local optima, they start shrinking again, resulting in slower start (smaller η) of optimization in the next iteration.

The extremely poor performance of AdaGrad on the full-non-iid dataset can support this intuition, since it prevents even the described fluctuation of the learning rate, instead it decreases it continuously.

The good performance of these algorithms on the 99% non-iid might be explainable with the presence of gradients of really big magnitude in the decaying average that controls the learning rate keeping it at an effective level.

Another interesting phenomenon is that in case of Adam – where momentum and adaptive learning rates are both applied – the strong deccelerating effect of learning rate adaption apparently overrides the help of the momentum. However looking at magnitude of performance differences it might be understandable.

Although according to our results these optimizers are not clearly beneficial in perspective of finding the best global model, they still could be useful for optimizing the global model at clients. One can argue, that in the end the goal of the entire federated optimization is to provide clients with a model performs well on their own data.

4.4 Stateful optimization with central state sharing

Learning method The method we use is a very simple one, similarly to the method described in [148]: Instead of averaging only the weight updates of the nodes, we maintain a similarly aggregated and broadcasted centralized set of variables representing the optimizer state, hoping that, at some extent, it helps to overcome challenges of FL we described in Section 4.2. The worker nodes uses the stateful methods for their local optimization steps, adjusting state variables according to the local data as well. Then, along with the model

weights, the nodes share the optimizer state variables with the parameter server where they will be averaged to get a global optimizer state. When the model of the workers will be updated from the parameter server, so will happen with the state too, and then the new loop begins.

Complexity measure To measure the performance and stability of the learning algorithms in the function of hardness of the task, we define complexity as the reciprocal of best accuracy we achieve across all the optimizers and hyper-parameter setups.

4.4.1 Experimental setup

Data sets Experiments were run with three image classification task such that `MNIST`, `Fashion-MNIST` and `CIFAR-10`. The number of nodes participating in learning, was set to 30. We keep 10% of the data for testing the performance of the centralized model, and then created local data sets in three highly skewed style:

1. "Fifty-fifty": Similarly to the experimental settings in [157], data has been divided into equally sized one or two class chunks;

2. "Full-non-iid": In this setting, all the nodes received data points only from a single class, that is intended to be the most challenging setup;

3. "99%-non-iid": the 99% of data at a node comes from the same class, while 1% is picked iid.

Models For the above mentioned three image classification tasks two types of NNs were utilized:

1. For `MNIST` and `Fashion-MNIST` data sets a fully connected (FC) network with a single hidden layer was used, based on [mni];

2. For the `CIFAR-10` data set we used a convolutional neural network (CNN), following the settings in [cif].

Hyper-parameters We run each algorithm for 80 training rounds with a rather small grid of common and algorithm specific hyper-parameters: learning rate $\eta \in \{0.1, 0.01, 0.001, 0.0001\}$, batch size $|\mathcal{B}| \in \{16, 32, 64\}$, with 1 epoch local training for the FC data sets, and 1,3 and 5 epochs on `Cifar-10`. Values for β_1, β_2 for momentum, ρ for RMSProp are within the set $\{0.5, 0.9, 0.95\}$.For this experiment we did not use learning rate decay.[1]

4.4.2 Result

In all our setups, with a very few exceptions, all stateful methods over-performed SGD (Table 4.1). In Figure 4.6, we plotted the best result of each optimizer on the most complex `Cifar-10` base data set. This shows how profound effect different distributions can have, and how much performance of stateful methods can differ.

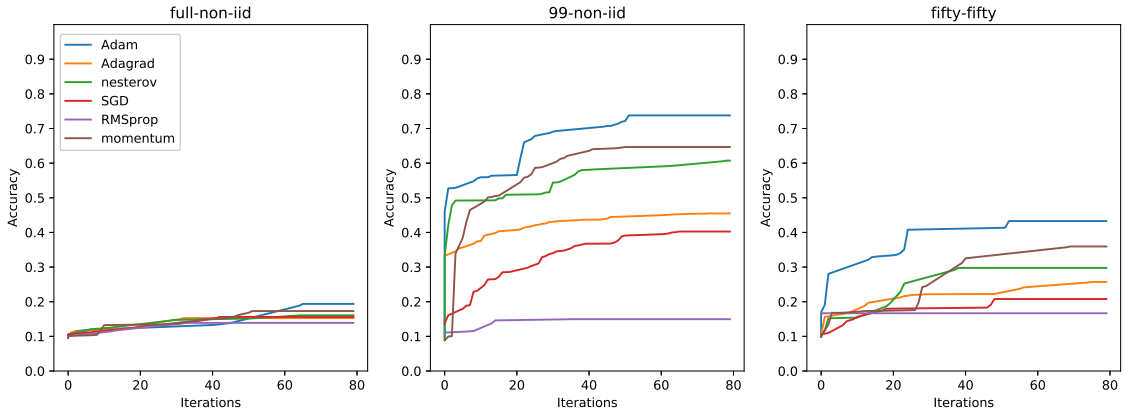


Figure 4.6 Accuracy on `Cifar-10`.

According to the results, ALR methods are performing best on relatively simple tasks (`MNIST` and `Fashion-MNIST`), both in mean performance and best performance. However, their performance drops dramatically on `Cifar-10`, where momentum methods gain advantage. Perhaps because ALR algorithms has been designed to converge fast on convex functions, and the more complex data brings more complex loss surface. Adam shows in each setup one of the bests if not the best results, as it is shown in Figure 4.6 and Table 4.1. Although its performance shows lot of variance and it is very sensitive to

[1]Hyper-parameters are denoted following Keras documentation https://keras.io/api/optimizers/

data set	C	SGD	AdaGrad	RMSProp	Moment.	Nesterov	Adam
Cifar-10 1	2.46	0.20/0.11	0.25/0.12	0.16/0.10	0.35/0.12	0.29/0.12	**0.43/0.13**
Cifar-10 2	6.09	0.15/0.10	0.15/0.10	0.13/0.10	0.17/0.10	0.16/0.10	**0.19/0.10**
Cifar-10 3	1.37	0.40/0.14	0.45/0.21	0.15/0.13	0.64/0.31	0.60/0.21	**0.73/0.33**
FMNIST 1	1.50	0.66/0.43	**0.73**/0.51	0.68/0.50	0.72/**0.53**	0.71/0.52	0.69/0.42
FMNIST 2	1.74	0.58/0.42	**0.69/0.53**	0.58/0.33	0.65/0.48	0.64/0.46	0.64/0.29
FMNIST 3	1.44	0.47/0.36	0.63/0.44	**0.74/0.54**	0.57/0.43	0.59/0.41	**0.74**/0.42
MNIST 1	1.15	0.83/0.51	0.89/0.66	0.86/**0.74**	0.87/0.65	0.88/0.63	**0.90**/0.54
MNIST 2	1.50	0.54/0.34	**0.85/0.57**	0.65/0.35	0.79/0.42	0.77/0.46	0.75/0.32
MNIST 3	1.17	0.67/0.42	0.78/0.56	**0.89/0.64**	0.76/0.47	0.76/0.50	**0.89**/0.48
$-\sigma(C,Acc)$		0.84/0.62	0.81/0.68	0.88/0.77	0.91/0.71	0.63/0.61	0.96/0.92

Table 4.1 Best/mean performance of optimizers on the examined data distributions.

hyper-parametrization (lower mean), in the most difficult tasks it results in the best mean performance along with best maximum performance.

4.4.3 Conclusion

We found that, in general, using stateful optimizers in FL might help to significantly increase learning performance. Naturally, these methods comes with a not negligible communication overhead since the optimizer state usually maintains one or more value per variable. Thus, for momentum the cost of communication is doubled or, for Adam, tripled. However, in some cases, this price can be worth to pay due to the tendency for stagnation of the federated training in a significant subset of hyper-parameter space.

Chapter 5

Evolutionary Federated Learning

This chapter is dedicated to present our experiment [208] on substituting model/update averaging gradient based NN training with completely privacy preserving evolutionary training.

Along with the immense computational requirement of training large scale ML models as NNs, the second problem with traditional data center based solutions, that FL is aiming to solve, is related to *privacy concerns*. It might happen that users of the applications that build on centralized model training are reluctant to share their possibly confidential data. We believe a particularly fitting scenario for this problem is the use case of medical applications. Each medical institute might have a lot of patient data, but that may be far from enough to train their own prediction models. Here, sharing the data across a big number of institute can yield a great help in developing automated diagnostic tools. But being the private nature of these data, hospitals probably decide not to share anything of this information either to protect their reputation or due to legal regulations.

FederatedAveraging works pretty well solving the *aggregation problem*, however using gradients or, equivalently, the local models for the global aggregation step still exposes some information on users data. To address *privacy concerns*, the solution is usually to apply achievements of differential privacy[38][63][3] atop the gradient based learning process.

In this experiment we present a slightly different approach, namely, we investigated whether it is possible to train NNs in a federated fashion without using gradient in any context. To approach the problem, it seemed to be a simple choice to try evolutionary algorithms. Since a rich literature is already available on evolutionary optimization of NNs, we only transfer this knowledge into the federated environment.

For the concrete task to be solved by our method we have chosen classfication of EEG-signals using convolutional neural networks (CNNs).

The main contributions of this paper are

1. a proof of concept for applicability of genetic algorithms to federated training of NNs without using vulnerable gradients;

2. presenting Federated Neuroevolution (FNE) a simple algorithm for the federated training, applying a distributed fitness function.

5.1 Neuroevolution

Evolutionary algorithms (EAs) follow the pattern of evolution as it is observed by biologists in the nature. According to this, in an infinite cycle of life, the most apt individuals can produce offsprings possessing a potentially slightly changed (mutated) mixture of their genoms that might result an enhanced ability to face challenges in their life. The main assumption in biology is that those individuals survive and create descendants with a bigger chance, who, in some aspects are superior to the others. The main structure of an EA is sketched in Algorithm 3

Algorithm 3 EA

1: generate an initial population G_0, $i = 0$
2: **repeat**
3: $\forall individual_j \in G_i : f_j = fitness(individual_j)$
4: select parents from G_i based on their fitness
5: produce offspring generation G_{i+1}
6: $\forall individual_j \in G_{i+1} : individual_j \leftarrow mutate(individual_j)$
7: **until** termination criterion is satisfied

EAs – as nature inspired methods in general – are often used to discover very complex, high dimensional and/or non-convex search problems, therefore, attempts to apply these methods on optimizing NNs has a long history.

Recently, nature-inspired methods in relation with NNs are used mostly for hyper-parameter tuning that includes searching for an efficient architecture.

A big part of this rich literature is concerned specifically CNN-s, what we apply for our problem. Methods of Genetic CNN [232], hierarchical evolution [146], large- scale evolution [55], asynchronous CNN evolution [218] and automatic CNN design [205] give graph based methods to design automatically the stack convolutional layers (skipping potentially fully connected parts of the network) for image classification trough genetical evolution of subsequent layers with various innovative encoding techniques.

In these scenarios, the learning itself is still based on calculating the gradient and updating the model according to that (backpropagation).

Using backpropagation though being based on calculation of gradients and on applying them on the weights of the network is exactly what we want to avoid in our experiment. Before the monocracy of derivative based training algorithms however biology-inspired training methods was a rather popular research topic, thus there is a rich, though a bit dated literature concerned with our constrained problem. [226] and [237] give a summary of the these initial approaches to neuroevolution (NE).

There is a very interesting branch of applications of NE for general NN-s, that includes techniques to purely genetically train the architecture along with the weights of the

networks. Among the most important algorithms that belong here it might be worth to mention NeuroEvolution of Augmenting Topologies (NEAT) [201], and Hypercube-based NEAT (HyperNEAT) [75] and its specializattion for modular evolution of NN-s, HyperNEAT-LEO [217] and Generative NE [216]. Despite of the power of HyperNEAT, we decided first to focus on training a predefined architecture, thus our method is based on more "traditional" NE algorithms.

For applying evolutionary approach on an issue, one need to specify an encoding of the problem, a selection, a crossover and a mutation method as well as a fitness function.

In the rest of this section we shortly describe the stages of an EA along with a couple of examples of how these stages have been implemented in some work on the field of NE, that gave inspiration to our algorithm. At the end of the section we also describe approaches aiming at handle overfitting that has been proven a serious problem in NE.

5.1.1 Encoding

Genetic algorithms work on sequence of features that would be mixed, or altered according some granularity defined over them. Thus the first step in solving a problem genetically is to provide a description of the search space. We refer to this description as encoding, that can be direct or indirect.

Direct encoding is the more traditional way of problem encoding, where sections of the genom more or less correspond to specific parameters. Some of the early methods handle some switches as well, that control the connectivity of the specific perceptrons.

[155] proposes a system based on a parallel genetic algorithm, ANNA ELEONORA, for learning both topology and for connection weights. Topology Utilizes binary representation of networks, with granularity encoding that is handled through one bit flag to determine connectivity, that is, whether the given edge is present in the recent setup or not, followed by the substring of weights. These substrings are ordered in a way that connections into the same neuron are grouped together.

[136] presents a variant of EA applied immediately for float weights. The input of the EA is a vector x of variables , that are the parameters of the model (that is the weights of the connections), the biases, and the newly invented link switches. Link switches are variables, that control the connectivity of the network, that is, negative value represents that the edge is switched off. The search space is constrained by upper and lower bounds on variables (weights): $x \in I_1 \times I_2 \times \cdots \times I_d$, where $I_i = [l_i, u_i], l_i, u_i \in \mathbb{R}$ for $i = 1, 2, \ldots, d$.

Theoretically using the connectivity features of the encoding the first method is able to evolve the architecture too. The issue with this approach to encoding is that the problem space grows very fast as we scale up the network (which we need if we want to solve complex problems).

Indirect Encoding The scaling problem of Direct Encoding can be solved with Indirect Encoding, that instead of separated representation of model parameters uses generative information. In HyperNEAT [75], which is maybe the most important representative of this class, genes of the genom are defining functions based on which weights can be generated.

5.1.2 Fitness

The fitness function serves to specify how well a given individual performs on the problem to be solved. A higher value of the fitness function means a better solution for the problem, while lower fitness value reports a poor performance. Fitness is often normalized thus a function that produces a fitness value 1 for a perfect solution, and 0 for completely wrong setup can work well. As an example for a normalized fitness in ML scenarios, [213] proposes a fitness function for NN defined as $f_{norm} = \frac{1}{1+err}$, where $err = \sum_{k=1}^{m} \frac{\sum_{i=1}^{d} |y_i - \hat{y}_i|}{md}$, with d denoting the output dimension, and m the number of examples, applying mean absolute error.

5.1.3 Crossover

Crossover is a method that defines, how we combine individuals of a generation to create offsprings for the next generations. One simple way is – as in [155] – to combine the

parts of parent individuals at some cutting points. Another approach is presented in [136], where crossover is actually taking the average of the corresponding weights of the two individuals: $x^{(t+1)} = \frac{x_1(t)+x_2(t)}{2}$ where $x(t)$s are the individuals represented as vectors of parameters.

5.1.4 Mutation

Mutation methods serve for adding extra variance to the individual genoms to enable them to discover a bigger part of the search space. [155] provides a representation that translates different topologies and encoding length into a common string format granting compositionally different descriptions. At mutations, it applies three separate probabilities for swapping bits such that granularity bits, connectivity bit, and weight bits. For effectively explore the search space, it uses EA simplex [21], instead of taking three populations and creating a fourth based on those.

In [136], where the possible values for genes are constrained, mutation is carried out according to the following formula: $x^{(t+1)} = x^{(t)} + B\delta$, $B \in \mathbb{R}^2$ is a diagonal matrix, with a diagonal $B_{ii} \in \{0,1\}$, and $l_i \leq x_i^{(t+1)} + \delta_i \leq u_i$. Based on this rule, the algorithm generates three individuals/chromosomes: at the first only one element of the diagonal of B can be one, at the second one a random number of diagonal of elements, and at the last, $B_{ii} = 1$ for all $i = 1, \ldots, d$. The one with the best fitness of these three will replace the weakest one in the next population.

5.1.5 Overfitting

Using EA usually involves a high computation demand, which can be reduced through decreasing the number of evaluations of the model, that is the size of training data on which we want to try out the models defined by a given generation of the genetic algorithms.

Earlier applications of EA usually did not use separation of data into training and test set (like [134], for example). Practitioners soon realized, however, that models trained this way perform poorly on not seen data points, revealing the tendency of evolutionary methods to strongly overfit on the training problems. This issue got in the center of interest,

when as an attempt to reduce run time they tried to use subsets of the training data to evaluate individuals.

[137] made comprehensive experiments proving that evolution is potentially able to extrapolate from the randomly chosen test sets. A very promising direction to reduce overfitting is random sampling, where at each generation, a random subset of the training data is chosen and evolution is performed based on the fitness on that sample. The Random Sampling Technique (RST) [74] was originally used for speeding up the GE runs in [149], however, it was already used for preventing overfitting. [83] and [81] drive some experiments on RST, where they were testing two parameters, the Random Subset Size (RSS) and the Random Subset Reset (frequency of changing the subset). They have found, interestingly, that the techniques performs best when both these values are set to one, that is in each iteration the fitness should be tested using a new randomly chosen data point.

In [82], the authors present versions of "interleaved sampling", that means, instead of random subsets, fitness at each round is evaluated alternating between one and all training samples, with various switching frequencies. As a result, they find that, on their test datasets, the best technique would be to switch in each round between single sample and all sample evaluation.

5.2 The problem

5.2.1 Data

For the experiment, we used the EEG Database Data Set [17]. The dataset contains 120 EEG trial data about 122 patients who either belong to the alcoholic or to the control group. In each trial, the patients were shown one or two images of the Snodgrass and Vanderwart picture set [199]. After showing them the stimuli, their brain activation was measured for 1 second on 64 points at 256 Hertz. The measurements are then labelled according to which group they belong to, thus the task of the model to be built is to predict which class of the two does a sample belong to.

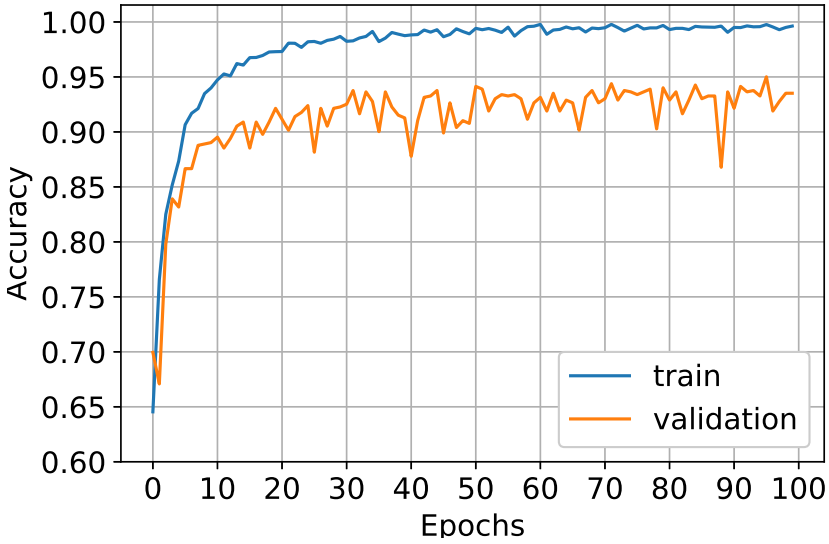


Figure 5.1 Baseline accuracy (backpropagation)

5.2.2 Network architecture

For the network architecture to train, we decided to use the shallow convolutional network from [187], that has been designed specifically for EEG based multiclass prediction problems. The essence of the network is three convolutional layer that are intended to recognize specific patterns in the signals. After two convolutional layers there is a pooling layer and then comes the third convolutional layer. On the output of this layer we applied batch normalization, then added the output dense layer with sigmoid activation.

For the control experiment, we used the AdaDelta optimizer [241] with Categorical Cross-Entropy loss function. At training we used a batch size of 64, 1.0 as learning rate, $\rho = 0.95$, and $\varepsilon = 10^{-7}$.

The control model after 100 epochs achieved a validation accuracy of 95% (see Figure 5.1).

5.3 The proposed methods

The algorithm runs according to the process defined in Algorithm 3. For starting off we create an initial generation in which for each individuals initialize the weights of the

models randomly. From the initial generation then we iterate along the fitness-selection-crossover-mutation loop. In this section we describe the particular methods we used for the different stages.

Selection

The candidate set of individuals for crossover is created by sorting the current generation's models based on their fitness and selecting the $n-1$ fittest models for crossover. The last parent selected for mating is not among the fittest ones, but chosen randomly from the rest, to add more variance.

Crossover functions

Crossover method defines the way according to which new individuals will be generated from the parent generation.

In our method, we pick two parents randomly from the pool of parents to produce the required offspring amount.

We run experiments with four crossover methods. The first three require flattening the vector of weights. These first three approaches are rather popular in EA research.

- **Halving mix:** In this approach, that is a simplified version of the one in [155], the vector of values from the parents are taken to create the offspring vector by taking the first half of it from the first parent and the second half from the second parent. This was the original approach in [69] too. Formally:

$$\{offspring_i\}_{i=1}^{n} = \begin{cases} a_i, & \textbf{if } i \leq n/2 \\ b_i, & \textbf{if } i > n/2 \end{cases} \tag{5.1}$$

  where n is the length of the model vectors and $\mathbf{a} = (a_1, a_2, \ldots, a_n)$, $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ are the parent vectors.

- **Interleave mix:** Here, the vector of values from the parents are taken to create the offspring vector by interleaving the two parent vectors. Formally:

$$\{offspring_i\}_{i=1}^{n} = \begin{cases} a_i, & \textbf{if } i \bmod 2 = 0 \\ b_i, & \textbf{if } i \bmod 2 = 1 \end{cases} \tag{5.2}$$

  where n is the length of the model vectors and $\mathbf{a}, \mathbf{b}$ are the parent vectors.

- **Mean mix:** In this method, similarly to [136], the vector of values from the parents are taken to create the offspring vector by taking the mean of the two parent vectors at each index. Formally:

$$\{offspring_i\}_{i=1}^{n} = \{\frac{a_i + b_i}{2}\} \tag{5.3}$$

  where n is the length of the model vectors and $\mathbf{a}, \mathbf{b}$ are the parent vectors.

- **Kernelwise mix:** In this algorithms the main idea was that we kept intact groups of weights during the crossover operation that "belongs together" in some sense. These were in our case neurons across the convolutional kernels/filters of one individual that are aiming at "examining" the same input (that is the weigths at the same coordinate of the weight matrices), and in the fully connected layers the weights that are connected to a single neuron (that is all the incoming or all the outgoing weights, our methods keeps intact the incoming ones).

In our experiments, the first three crossover methods did not converge. This could be because these approaches are very low level and do not care about the network structure or the patterns learned in the kernels.

Kernelwise mixing is a higher level approach, what we tried after taking a look at how genetics works in nature. In nature, the heredity is also a higher level mixing of genes, instead of low level mix of organic molecules. Thus, traits of the parents are kept intact. The resemblance to genetics can be summarized as follows: the DNA is the network's

weights, a gene is a filter and an organic molecule is a float value. With this latest method, mixing the evolutionary training was converging so we were applying this in our approach.

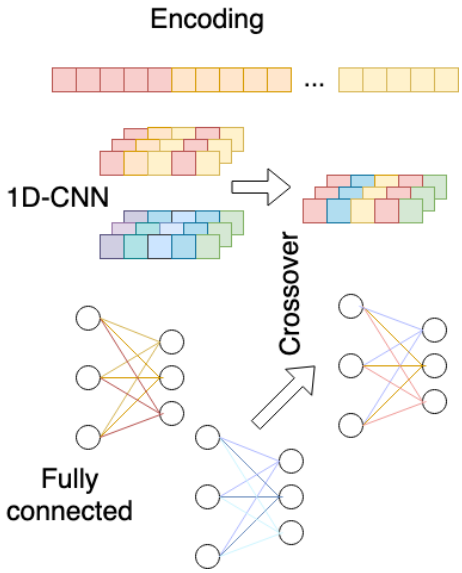


Figure 5.2 The idea of encoding and the "winner method" for creating crossover. The red and blue colors correspond to the different models, while the weights with the same color and tone denoted the group of weights that have been kept together during the crossover.

5.3.1 Mutation functions

Crossover on its own results in generations that are only combinations of the initial generation according to the defined rules. Thus using merely crossover restrict the space searched by the algorithm. To break this random alternations of the offspring are applied in form of mutation functions.

For defining a mutation function we must define the number of mutated values and the scale of the mutation on these values. For the former we used probabilistic value determining the chance of mutation for each value in the model. The latter is a float value determining how much is the impact on each mutating value.

There are the following two main approaches we tried for mutating values in a network:

- **Mutate by offset (from [69]):** Here, we add a random value to the selected values. In our implementation, the offset was a random value between $[-mutation\_rate, mutation\_rate]$.

- **Mutate by multiplication (from [168]):** Here, we multiply the selected values with a random value. In our implementation, the multiplication factor was a random value between $[\frac{100-mutation\_rate}{100}, \frac{100+mutation\_rate}{100}]$.

After experimenting, we found a lot better convergence rate with the second approach.

5.3.2 Federated fitness function

For fitness, which should be maximized during the evolutionary training, we have chosen the Negative Mean Squared Error (NMSE), that is defined as in Equation (5.4).

$$f_{NMSE}(w) = -1 * \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{d} (\hat{y}_j^{(i)} - y_j^{(i)})^2 \tag{5.4}$$

where $\hat{\mathbf{y}}$ is the predicted output vector using parameters $\mathbf{w}$, $\mathbf{y}$ is the target output vector, d is the output dimension and n is the number of examples. This a is slightly different function, than the one in the example in section 5.1.2, but it's behaviour is the same($\frac{\partial}{\partial w_i} f_{NMSE}(\mathbf{w}) * \frac{\partial}{\partial w_i} f_{norm}(\mathbf{w}) > 0, \forall \mathbf{w}, i$)

Applying the NMSE fitness for the original optimization problem in Equation (5.5) our task will be to maximize NMSE with respect to $\mathbf{w}$:

$$max_{\mathbf{w} \in \mathbb{R}^d} f_{\text{NMSE}}(\mathbf{w}) = -\sum_{i=1}^{n} \|\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)}\|_2^2. \tag{5.5}$$

5.3.3 Federated optimization and avoiding overfitting

In our setup the generation of individuals, that is the selection, the crossover and mutation happens at a centralized location at a parameter server. The connected nodes of the system participate in the optimization through evaluating the different proposed setup. The fitness of an individual can be calculated as a weighted average over the local fitness values, in theory, during the training tough, as we will see we should not use this measurement to prevent overfitting.

Avoiding overfitting has been studied in [83, 137, 81, 82], as it is discussed in Section 5.1.5. The main idea is that we must not include the entire training set in the whole duration of the training. Instead, what most articles propose, is to use subsets of the training data in each generation. The training subset can be changed every generation or kept intact for a few generations. Studies interestingly show that randomly selecting a single training sample is also very effective both for convergence and avoiding over-fitting. Another suggested tweak is to include the full dataset every once in a while.

Due the distributed nature of the problem, it was a rather natural idea to incorporate the native data partitioning of the federated setup, and to do the subset selection at a higher level and treat the nodes as units of the subset creation, instead of specific data points. Thus, in each generation, the Federated Neuroevolution algorithm selects a subset of the nodes for evaluating the fitness of the current generation. To ensemble the evaluation sets, we have tried the following three approaches:

1. **Random single element for each generation:** Here, in each iteration we ask a randomly selected node to evaluate the population's fitness on a randomly selected single training sample of it's own.

2. **Random subset for each generation:** In this approach a random subset of nodes are selected to evaluate. We found this method the most efficient.

3. **Moving window subset for each generation:** Here, we first order the nodes and then select a slice of the list of nodes. This is the window and in every n generation we move the window to the right by 1.

The second approach of randomly selecting a subset of nodes had the best performance. Even if, according to the literature, method 1 works pretty well, in our experiments, the training did not converge at all. The third method seemed to be more promising, training convergence was slower with this method than in the case of the second method and the convergence also capped around 75% validation accuracy.

The main algorithm

Using the fitness evaluation methods we described in Section 5.3.3, the main run of the optimization looks like the following:

- **Validation:** On the server, we retain a validation set and in each generation we calculate and store the validation accuracy of the fittest model of the current generation. This is not far fetched as we can assume that in a Federated setting the server driving the learning would already have a dataset of it's own.

- **Avoiding critical points:** Based on the history of validation accuracies, we check the last n entries for a match with the current validation accuracy. If there is a match, we conclude that the evolution has reached some kind of critical point of the fitness function as local maximum or saddle point. That is, however we try to combine and mutate the individuals of the subsequent generations, the fitness/accuracy does not increase. Our hypothesis is, that in this case the population stuck in a higher region of the values of fitness function, and in the neighbourhood defined by our mutation rate the offsprings cannot find any increasing directions. In this case we start gradually increasing the mutation rate and the mutation chance multiplier which is initially set to 1. Once the algorithm is out of the local maximum, we reset the values of the mutation rate and mutation chance to the original values. There is an upper bound on the mutation multiplier.

- **Early stopping:** We save the fittest model of each generation, as an additional means to stop before we overfit.

5.4 Results

We have run the described evolution algorithm for 5000 generations (Figure 5.3). For our setup, we observed that the convergence was slow but steady, overall.

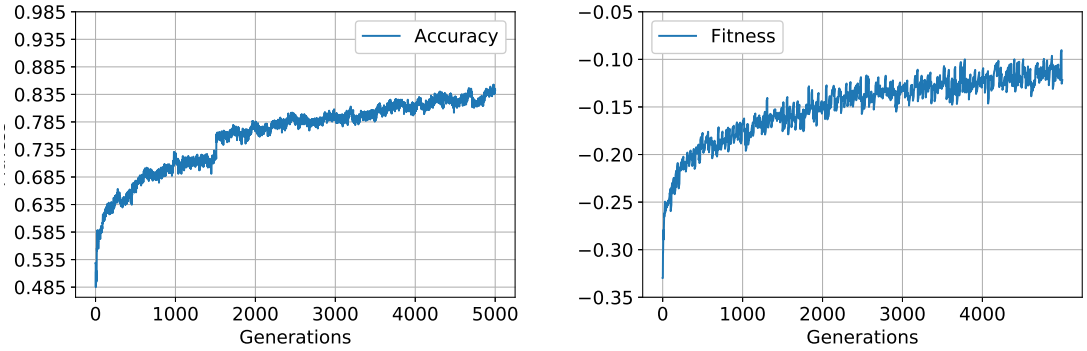


Figure 5.3 Running Federated Neuroevolution on the EEG dataset for 5000 generations. Fitness is NMSE from equation 5.4, Accuracy is validation accuracy.

	Minimum value	**Maximum value**
Validation Accuracy	48.50%	85.28%
Fitness NMSE	−0.3297	−0.0903

Table 5.1 Federated Neuroevolution Performance on the EEG Dataset

From a fully random state, the algorithm was able to get to 85% validation accuracy as seen in Table 5.1. This is, of course, a lot less than the baseline but still a good result considering using Neuroevolution for training weights which is not the best method for training NNs.

5.5 Conclusion

In this paper we described our experiments with a simple method, what we call Federated Neuroevolution (FNE), that is an application of EA adapted for FL of NNs.

We found that our method is applicable on the studied scenario yielding some advantages over the traditional FL methods.

An advantage of EA, compared to the gradient based algorithms originated from [129], [40] or [157], is that it requires even less client data transfer to the server. While FedAVG exposes the client side data distribution and the gradients during learning, FNE only expose the amount of data points of the clients and an abstract fitness number of the model.

The clear disadvantage is that the convergence is a lot slower. We needed 5000 iterations of the algorithm to get to an 85% accuracy which is still less then the baseline's 95%. At this point though our purpose was merely to demonstrate the feasibility of derivative-free learning of NN-s in a FL scenario.

In summary, the technique we introduced, trades off learning speed for privacy gains. We may need a lot of communication rounds which can be bad in a real-world setting of mobile users, but for some use cases, like for data from medical institutions, the rounds of communication is not of primary importance, while keeping data privacy is essentials. Another aspect of techniques similar to FNE that might be interesting, is that there is no traditional, backpropagation based learning, that is at client side we can save this rather expensive stages of the learning process.

In the future we think there are several possible directions to develop FNE to make it practical. First the rather poor performance of the system might be improved through experimenting with different submethods (selection , crossover, etc.)

Following the trends in genetic algorithms, the search space could be extended to the network architecture too. This way we could reduce the bias and variance introduced by the model architecture that is chosen rather blindly at the initiation phase of the learning.

Bearing in mind the main purpose of the experiments, that is prevent the communicating the gradients, a range of derivative free methods are available as Differential Evolution [106], Particle Swarm Optimization[72] or other biology inspired methods like Artifical Bee Colony [73]. Similarly, advanced optimization methods as CMA-ES[92] might be applied.

It could be also interesting to experiment with more efficient utilization of resources, since in the current setup in each round the vast majority of nodes is idle.

Chapter 6

Peer-to-peer Federated Learning of Neural Networks

This chapter a view of FedAvg through which federated training can be turned into a decentralized, peer-to-peer-like system [125].

6.1 Motivation for decentralized FL

As we have described more detailed in Chapter 3, since, according to the problem statement of FL, the number of nodes K is extremely large, the management of these nodes even with the sub-setting that is introduced in FedAvg might be a challenging task. Consequently one of the most apparent practical issue of FedAvg is delays that stem from the centralized synchronous nature of the algorithm. Delays are introduced by *overcrowded channels* around the coordinator and by so-called *struggling nodes*, both leading to a slow-down in the training process. To solve this problem a variety of techniques have been proposed providing strategies to orchestrate updates, such that, applying *traditional scheduling techniques* [234]; using *federated client selection* [167] that prefers nodes with the best communication and computation capabilities; dynamically adapting of training scheme to available resources [224]; or allowing *asynchronous communication patterns* [44]. Another viable way for reducing communication burden seeks to decrease the size of data

to be communicated through some kind of *quantization* [189, 203, 60, 4, 5]. Methods for compressing updates also include techniques that build on characteristics of NN training, such that pruning updates [61] through variational dropout [123] or evaluating the importance of parameter layers in NNs [44].

Assuming the convexity of the loss function, a range of innovative methods has been proposed, mostly built on the "communication-efficient distributed dual coordinate ascent" (CoCoA) framework using dual optimization ([112]), with the main goal to minimize the number of communication rounds during the learning.

A different perspective to deal with communication difficulties is to *decentralize the training* across multiple parameter servers as it has been already proposed in DistBelief [51]. Going further, for convex loss functions, a number of completely peer-to-peer gossip-based [58] asynchronous algorithms has been proposed such as [94], to mention only one example. Dual optimization has been used in peer-to-peer setup as well, for example the alternating direction method of multipliers [29, 225] in [215, 18].

A second important issue in FL is the degradation of performance which might be caused by *weight divergence* [251, 240], resulting from averaging the updates, approached by so-called neuron matching [240]. Another cause of the dropped performance might be the fact that updates applied on the common model of FL can be viewed as a MBGD with updates being computed over extremely large "mini batches", that can cause serious generalization gaps [156].

In general, higher number of participating nodes, bigger local batch sizes and stronger divergence in the local data distributions lead to degraded performance and, especially for more complex tasks and models, they often prevent the system to learn any interpretable consensus model.

6.1.1 Our Contribution

The two migrating model (MM) approaches we present in this paper can be derived from FedAvg in the following way: (i) splitting the coordinator into multiple smaller processes, that is, multiple "coordinators" collect the updates from a smaller amount of nodes (per

coordinator) and (ii) instead of having a fixed node for the coordinator, the models to be trained are passed from node to node, each adding its update when it "owns" the model. Finally, (iii) to simulate the model averaging step, the updates are written in a buffer and their average will be applied on the model with a predefined frequency. (iv) We will also discuss a special case of the MM approach, in which we omit the buffer and apply the updates without delay.

The contributions of the presented MM approach are:

1. reduction of the complications of centralized FL through evening the communication burden over the whole network by simulating FedAvg in a peer-to-peer environment;

2. by reducing the number of updates used for model averaging (or excluding averaging in the special case) and, thus, using smaller effective batch sizes (at the price of more biased updates) it is possible to dramatically reduce the general communication and computation cost in exchange for, according to our empirical results, only a slightly worst training efficiency;

3. using a simple mechanism to involve those nodes in the training process, that promise the most performance gain, incentivating training on as diverse data as possible (closer to iid. wrt. whole distribution);

4. any node can initiate an optimization of new models, so the network optimizes various models in parallel, completely asynchronously.

For inference, the nodes can use some of the previously seen models, potentially in an ensemble fashion to obtain a kind of "global model", with competitive performance to FedAvg, at least according to our experiments. For performance evaluation at first place we wanted to compare our method to FedAvg over at least as many nodes, as in our methods, thus we used bagging ensembles, along with measurements for the performance of a single model.

In a real-world scenario, a *tracker* can be deployed to provide information about the network, the various models and their migration within the network.

6.2 Migrating Models (MM)

Let us assume that nodes $v^1, v^2, \ldots, v^{K'} \in V$ participating in the training are organized into a graph $G(V,E)$ and the models with parameters $\mathbf{w}^1, \ldots, \mathbf{w}^{K'}$ $(K' \leq K)$ are travelling along the edges $e \in E$ of the graph.

The initial phase of the learning algorithm starts with random initialization of K' model parameter sets $\mathbf{w}^1, \ldots, \mathbf{w}^{K'}$ at a subset of nodes (in our simulation, K' is a hyper-parameter, in the real world it might change dynamically). These initial weights need not be aligned across the nodes. Different initializations may even help to explore a bigger portion of the parameter space. As long as the input and output dimensions are aligned across the system, any kind of models can be used in the presented MM approach, whose training is done by MBGD (i.e. not only NNs).

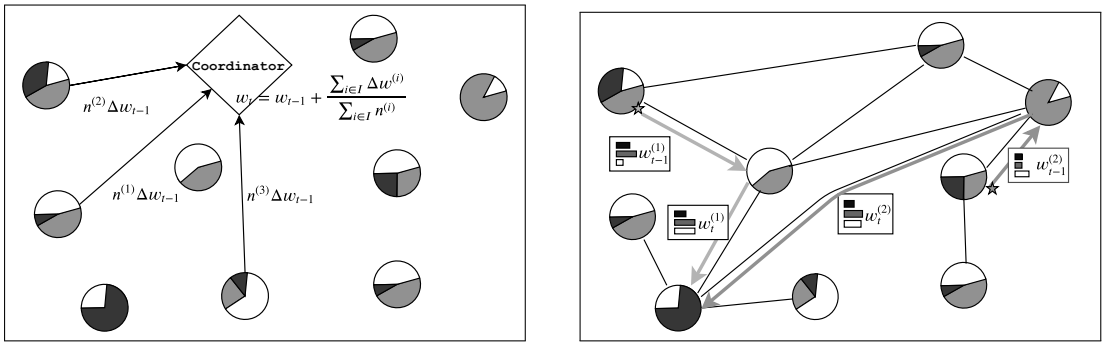


Figure 6.1 Visualisation of FedAvg (left) and MM (right) approaches. Pie charts represent data distributions at nodes with colors corresponding to classes. FedAvg randomly picks some nodes, refines the common model on them, then adds the averaged update to it. In MM, a node initiates a model (indicated by stars referring to two different models), trains it on the local data and computes the belief vector about its performance (rectangle with confidence per class as the colored bars). In the following step the model will be moved to the next available node with the largest expected gain.

Updating the models, as visualized in the Figure 6.1, happens in a completely asynchronous manner: When a model has been trained on a partial set of data, it looks for an appropriate next location to move, preferably a node, where the data is likely to help improving the model.

Choosing next location. To decide to which node should a model $\mathbf{w}^k$ be moved for further training, each model maintains a *belief vector* $\mathbf{p}^k \in \mathbb{R}^C$, that corresponds to a guess for its (per class) performance on the global data for the C classes.

All the entries of $\mathbf{p}^k$ start with 0 and, after training on a dataset, the belief vector is updated according to the Algorithm 4. The idea is to set believes to performance of the updated model on an i.i.d. test subset of local data, if the class is present, and discount the values of the absent ones, since we expect this performance to decrease. After obtaining the new belief vector, a new location for the model $\mathbf{w}^k$ is chosen, based on data sets residing at the neighboring nodes $N_G(v^k)$ from its current host node v^k. Selection of the next node to move the model to, is based on an expected performance gain, that is described in the Algorithm 5, where I is the indicator function returning 1 if its parameter is true. In this step the node broadcasts its belief vector to the neighbours, who will push up the expected performance on classes that they possess (we set it to 1 for simplicity), and send the sum of the entries back. The intuition is that the highest sum promises the most performance gain.

Algorithm 4 Updating the belief vector $\mathbf{p}^k$ of the model $\mathbf{w}^k$ based on F1-scores $\mathbf{t}^k$ on a test set of the given node k, with a discount rate ξ

procedure BELIEFUPDATE($\mathbf{p}^k = (p_1^k, p_2^k, \ldots, p_C^k), \mathbf{t^k} = (t_1^k, t_2^k, \ldots, t_C^k)$)
    $\mathbf{p^k} \leftarrow (1-\xi)\mathbf{p^k}$
    for $c \leftarrow 1$ to C **do**
        $p_c^k \leftarrow \max\{p_c^k, t_c^k\}$
    end for
    return $\mathbf{p}^k$
end procedure

Algorithm 5 Finding the best node to migrate the model to

procedure GETMAXBENEFITNODE($v^k, \mathbf{p}^k = (p_1^k, p_2^k, \ldots, p_C^k)$)
    $j^* \leftarrow \arg\max_{j | v^j \in N_G(v^k)} \underbrace{\sum_{c \in \mathcal{C}} \max\{I(\exists i \in \mathcal{D}^j (y_i = c)), p_c^k\}}_{\text{benefit at the node } v^j \text{(computed at the node } v^j)}$
    return v^{j^*}
end procedure

Update buffer. To simulate the aggregation step, for each model $\mathbf{w}^k$ we specify a buffer size σ^k that defines that, in a given "training round" r, how many nodes the model should visit before the aggregated (weighted averaged) update will be applied on its parameters $\mathbf{w}^k_{r-1}$. By updating the believes before each model relocation, we hope to get an aggregated update more similar to one that would have been resulted from an i.i.d. training run (wrt. general distribution).

Algorithm 6 Migrating models – Experimental algorithm
K' – number of models
β, η – number of epochs and the learning rate at local training

procedure TRAINING(K', β, η)
  initialize graph $G(V, E)$
  $v^1, \ldots, v^{K'} \leftarrow$ Pick K' nodes randomly from G
  initialize belief vectors $\mathbf{p}^1, \ldots, \mathbf{p}^{K'} \leftarrow \mathbf{0}$
  initialize buffer sizes $\sigma^1, \ldots, \sigma^{K'} \leftarrow 1$
  initialize $\mathbf{w}^1_0, \ldots, \mathbf{w}^{K'}_0$ randomly
  $r \leftarrow 0$
  repeat
    for all $k \in \{1, \ldots, K'\}$ **do** in parallel
      $\delta \leftarrow 0$
      count $\leftarrow 0$
      $l^k_r \leftarrow$ accuracy$(\mathbf{w}^k_r, \mathcal{D}^k)$ ▷ Test accuracy at current node's test set
      $\sigma^k \leftarrow$ UpdateBufferSize(l^k_r, σ^k) ▷ Extend the buffer size if needed
      for $s \leftarrow 1$ to σ^k **do** ▷ Collecting the σ^k updates
        $\delta \leftarrow \delta +$ ClientUpdate$(k, \mathbf{w}^k_r, \beta, \eta) * n^k$
        count $\leftarrow$ count $+ |D^k|$
        $\mathbf{t}^k \leftarrow$ Test$(\mathcal{D}^k, \mathbf{w}^k_r)$ ▷ Test the F1-score of $\mathbf{w}$ on $\mathcal{D}^k$
        $\mathbf{p}^k \leftarrow$ BeliefUpdate$(\mathbf{p}^k, \mathbf{t}^k)$
        $v^k \leftarrow$ GetMaxBenefitNode$(v^k, \mathbf{p}^k)$
      end for
      $\mathbf{w}^k_{r+1} \leftarrow \frac{\delta}{\text{count}}$
    end for
    $r \leftarrow r + 1$
  until stop
end procedure

To keep the computation and communication costs low, we start from $\sigma^k = 1$ for all $k = 1, \ldots, K'$. Then, during the training, we monitor the improvement of the loss functions l^k and, based on some heuristics, we extend the buffer if we experience that performance

Algorithm 7 Migrating models – Client update

procedure CLIENTUPDATE($k, \mathbf{w}, \beta, \eta$)
  $\mathbf{w}' \leftarrow \mathbf{w}$
  $B \leftarrow$ split $\mathcal{D}^k$ to set of batches
  for each local epoch i from 1 to β **do**
    for all $\mathcal{B} \in B$ **do**
      $\mathbf{w}' \leftarrow \mathbf{w}' - \eta \nabla^{\mathcal{B}} f(\mathbf{w})$
    end for
  end for
  return $\mathbf{w}'$
end procedure

is not improving at the expected rate[1]. The MM method is described in the Algorithm 6 and 7.

Simple Migrating Models (sMM) As a specific case of MM, we also experimented with the simplest possible setup, denoted here as sMM, where $\sigma = 1$. In this case, at the price of highly biased updates, we can further reduce the communication and training costs of MM for a single update. Besides, there are two factors which are believed to decrease the performance of FedAVG such that (i) using large batch sizes and (ii) performing model averaging. In sMM, which for models with MBGD training is equivalent to a single node MBGD, both of these factors are excluded. It might be worth to note, that since we are not averaging over the updates of the nodes, sMM can be applied for different training methods, consequently different models as well (for example decision trees).

6.3 Experiments

Data sets and models. Experiments were run with three image classification task such that `MNIST`, `Fashion-MNIST` and `CIFAR-10`. The number of nodes for the two simplest cases, `MNIST` and `Fashion-MNIST`, was set to 200. For `CIFAR-10`, due to heavy computa-

[1] A simple heuristic we have used here was to extend the buffer size if the exponential moving average of the model performance on new datasets (on the new nodes) shows no improvement after θ steps, where θ is a hyper-parameter.

tional load and the danger of stagnation in learning process, the number of nodes was set to 50. Similarly to the experimental settings in [157], 90% of each data set have been divided into equally sized one or two class chunks. 10% of this data, that has been assigned to each node, was assigned to a local test set (with the same distribution as in the local train set). The remaining 10% of the data was retained for evaluating the overall performance of the algorithms.

For the above mentioned three image classification tasks two types of NNs were utilized: For `MNIST` and `Fashion-MNIST` data sets we used a fully connected single hidden layer network (FCN) [2] For the `CIFAR-10` data set we used a convolutional neural network (CNN) [3]. Experiments with FedAvg and the proposed MM and sMM approaches were performed.

Hyper-parameters , such that the size $|\mathcal{B}|$ of batches b, the number of epochs β and the learning rate η of the optimization process were tuned using grid search with the following values: $\beta \in \{1,3,5,10\}$, $|\mathcal{B}| \in \{10,32,64\}$, $\eta \in \{1,0.1,0.01\}$ when we used FCNs and $\eta \in \{0.1,0.001,0.0001\}$ for the case of CNNs. Besides these, we run tests with different maximal buffer sizes (number of gradients to be collected) $\sigma \in \{1,3,5\}$ and also tested the general performance of bagging ensembles of a given number of models $K' \in \{1,4\}$. Here, bagging refers to predictions resulting from averaging unnormalized per-class activations of a selected subset of the models $\mathbf{w}^1, \mathbf{w}^2, \ldots, \mathbf{w}^{K'}$, that are trained in our network. These hyper-parameters were chosen in a way, that the communication and computation costs should be upper bounded by that one of FedAvg. We defined the number of maximum

[2] Following the Keras reference model for `MNIST` (not available anymore):
input: 784 dimension vector (=28 $\times$ 28) $\rightarrow$ dropout $\rightarrow$ dense with 128 units $\rightarrow$ sigmoid activation $\rightarrow$ dropout $\rightarrow$ dense with 10 units $\rightarrow$ softmax.

[3] Following the Keras reference model for `CIFAR-10` (not available anymore):
input: $32 \times 32 \times 3$ image$\rightarrow$
2d convolution with 32 3×3 filter and same padding and ReLU$\rightarrow$ 2d convolution with 32 3×3 filter and same padding and ReLU$\rightarrow$ 2×2 maxpooling $\rightarrow$ Droput$\rightarrow$
2d convolution with 64 3×3 filter and same padding and ReLU$\rightarrow$ 2d convolution with 64 3×3 filter and same padding and ReLU $\rightarrow$ 2×2 maxpooling $\rightarrow$ Droput $\rightarrow$
dense with 512 units and ReLU $\rightarrow$ Dropout $\rightarrow$ dense with 10 units $\rightarrow$ softmax

collected node updates such that the number of communication rounds will be always upper bounded by that of FedAvg [4].

Dropout. Due to the strongly skewed nature of the local data sets, after each training round the sMM models tend to overfit on the local data. Thus, a strong regularization is necessary. Therefore, we applied dropout, as described in [99], with a high probability: 0.25, 0.25, 0.5 for the layers of the used CNN, and 0.5 for the single hidden layer of the used FCN. For the control experiments, we did not use any dropout since we found out that it has a bad impact on FedAvg. Thus, for the buffered learning we decreased the dropout rate π of the model $\mathbf{w}^k$ by a factor σ^k, i.e. $\pi^k = \frac{\pi}{\sigma^k}$.

Graph. We used a full graph topology, where for simulating a more realistic network, we randomly picked 5 nodes and chose the most promising relocation targets among those. For the discount rate at the belief update step we used $\xi = 0.05$.

6.3.1 Results

Results are summarized in Table 6.1 and visualized in Figures 6.2, 6.3 and 6.4. The data series of these figures show the average performance of the 10 best results for the four versions of federated learning, namely, (i) MM with bagging ensembles (denoted as MM), (ii) the FedAvg baseline (FedAvg), (iii) sMM with bagging ensembles (sMM), and, (iv) MM with only one single model (Single MM). On the left side of these figures the performance comparison in terms of accuracy is shown, the middle charts depict the communication costs (i.e. how many times the weights of the model had to be forwarded) while on the right side the corresponding computational costs of these algorithms are showed.

The computation costs has been calculated for a training round r by multiplying N_r, the number of nodes participating in the training in the round (that is $N_r = K'$, with K' being our initial choice in sMM, $N_r = \sum_{k=1}^{K'} \sigma_r^k$ in MM, while $N_r = K' = \gamma K$ for FedAvg), and the

[4] For the buffer size extension we set a threshold for improvement to $\theta = 15$, that is, if the accuracy did not improve in the last 15 relocations then we extend the number of models to aggregate.

number of epochs β. Thus, the accumulated computation costs at round r are computed by $\text{cost}_r^{\text{comp}} = \text{cost}_{r-1}^{\text{comp}} + N_r * \beta$ The communication cost values were calculated in a similar way, such that $\text{cost}_r^{\text{comm}} = \text{cost}_{r-1}^{\text{comm}} + \omega_r$, where ω is the number of transmissions of the model weights, that is necessary for updating all trained models. For sMM, $\omega_r = N_r$ and, for MM, $\omega_r = \sum_{k=1}^{K'} \sigma_r^k$. That equals to the number of participating nodes in a training round, because collecting the gradients from σ_r^k nodes means the same number of transmissions. The value of ω for FedAvg is, on the other hand, $2 * N_r$ since each active node has to first acquire the recent model and then to send back the updates. At these values, we calculated the number of transmissions necessary only for the training and did not include the costs of broadcasting the common models for inference to each node.

The performance of the algorithms is measured via accuracy, since for the MM and sMM methods averaging ensembles were used for making predictions.

Results have shown that the sMM algorithm is very simple and viable method to train NNs at a very low computational and communication cost. Its performance, however, stays under the performance of the FedAvg baseline, even with ensembles.

The MM approach closed this gap in a trade-off for increasing costs. We believe, however, that producing a similar performance to FedAvg with a decentralized algorithm, such that the proposed MM, is promising[5] [6].

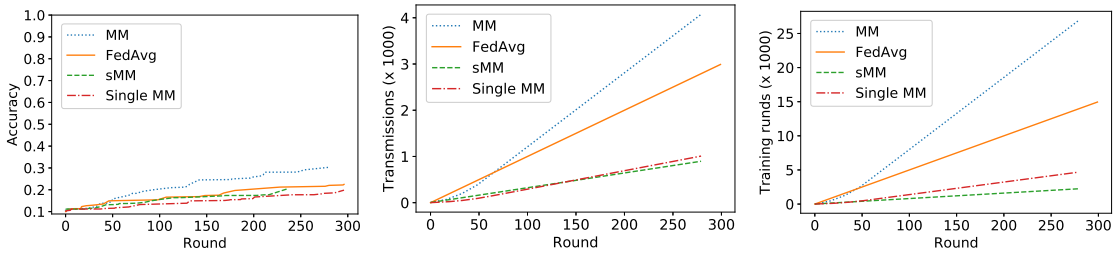


Figure 6.2 Accuracy, Communication cost and Training cost on `CIFAR-10`

[5]The accuracy, along with the communication and computation costs, of ensemble of MMs exceeds that one of FedAvg. The reason for that is that, due to resource intensity of `CIFAR-10` training, the number of participating nodes have been reduced to 50, which means fewer models have been used in FedAvg, while the hyper-parameters (K' and maximum σ) of sMM and MM have been kept unchanged.

[6]Hyper-parameter search was not performed for the γ parameter of FedAvg, $\gamma = 1/10$ was used according to [157]. The main reason was our limited computation possibilities. However, since the settings for γ also affect sMM and MM (e.g. the buffer size or the ensemble count), it is possible that the gap between the communication and computation costs might be different in case of a large-scale hyper-parameter search.

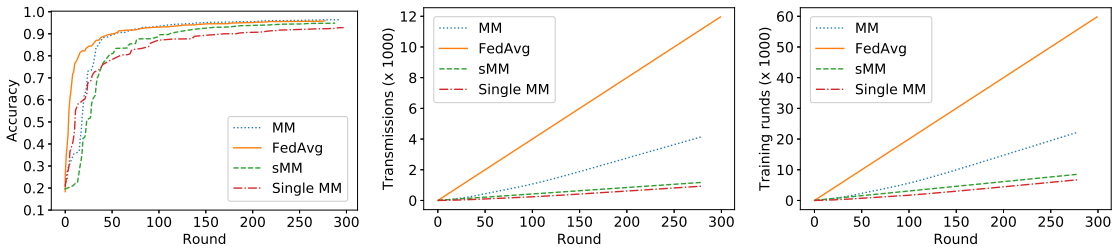


Figure 6.3 Accuracy, Communication cost and Training cost on `MNIST`

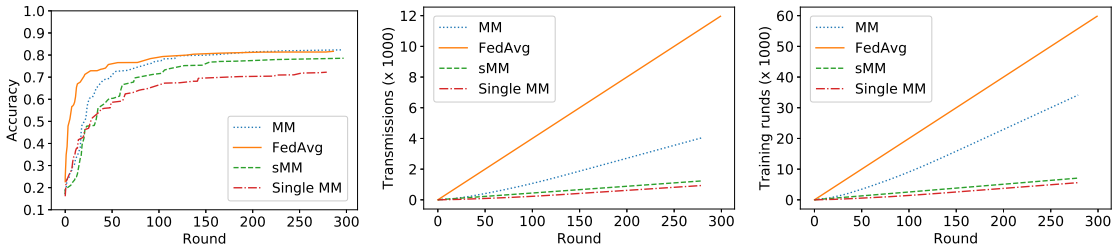


Figure 6.4 Accuracy, Communication cost and Training cost on `Fashion-MNIST`

	Accuracy				Parameter transmission ($\times 10^4$)				Training epochs($\times 10^4$)			
	Fed Avg	MM	sMM	Single MM	Fed Avg	MM	sMM	Single MM	Fed Avg	MM	sMM	Single MM
C10	0.22	**0.30**	0.2	0.19	14.95	26.90	**2.24**	4.67	2.99	4.07	**0.89**	1.00
MN	0.95	**0.96**	0.94	0.92	59.8	22.27	8.54	**6.74**	11.96	4.14	1.17	**0.92**
F-M	0.81	**0.82**	0.78	0.72	59.8	34.14	7.08	**5.59**	11.96	4.03	1.23	**0.93**

Table 6.1 The averages of best 10 results in terms of accuracy of each methods, with the corresponding communication and computations costs. Bold fonts denote the best values i.e. best accuracy, lowest communication and computation cost.

6.4 Discussion

Besides the promising results of the experiments, a few issues have to be mentioned concerning the presented methods, though. These are the following:

Ensembles A disadvantage of the proposed methods is the increased resource demand of the ensembles in inference time at the nodes, if one decides to boost performance this way. This could be alleviated trough distillation ([98]), however, at the price of additional computation load. Another question is, how a node acquires the necessary number of high

quality models for inference. In our experiment, we trained the exact number of models that constitute the predictor and used each model in the tested ensemble.

Stagnation The potentially advantageous effect of omitting model averaging is probably balanced out with the loss of its regularization effect. Despite of the strong regularization, the local models overfit on the most recent data sets. This leads, in a significant number of cases, to a stagnating ensemble accuracy. However, we experienced a similar phenomenon in the case of FedAvg as well. An analysis for the poor performance of FedAvg in strongly skewed data sets can be found for example in [103] or in [220].

Real world usage In our experiment, we have been working with a fully connected graph, that is naturally unrealistic in real world scenarios. The distribution of data over the nodes, along with the number of nodes, is pretty far from the characteristics given by [129]. However, using similar set-ups is a common practice as we have seen in the literature.

6.4.1 Convergence

The proposed models can be viewed as generalizations of FedAvg: For $K' = 1$ and a constant gradient buffer size $\sigma = \gamma K$, in the case of a fully connected graph, the MM method becomes equivalent to FedAvg. Moreover, also in the fully connected case, $\gamma = 1/K$ (in case of FedAvg) and $K' = 1$ and $\pi = 0$ (for MM), leads to the same method, apart from involvement of the parameter server at FedAvg. Finally, if we limit $\sigma = 1$ for MM, we get the sMM algorithm.

Uncertain convergence To carry out a thorough analysis of convergence of NN training in a FL setting involves many variables which make giving meaningful guarantees extremely hard. A lot of effort [224, 141, 142, 252, 202, 221, 227, 121, 239] have been carried out in this direction, however, due to the complexity of the problem, all of them make certain restrictions. Even for the case of convex optimization, there are assumptions such that iid data distribution across the nodes or all the devices being active (the latter is equivalent to FedSGD). The latter assumption was made in [121, 239, 224], while authors

of [252, 202, 221, 227] build on both. [141] and [142] provides convergence analysis for true FedAvg with non-iid data, but for the case of a strongly convex optimization objective, not applicable for the case of NNs.

Convex case Following the reasoning of [142], for convex optimization, the convergence rate of FedAvg is $\mathcal{O}(\frac{1}{n})$, where n is the total number of data points which contribute to the optimization. According to their analysis, to achieve an accuracy of ε, the number of training round to execute is

$$\frac{n}{E} = \mathcal{O}\left[\frac{1}{\varepsilon}\left(\left(1+\frac{1}{K'}\right)E\overline{||\nabla l||^2} + \frac{\sum_{k=1}^{K}\gamma^{k2}\overline{Var(\nabla l^k)} + \Gamma + \overline{||\nabla l||^2}}{E}\right)\right] \tag{6.1}$$

where $E = \beta|\mathcal{B}|$ is the number of data points, from which the updates are computed between two communication rounds, γ^k is the probability of selecting the node k for the update round, $\Gamma = F^* - \sum_{k=1}^{K} p^k F^{k*}$ quantifies "non-iid-ness" that is the difference of globally optimal loss and the local optima. $\overline{Var(\nabla l^k)}$ is a bound for variance, while $\overline{||\nabla l||^2}$ stands for squared norm of local stochastic gradients.

6.4.2 Privacy

An important challenge in FL, and distributed machine learning (ML) in general, is the question of privacy of potentially sensitive data. In our setup we see the following two major points for potential attacks:

Forward inference If we use the method of Algorithm 5, it is feasible to combinatorically infer classes of data held at the candidate nodes, granting an additional vulnerability to our method compared to FedAvg. However, applying a random noise over the indicator function, apart from some extreme situations, considerably decreases the vulnerability of the proposed approach.

Backward inference According to our best knowledge, as it is summarized in [101], attacks on privacy in distributed ML build on regular and frequent update messages following the same routes. Peer-to-peer nature of the training process adds an additional complexity to deal with, making it necessary to have access to all communication channels of the attacked node to achieve similar effectiveness of an attack. Gradient leakage attacks [250] can be executed having access to a single update vector, currently however they work only on single batch updates, or multiple batches with very few examples included. For both cases the tracker can be used to ensure legitimacy of routes, for example, to avoid building loops around the target or even to redraw the connection graph from time to time.

6.5 Conclusion

We presented our approach to alleviate the challenges imposed by the federated learning setup and, in general, distributed machine learning systems. The key idea of the proposed approach is that, instead of the transmission of model updates, the models themselves travel to the location of the data, evening the communication needs across the network.

With this (almost complete) decentralization of the learning process, the synchronization problems and straggler effect can be bypassed as well as communication burden at the parameter servers is not present anymore.

Our experiments have shown that similar performance can be achieved compared to the federated averaging baseline, however, with less communication and computational cost (at the price of using ensembles of small number of models at prediction phase). Based on the results of our experiments and the fact that the proposed approach is a generalization of the popular federated averaging approach, the presented work is worth further research.

Chapter 7

Conclusions

In this dissertation we presented our ideas to address the problems of FL, that we perceived to be the biggest issues of the method. Namely:

- Privacy issues,
- Performance degradation, and
- Communication bottlenecks.

In general we can summarize our work as trying to give as simple solutions to these problems as possible.

Privacy issues - Genetic algorithms in FL In Chapter 5 we presented our experiments with gradient free training of NNs. According to our results [208], these nature inspired methods might actually be able to fit complex models, even if they naturally proceed very slowly. Nevertheless full exclusion of backpropagation has the advantages of reducing the local training at small user equipment to simple loss calculation, and making privacy attacks, that are based on capturing the updates impossible. We believe that examining the effect of this method on training data memorization and the stemming membership inference attacks might be an interesting research direction in the future.

Returning to more conventional learning methods, based on the train of thought, that we presented in Chapter 4, FedAvg can approximately be viewed as MBGD training, with extremely large batch sizes. Following this assumption the experiments of Chapter 4 and 6 correspond to testing techniques in the setup of FL, that are widely used to facilitate single node training, such that stateful optimization methods and hyperparameter-tuning.

Performance degradation - Stateful optimization Our second direction of research, that has been presented in Chapter 4, aimed at increasing the performance of federated learning, using stateful optimization methods as RMSProp, Adam, or AdaGrad. These have been originally designed to overcome similar problems for single machine MBGD, through adapting the step sizes of the optimization, and/or reducing the variance of the update directions. When these methods were applied at central aggregation [126], we experienced significant improvement in the speed of learning. Curiously faster training also happens when these methods were applied at the nodes only [64], that presumably helped the local training to find better optima given the local data sets. This eventually seems to help the performance of the averaged model as well.

Communication bottlenecks- Peer-to-peer FL In our experiments that have been presented in Chapter 6 we presented a way to turn FedAvg into a peer-to-peer process, by effectively passing the role of the coordinator from node to node, and collecting the individual updates into a buffer, before they get averaged and applied to the parameters. This method (at least in our setup) was able to produce similar performance to baseline of FedAvg, while strongly reduced the computation and communication burden of the training. With allowing extreme hyper-parametrization of FedAvg (namely the pushing the number of participating nodes down to a single one, and having the possibility to do so with the number of local training epochs too), apart from the iid-ness assumption over batches, we can virtually reproduce a single node MBGD training 6. Thus apparently reducing the effective batch sizes of updates in FL could balance out the negative effect of updates computed on the skewed data sets form a very few, or even from a single node.

There is an opinion recently in the scientific community, according to which the methods of NN training need fundamental improvement. In these opinions, the insufficiency of recent training methods is residing in the phenomenon, that NN models eventually act as feature encoders, basically memorizing the training data, and they behave as kernel machines [59].

Following these assumptions, the presence of adversarial examples, the threat of membership inference and model inversion attacks, or the under-performance of models in real-world environments [50] are all symptoms of this insufficiency, namely the inability to model properly the underlying data manifolds. Since we are not able to communicate properly what we actually want the models to learn, they tend to find *shortcuts*, that is features, that are the easiest to learn and produce the best performance on the training data [78, 108, 77, 71]. These phenomenons also show, that recent accuracy-based performance evaluations and/or regularization techniques are just simply not sufficient to force the networks to generalize well enough, and thus learn the features, that a human would expect to work properly.

We believe therefore, that the future directions of progress in NN training and FL will reside in methods that results in more simple, explainable models, that will also need way less training data (for example following the ideas of [96]). Also better training methods might rely less on specific training examples, thus revealing less information specific to those, both during inference (membership inference) and training time (gradient decryption).

Bibliography

[cif] Keras reference model for cifar-10. https://keras.io/examples/cifar10_cnn/. Accessed: 2020-02-04.

[mni] Keras reference model for mnist. https://keras.io/examples/mnist_mlp/. Accessed: 2020-02-04.

[3] Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016). Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM.

[4] Aji, A. F. and Heafield, K. (2017). Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021*.

[5] Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. (2017). Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pages 1709–1720.

[6] Almadhoun, N., Ayday, E., and Ulusoy, Ö. (2020). Inference attacks against differentially private query results from genomic datasets including dependent tuples. *Bioinformatics*, 36(Supplement_1):i136–i145.

[7] Amiri, M. M. and Gündüz, D. (2020a). Federated learning over wireless fading channels. *IEEE Transactions on Wireless Communications*.

[8] Amiri, M. M. and Gündüz, D. (2020b). Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air. *IEEE Transactions on Signal Processing*, 68:2155–2169.

[9] Anil, R., Pereyra, G., Passos, A., Ormandi, R., Dahl, G. E., and Hinton, G. E. (2018). Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235*.

[10] Aono, Y., Hayashi, T., Wang, L., Moriai, S., et al. (2017a). Privacy-preserving deep learning: Revisited and enhanced. In *International Conference on Applications and Techniques in Information Security*, pages 100–110. Springer.

[11] Aono, Y., Hayashi, T., Wang, L., Moriai, S., et al. (2017b). Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345.

[12] Aslan, Ö., Zhang, X., and Schuurmans, D. (2014). Convex deep learning via normalized kernels. In *Advances in Neural Information Processing Systems*, pages 3275–3283.

[13] Ateniese, G., Mancini, L. V., Spognardi, A., Villani, A., Vitali, D., and Felici, G. (2015). Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks*, 10(3):137–150.

[14] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

[15] Babanezhad, R., Ahmed, M. O., Virani, A., Schmidt, M., Konečnỳ, J., and Sallinen, S. (2015). Stop wasting my gradients: Practical svrg. *arXiv preprint arXiv:1511.01942*.

[16] Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V. (2020). How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR.

[17] Begleiter, H. (1995). Eeg database data set. *New York, NY: Neurodynamics Laboratory, State University of New York Health Center Brooklyn*.

[18] Bellet, A., Guerraoui, R., Taziki, M., and Tommasi, M. (2017). Personalized and private peer-to-peer machine learning. *arXiv preprint arXiv:1705.08435*.

[19] Bernau, D., Grassal, P.-W., Robl, J., and Kerschbaum, F. (2019). Assessing differentially private deep learning with membership inference. *arXiv preprint arXiv:1912.11328*.

[20] Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., and Anandkumar, A. (2018). signsgd: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*.

[21] Bersini, H. and Seront, G. (1992). In search of a good crossover between evolution and optimization. *M anner and Manderick*, 1503:479–488.

[22] Bittau, A., Erlingsson, Ú., Maniatis, P., Mironov, I., Raghunathan, A., Lie, D., Rudominer, M., Kode, U., Tinnes, J., and Seefeld, B. (2017). Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 441–459.

[23] Blatt, D., Hero, A. O., and Gauchman, H. (2007). A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51.

[24] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*.

[25] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konecny, J., Mazzocchi, S., McMahan, H. B., et al. (2019). Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*.

[26] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. (2017). Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191.

[27] Bordes, A., Bottou, L., and Gallinari, P. (2009). Sgd-qn: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10:1737–1754.

[28] Bottou, L. (1998). Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142.

[29] Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122.

[30] Bradley, J. K., Kyrola, A., Bickson, D., and Guestrin, C. (2011). Parallel coordinate descent for l1-regularized loss minimization. *arXiv preprint arXiv:1105.5379*.

[31] Bregman, L. M. (1967). The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR computational mathematics and mathematical physics*, 7(3):200–217.

[32] Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541. ACM.

[33] Byrd, R. H., Hansen, S. L., Nocedal, J., and Singer, Y. (2016). A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031.

[34] Caldas, S., Konečny, J., McMahan, H. B., and Talwalkar, A. (2018). Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*.

[35] Carlini, N., Mishra, P., Vaidya, T., Zhang, Y., Sherr, M., Shields, C., Wagner, D., and Zhou, W. (2016). Hidden voice commands. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 513–530.

[36] Carlini, N. and Wagner, D. (2018). Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE.

[37] Caruana, R., Niculescu-Mizil, A., Crew, G., and Ksikes, A. (2004). Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18. ACM.

[38] Chaudhuri, K., Monteleoni, C., and Sarwate, A. D. (2011). Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(Mar):1069–1109.

[39] Chaum, D. L. (1981). Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90.

[40] Chen, J., Pan, X., Monga, R., Bengio, S., and Jozefowicz, R. (2016). Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*.

[41] Chen, X., Liu, C., Li, B., Lu, K., and Song, D. (2017). Targeted backdoor attacks on deep learning systems using data poisoning.

[42] Chen, Y. (2019a). Accelerated gradient methods [lecture slides for class :ele 522: Large-scale optimization for data science]. University Lecture.

[43] Chen, Y. (2019b). Mirror descent, large-scale optimization for data science. Lecture slide.

[44] Chen, Y., Sun, X., and Jin, Y. (2019). Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE Transactions on Neural Networks and Learning Systems*.

[45] Chilimbi, T., Suzue, Y., Apacible, J., and Kalyanaraman, K. (2014). Project adam: Building an efficient and scalable deep learning training system. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 571–582.

[46] Cireşan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*.

[47] Corinzia, L. and Buhmann, J. M. (2019). Variational federated multi-task learning. *CoRR*, abs/1906.06268.

[48] Courbariaux, M. and Bengio, Y. (2016). Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830.

[49] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

[50] D'Amour, A., Heller, K., Moldovan, D., Adlam, B., Alipanahi, B., Beutel, A., Chen, C., Deaton, J., Eisenstein, J., Hoffman, M. D., et al. (2020). Underspecification presents challenges for credibility in modern machine learning. *arXiv preprint arXiv:2011.03395*.

[51] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. (2012). Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231.

[52] Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654.

[53] Deng, L., Yu, D., and Platt, J. (2012). Scalable stacking and learning for building deep architectures. In *2012 IEEE International conference on Acoustics, speech and signal processing (ICASSP)*, pages 2133–2136. IEEE.

[54] Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and de Freitas, N. (2013). Predicting parameters in deep learning. *CoRR*, abs/1306.0543.

[55] Desell, T. (2017). Large scale evolution of convolutional neural networks using volunteer computing. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 127–128. ACM.

[56] Dietterich, T. G. (2000). Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer.

[57] Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654.

[58] Dimakis, A. G., Kar, S., Moura, J. M., Rabbat, M. G., and Scaglione, A. (2010). Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98(11):1847–1864.

[59] Domingos, P. (2020). Every model learned by gradient descent is approximately a kernel machine. *arXiv preprint arXiv:2012.00152*.

[60] Dryden, N., Moon, T., Jacobs, S. A., and Van Essen, B. (2016). Communication quantization for data-parallel training of deep neural networks. In *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, pages 1–8. IEEE.

[61] Du, W., Zeng, X., Yan, M., and Zhang, M. (2018). Efficient federated learning via variational dropout. *Openreview, withdrawn*.

[62] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[63] Dwork, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407.

[64] Felbab, V., Kiss, P., and Horváth, T. (2019). Optimization in federated learning. In *CEUR Workshop Proceedings (CEUR-WS.org), ISSN: 1613-0073.*, volume 2473, pages 58–65. ceur-ws.org.

[65] Fercoq, O. and Richtárik, P. (2015). Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023.

[66] Finlayson, S. G., Chung, H. W., Kohane, I. S., and Beam, A. L. (2018). Adversarial attacks against medical deep learning systems. *arXiv preprint arXiv:1804.05296*.

[67] Fredrikson, M., Jha, S., and Ristenpart, T. (2015). Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333.

[68] Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D., and Ristenpart, T. (2014). Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 17–32.

[69] Gad, A. (2019). Artificial neural networks optimization using genetic algorithm with python. *Towards Data Science*.

[70] Ganju, K., Wang, Q., Yang, W., Gunter, C. A., and Borisov, N. (2018). Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–633.

[71] Gardner, M., Artzi, Y., Basmova, V., Berant, J., Bogin, B., Chen, S., Dasigi, P., Dua, D., Elazar, Y., Gottumukkala, A., et al. (2020). Evaluating models' local decision boundaries via contrast sets. *arXiv preprint arXiv:2004.02709*.

[72] Garro, B. A., Sossa, H., and Vazquez, R. A. (2009). Design of artificial neural networks using a modified particle swarm optimization algorithm. In *2009 International Joint Conference on Neural Networks*, pages 938–945. IEEE.

[73] Garro, B. A., Sossa, H., and Vázquez, R. A. (2011). Artificial neural network synthesis by means of artificial bee colony (abc) algorithm. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 331–338. IEEE.

[74] Gathercole, C. and Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In *International Conference on Parallel Problem Solving from Nature*, pages 312–321. Springer.

[75] Gauci, J. and Stanley, K. (2007). Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 997–1004. ACM.

[76] Geiping, J., Bauermeister, H., Dröge, H., and Moeller, M. (2020). Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*.

[77] Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. (2020). Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673.

[78] Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. (2018). Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*.

[79] Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178.

[80] Goldenbaum, M. and Stanczak, S. (2013). Robust analog function computation via wireless multiple-access channels. *IEEE Transactions on Communications*, 61(9):3863–3877.

[81] Gonçalves, I. and Silva, S. (2011). Experiments on controlling overfitting in genetic programming. In *15th Portuguese conference on artificial intelligence (EPIA 2011)*, pages 10–13.

[82] Gonçalves, I. and Silva, S. (2013). Balancing learning and overfitting in genetic programming with interleaved sampling of training data. In *European Conference on Genetic Programming*, pages 73–84. Springer.

[83] Gonçalves, I., Silva, S., Melo, J. B., and Carreiras, J. M. (2012). Random sampling technique for overfitting control in genetic programming. In *European Conference on Genetic Programming*, pages 218–229. Springer.

[84] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[85] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[86] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

[87] Gower, R., Goldfarb, D., and Richtárik, P. (2016). Stochastic block bfgs: Squeezing more curvature out of data. In *International Conference on Machine Learning*, pages 1869–1878. PMLR.

[88] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

[89] Graves, A. (2011). Practical variational inference for neural networks. *Advances in neural information processing systems*, 24:2348–2356.

[90] Gu, T., Dolan-Gavitt, B., and Garg, S. (2017). Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*.

[91] Halevi, S., Lindell, Y., and Pinkas, B. (2011). Secure computation on the web: Computing without simultaneous interaction. In *Annual Cryptology Conference*, pages 132–150. Springer.

[92] Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18.

[93] Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D. (2018). Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*.

[94] Hegedűs, I., Danner, G., and Jelasity, M. (2019). Gossip learning as a decentralized alternative to federated learning. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 74–90. Springer.

[95] Hestenes, M. R., Stiefel, E., et al. (1952). Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436.

[96] Hinton, G. (2021). How to represent part-whole hierarchies in a neural network. *arXiv preprint arXiv:2102.12627*.

[97] Hinton, G., Srivastava, N., and Swersky, K. (2012a). Neural networks for machine learning. *Coursera, video lectures*, 264.

[98] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

[99] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

[100] Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13.

[101] Hitaj, B., Ateniese, G., and Perez-Cruz, F. (2017). Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–618.

[102] Hoffer, E., Hubara, I., and Soudry, D. (2017). Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741.

[103] Hsu, T.-M. H., Qi, H., and Brown, M. (2019). Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*.

[104] Hui, J. (2019). Machine learning – fundamental.

[105] Humphries, T., Rafuse, M., Tulloch, L., Oya, S., Goldberg, I., Hengartner, U., and Kerschbaum, F. (2020). Differentially private learning does not bound membership inference. *arXiv preprint arXiv:2010.12112*.

[106] Ilonen, J., Kamarainen, J.-K., and Lampinen, J. (2003). Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters*, 17(1):93–105.

[107] Ilyas, A., Engstrom, L., Athalye, A., and Lin, J. (2018). Black-box adversarial attacks with limited queries and information. In *International Conference on Machine Learning*, pages 2137–2146. PMLR.

[108] Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. (2019). Adversarial examples are not bugs, they are features. *arXiv preprint arXiv:1905.02175*.

[109] Ishai, Y., Kilian, J., Nissim, K., and Petrank, E. (2003). Extending oblivious transfers efficiently. In *Annual International Cryptology Conference*, pages 145–161. Springer.

[110] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713.

[111] Jacob, L., Vert, J.-p., and Bach, F. R. (2009). Clustered multi-task learning: A convex formulation. In *Advances in neural information processing systems*, pages 745–752.

[112] Jaggi, M., Smith, V., Takác, M., Terhorst, J., Krishnan, S., Hofmann, T., and Jordan, M. I. (2014). Communication-efficient distributed dual coordinate ascent. In *Advances in neural information processing systems*, pages 3068–3076.

[113] Jayaraman, B. and Evans, D. (2019). Evaluating differentially private machine learning in practice. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1895–1912.

[114] Jayaraman, B., Wang, L., Knipmeyer, K., Gu, Q., and Evans, D. (2020). Revisiting membership inference under realistic assumptions. *arXiv preprint arXiv:2005.10881*.

[115] Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.-M., and Van Steen, M. (2007). Gossip-based peer sampling. *ACM Transactions on Computer Systems (TOCS)*, 25(3):8.

[116] Jeong, E., Oh, S., Kim, H., Park, J., Bennis, M., and Kim, S.-L. (2018). Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *arXiv preprint arXiv:1811.11479*.

[117] Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 26, pages 315–323. Curran Associates, Inc.

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

[119] Kashmar, A. (2018). *Numerical Analysis Course/ Chapter 1: Root Finding*.

[120] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

[121] Khaled, A., Mishchenko, K., and Richtárik, P. (2019). First analysis of local gd on heterogeneous data. *arXiv preprint arXiv:1909.04715*.

[122] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[123] Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583.

[124] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[125] Kiss, P. and Horváth, T. (in press). Migrating models: A decentralized view on federated learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer.

[126] Kiss, P., Horváth, T., and Felbab, V. (2020). Stateful optimization in federated learning of neural networks. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 348–355. Springer, Cham.

[127] Kiss, P., Reale, A., Ferrari, C. J., and Istenes, Z. (2018). Deployment of iot applications on 5g edge. In *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, pages 1–9. IEEE.

[128] Konečnỳ, J., Liu, J., Richtárik, P., and Takáč, M. (2015). Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):242–255.

[129] Konečnỳ, J., McMahan, H. B., Ramage, D., and Richtárik, P. (2016). Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*.

[130] Konecný, J., McMahan, H. B., Ramage, D., and Richtárik, P. (2016). Federated optimization: Distributed machine learning for on-device intelligence. *CoRR*, abs/1610.02527.

[131] Konečnỳ, J. and Richtárik, P. (2013). Semi-stochastic gradient descent methods. *arXiv preprint arXiv:1312.1666*.

[132] Konečnỳ, J. and Richtárik, P. (2018). Randomized distributed mean estimation: Accuracy vs. communication. *Frontiers in Applied Mathematics and Statistics*, 4:62.

[133] Konečný, J., McMahan, H. B., Yu, F. X., Richtarik, P., Suresh, A. T., and Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*.

[134] Koza, J. R. and Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

[135] Kwon, A., Lazar, D., Devadas, S., and Ford, B. (2016). Riffle: An efficient communication system with strong anonymity. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115–134.

[136] Lam, H., Ling, S., Leung, F. H., and Tam, P. K.-S. (2001). Tuning of the structure and parameters of neural network using an improved genetic algorithm. In *IECON'01. 27th Annual Conference of the IEEE Industrial Electronics Society (Cat. No. 37243)*, volume 1, pages 25–30. IEEE.

[137] Langdon, W. (2011). Minimising testing in genetic programming. *RN*, 11(10):1.

[138] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer.

[139] Lee, Y. T. and Sidford, A. (2013). Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *2013 ieee 54th annual symposium on foundations of computer science*, pages 147–156. IEEE.

[140] Li, F. and Liu, B. (2016). Ternary weight networks. *CoRR*, abs/1605.04711.

[141] Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. (2018). Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*.

[142] Li, X., Huang, K., Yang, W., Wang, S., and Zhang, Z. (2019). On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*.

[143] 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. In *Advances in Neural Information Processing Systems*, pages 5330–5340.

[144] Liu, C., Chakraborty, S., and Mittal, P. (2016). Dependence makes you vulnberable: Differential privacy under dependent tuples. In *NDSS*, volume 16, pages 21–24.

[145] Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528.

[146] Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. (2017a). Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*.

[147] Liu, S., Pan, S. J., and Ho, Q. (2017b). Distributed multi-task relationship learning. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 937–946. ACM.

[148] Liu, W., Chen, L., Chen, Y., and Zhang, W. (2019). Accelerating federated learning via momentum gradient descent. *arXiv preprint arXiv:1910.03197*.

[149] Liu, Y. and Khoshgoftaar, T. (2004). Reducing overfitting in genetic programming models for software quality classification. In *Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings.*, pages 56–65. IEEE.

[150] Long, Y., Bindschaedler, V., and Gunter, C. A. (2017). Towards measuring membership privacy. *arXiv preprint arXiv:1712.09136*.

[151] Lowd, D. and Meek, C. (2005). Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647.

[152] Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. *arXiv preprint arXiv:1709.02540*.

[153] Ma, C., Konečnỳ, J., Jaggi, M., Smith, V., Jordan, M. I., Richtárik, P., and Takáč, M. (2017). Distributed optimization with arbitrary local solvers. *Optimization Methods and Software*, 32(4):813–848.

[154] Mairal, J., Koniusz, P., Harchaoui, Z., and Schmid, C. (2014). Convolutional kernel networks. In *Advances in neural information processing systems*, pages 2627–2635.

[155] Maniezzo, V. (1994). Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on neural networks*, 5(1):39–53.

[156] Masters, D. and Luschi, C. (2018). Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*.

[157] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., et al. (2016). Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*.

[158] McMahan, H. B., Ramage, D., Talwar, K., and Zhang, L. (2017). Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*.

[159] Minka, T. P. (2013). Expectation propagation for approximate bayesian inference. *arXiv preprint arXiv:1301.2294*.

[160] Mitchell, T. M. et al. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877.

[161] Mohri, M., Sivek, G., and Suresh, A. T. (2019). Agnostic federated learning. *arXiv preprint arXiv:1902.00146*.

[162] Moritz, P., Nishihara, R., and Jordan, M. (2016). A linearly-convergent stochastic l-bfgs algorithm. In *Artificial Intelligence and Statistics*, pages 249–258. PMLR.

[163] Nazer, B. and Gastpar, M. (2007). Computation over multiple-access channels. *IEEE Transactions on information theory*, 53(10):3498–3516.

[164] Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate $o(1/k^2)$. In *Sov. Math. Dokl*, volume 27.

[165] Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362.

[166] Nesterov, Y. and Nemirovskii, A. (1994). *Interior-point polynomial algorithms in convex programming*. SIAM.

[167] Nishio, T. and Yonetani, R. (2018). Client selection for federated learning with heterogeneous resources in mobile edge. *CoRR*, abs/1804.08333.

[168] Oullette, R., Browne, M., and Hirasawa, K. (2004). Genetic algorithm optimization of a convolutional neural network for autonomous crack detection. In *Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753)*, volume 1, pages 516–521. IEEE.

[169] Polson, N. G., Sokolov, V., et al. (2017). Deep learning: a bayesian perspective. *Bayesian Analysis*, 12(4):1275–1304.

[170] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

[171] Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279.

[172] Reale, A., Kiss, P., Ferrari, C., Kovács, B., Szilágyi, L., and TÓTH, M. (2018). Application functions placement optimization in a mobile distributed cloud environment. *Studia Informatica*, (2):37–52.

[173] Reale, A., Kiss, P., Tóth, M., and Horváth, Z. (2019). Designing a decentralized container based fogcomputing framework for task distribution and management. *International Journal of Computers and Communications*, 13:1–7.

[174] Reddi, S. J., Konečnỳ, J., Richtárik, P., Póczós, B., and Smola, A. (2016). Aide: Fast and communication efficient distributed optimization. *arXiv preprint arXiv:1608.06879*.

[175] Reid, M. (2013). Meeting the bergman divergences.

[176] Reyzin, L., Smith, A. D., and Yakoubov, S. (2018). Turning hate into love: Homomorphic ad hoc threshold encryption for scalable mpc. *IACR Cryptol. ePrint Arch.*, 2018:997.

[177] Ribeiro, M. T., Wu, T., Guestrin, C., and Singh, S. (2020). Beyond accuracy: Behavioral testing of nlp models with checklist. *arXiv preprint arXiv:2005.04118*.

[178] Richtárik, P. and Takáč, M. (2014). Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1):1–38.

[179] Richtárik, P. and Takáč, M. (2016). Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1-2):433–484.

[180] Romberg, J. (2017). A second look at duality [lecture slides for class : Ece 8823a, convex optimization: Theory, algorithms, and applications]. University Lecture.

[181] Roth, E., Noble, D., Falk, B. H., and Haeberlen, A. (2019). Honeycrisp: large-scale differentially private aggregation without a trusted core. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 196–210.

[182] Roux, N. L., Schmidt, M., and Bach, F. (2012). A stochastic gradient method with an exponential convergence rate for finite training sets. *arXiv preprint arXiv:1202.6258*.

[183] Rudin, L. I., Osher, S., and Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268.

[184] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

[185] Salem, A., Zhang, Y., Humbert, M., Berrang, P., Fritz, M., and Backes, M. (2018). Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246*.

[186] Sattler, F., Müller, K.-R., and Samek, W. (2019). Clustered federated learning: Model-agnostic distributed multi-task optimization under privacy constraints. *arXiv preprint arXiv:1910.01991*.

[187] Schirrmeister, R. T., Springenberg, J. T., Fiederer, L. D. J., Glasstetter, M., Eggensperger, K., Tangermann, M., Hutter, F., Burgard, W., and Ball, T. (2017). Deep learning with convolutional neural networks for eeg decoding and visualization. *Human brain mapping*, 38(11):5391–5420.

[188] Schmidt, M., Le Roux, N., and Bach, F. (2017). Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112.

[189] Seide, F., Fu, H., Droppo, J., Li, G., and Yu, D. (2014). 1-bit stochastic gradient descent and application to data-parallel distributed training of speech dnns. In *Interspeech 2014*.

[190] Shalev-Shwartz, S. and Zhang, T. (2013). Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599.

[191] Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

[192] Shamir, O. and Srebro, N. (2014). Distributed stochastic optimization and learning. In *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 850–857. IEEE.

[193] Shamir, O., Srebro, N., and Zhang, T. (2014). Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008.

[194] Shi, E., Chan, T. H., Rieffel, E., Chow, R., and Song, D. (2011). Privacy-preserving aggregation of time-series data. In *Proc. NDSS*, volume 2, pages 1–17. Citeseer.

[195] Shokri, R. and Shmatikov, V. (2015). Privacy-preserving deep learning. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1310–1321, New York, NY, USA. ACM.

[196] Shokri, R., Stronati, M., Song, C., and Shmatikov, V. (2017a). Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE.

[197] Shokri, R., Stronati, M., Song, C., and Shmatikov, V. (2017b). Membership inference attacks against machine learning models.

[198] Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A. S. (2017). Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434.

[199] Snodgrass, J. G. and Vanderwart, M. (1980). A standardized set of 260 pictures: norms for name agreement, image agreement, familiarity, and visual complexity. *Journal of experimental psychology: Human learning and memory*, 6(2):174.

[200] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[201] Stanley, K. O. and Miikkulainen, R. (2002). Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 2, pages 1757–1762. IEEE.

[202] Stich, S. U. (2018). Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*.

[203] Strom, N. (2015). Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*.

[204] Subramanyan, P., Sinha, R., Lebedev, I., Devadas, S., and Seshia, S. A. (2017). A formal foundation for secure remote execution of enclaves. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2435–2450.

[205] Sun, Y., Xue, B., Zhang, M., and Yen, G. G. (2018). Automatically designing cnn architectures using genetic algorithm for image classification. *arXiv preprint arXiv:1808.03818*.

[206] Suresh, A. T., Yu, F. X., Kumar, S., and McMahan, H. B. (2017). Distributed mean estimation with limited communication. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3329–3337. JMLR. org.

[207] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147.

[208] Szegedi, G., Kiss, P., and Horváth, T. (2019). Evolutionary federated learning on eeg-data. In *CEUR Workshop Proceedings (CEUR-WS.org), ISSN: 1613-0073.*, volume 2473, pages 71–78. ceur-ws.org.

[209] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[210] Thibaux, R. and Jordan, M. I. (2007). Hierarchical beta processes and the indian buffet process. In *Artificial Intelligence and Statistics*, pages 564–571.

[211] Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. (2016). Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618.

[212] Tsai, C.-Y., Saxe, A. M., and Cox, D. (2016). Tensor switching networks. In *Advances in Neural Information Processing Systems*, pages 2038–2046.

[213] Tsai, J.-T., Chou, J.-H., and Liu, T.-K. (2006). Tuning the structure and parameters of a neural network by using hybrid taguchi-genetic algorithm. *IEEE Transactions on Neural Networks*, 17(1):69–80.

[214] Tschantz, M. C., Sen, S., and Datta, A. (2020). Sok: Differential privacy as a causal property. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 354–371. IEEE.

[215] Vanhaesebrouck, P., Bellet, A., and Tommasi, M. (2017). Decentralized collaborative learning of personalized models over networks. In *Artificial Intelligence and Statistics*, pages 509–517. PMLR.

[216] Verbancsics, P. and Harguess, J. (2013). Generative neuroevolution for deep learning. *arXiv preprint arXiv:1312.5355*.

[217] Verbancsics, P. and Stanley, K. O. (2011). Constraining connectivity to encourage modularity in hyperneat. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1483–1490. ACM.

[218] Vidnerová, P. and Neruda, R. (2018). Asynchronous evolution of convolutional networks. In *ITAT*, pages 80–85.

[219] Wang, B. and Gong, N. Z. (2018). Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 36–52. IEEE.

[220] Wang, H., Kaplan, Z., Niu, D., and Li, B. (2020). Optimizing federated learning on non-iid data with reinforcement learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1698–1707. IEEE.

[221] Wang, J. and Joshi, G. (2018). Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms. *arXiv preprint arXiv:1808.07576*.

[222] Wang, K., Mathews, R., Kiddon, C., Eichner, H., Beaufays, F., and Ramage, D. (2019). Federated evaluation of on-device personalization. *arXiv preprint arXiv:1910.10252*.

[223] Wang, S. and Manning, C. (2013). Fast dropout training. In *international conference on machine learning*, pages 118–126. PMLR.

[224] Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C., He, T., and Chan, K. (2018). Adaptive federated learning in resource constrained edge computing systems. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*.

[225] Wei, E. and Ozdaglar, A. (2013). On the o (1= k) convergence of asynchronous distributed alternating direction method of multipliers. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 551–554. IEEE.

[226] Whitley, D., Starkweather, T., and Bogart, C. (1990). Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing*, 14(3):347–361.

[227] Woodworth, B., Wang, J., Smith, A., McMahan, B., and Srebro, N. (2018). Graph oracle models, lower bounds, and gaps for parallel stochastic optimization.

[228] Wright, S. and Nocedal, J. (1999). Numerical optimization. *Springer Science*, 35(67-68):7.

[229] Wu, X., Fredrikson, M., Jha, S., and Naughton, J. F. (2016). A methodology for formalizing model-inversion attacks. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 355–370. IEEE.

[230] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

[231] Xiao, L. and Zhang, T. (2014). A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075.

[232] Xie, L. and Yuille, A. (2017). Genetic cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1379–1388.

[233] Yan, M., Fletcher, C. W., and Torrellas, J. (2020). Cache telepathy: Leveraging shared resource attacks to learn {DNN} architectures. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 2003–2020.

[234] Yang, H. H., Liu, Z., Quek, T. Q., and Poor, H. V. (2019). Scheduling policies for federated learning in wireless networks. *IEEE Transactions on Communications*.

[235] Yang, K., Jiang, T., Shi, Y., and Ding, Z. (2020). Federated learning via over-the-air computation. *IEEE Transactions on Wireless Communications*, 19(3):2022–2035.

[236] Yao, A. C.-C. (1986). How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE.

[237] Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.

[238] Yeom, S., Giacomelli, I., Fredrikson, M., and Jha, S. (2018). Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 268–282. IEEE.

[239] Yu, H., Yang, S., and Zhu, S. (2019). Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:5693–5700.

[240] Yurochkin, M., Agarwal, M., Ghosh, S., Greenewald, K., Hoang, T. N., and Khazaeni, Y. (2019). Bayesian nonparametric federated learning of neural networks. *arXiv preprint arXiv:1905.12022*.

[241] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

[242] Zeng, Q., Du, Y., Leung, K. K., and Huang, K. (2019). Energy-efficient radio resource allocation for federated edge learning. *arXiv preprint arXiv:1907.06040*.

[243] Zhang, C., Bütepage, J., Kjellström, H., and Mandt, S. (2018a). Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026.

[244] Zhang, S., Choromanska, A. E., and LeCun, Y. (2015). Deep learning with elastic averaging sgd. In *Advances in neural information processing systems*, pages 685–693.

[245] Zhang, Y., Duchi, J. C., and Wainwright, M. J. (2013). Communication-efficient algorithms for statistical optimization. *The Journal of Machine Learning Research*, 14(1):3321–3363.

[246] Zhang, Y., Liang, P., and Wainwright, M. J. (2017). Convexified convolutional neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4044–4053. JMLR. org.

[247] Zhang, Y. and Lin, X. (2015). Disco: Distributed optimization for self-concordant empirical loss. In *International conference on machine learning*, pages 362–370.

[248] Zhang, Y., Xiang, T., Hospedales, T. M., and Lu, H. (2018b). Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4320–4328.

[249] Zhang, Y. and Yeung, D.-Y. (2012). A convex formulation for learning task relationships in multi-task learning. *arXiv preprint arXiv:1203.3536*.

[250] Zhao, B., Mopuri, K. R., and Bilen, H. (2020). idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*.

[251] Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., and Chandra, V. (2018). Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*.

[252] Zhou, F. and Cong, G. (2018). On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*.

[253] Zhu, G., Wang, Y., and Huang, K. (2018a). Broadband analog aggregation for low-latency federated edge learning (extended version). *arXiv preprint arXiv:1812.11494*.

[254] Zhu, G., Wang, Y., and Huang, K. (2018b). Low-latency broadband analog aggregation for federated edge learning. *arXiv preprint arXiv:1812.11494*.

[255] Zhu, L., Liu, Z., and Han, S. (2019). Deep leakage from gradients.

[256] Zinkevich, M., Weimer, M., Li, L., and Smola, A. J. (2010). Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603.

Appendix A

Background

A.1 Convex optimization

A.1.1 Convex functions

A $C \subseteq \mathbb{R}^n$ set is a **convex set** if for any $x, y \in C$

$$tx + (1-t)y \in C, \ \forall\, 0 \leq t \leq 1. \tag{A.1}$$

A **convex combination** of $x_1, \ldots, x_n \in \mathbb{R}^n$ is any linear combination:

$$\theta_1 x_1 + \ldots, \theta_k x_k, \ \text{ where } \sum_i \theta_i = 1 \tag{A.2}$$

A **convex hull** of C is all linear combination of elements in C.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is **convex function**, if $\mathrm{dom}(f) \subseteq \mathbb{R}^n$ is convex, and

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y), \text{ for } 0 \leq t \leq 1 \text{ and } \forall x, y \in \mathrm{dom}(f) \tag{A.3}$$

Visually this can be described as the line of the function lies below any line segment joining $f(x)$ and $f(y)$.

A **concave function** is a function, whose negative is convex.

A f is **proper convex function**, if it takes a finite value at least at one point $x \in C$, and lower bounded: $f(x) > -\infty,\ \forall x \in C$.

A.1.2 Subgradients and Gradient(steepest descent)

A vector $y \in \mathbb{R}^n$ is a **subgradient** of a convex function f at $x \in C$ if

$$f(z) \geq f(x) + \langle z - x, y \rangle,\ \forall z \in C. \tag{A.4}$$

That is equivalent to f having a supporting hyperplane with a slope y.
The set of subgradients of a convex function f at point $x \in \mathrm{dom}(f)$ is called **subdifferential** and denoted as $\partial f(x)$.
A function f is **differentiable** at point x if there is only a unique subgradient at x, and in this case $\partial f(x) = \nabla f(x)$ (that is it is equal to the gradient). A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if it has non-empty subdifferential set everywhere.
First order optimiality condition for a proper convex function f: $x^* \in \arg\min_{x \in C} f(x) \iff 0 \in \partial f(x^*)$.

Iterative Descent algorithms In iterative decent methods start form an initial point $\mathbf{x}_0$,and construct a sequence: $\{x_t\}$, were $f(\mathbf{x}_{t+1}) < f(\mathbf{x}_t)$.

$$\min_x f(\mathbf{x}) s.t. \mathbf{x} \in \mathbb{R}^n \tag{A.5}$$

A vector $\mathbf{d}$ is a **descent direction** at $\mathbf{x}$ if

$$f(\mathbf{x}; \mathbf{d}) = \underbrace{\lim_{\tau \searrow 0} \frac{f(\mathbf{x} + \tau \mathbf{d}) - f(\mathbf{x})}{\tau}}_{\text{directional derivative}} = \nabla f(\mathbf{x})^T \mathbf{d} < 0 \tag{A.6}$$

Using a descent direction at $\mathbf{x}_t$ and a $\eta_t > 0$ stepsize:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \eta_t \mathbf{d}_t \tag{A.7}$$

The most important example is **gradient descent (GD)** also called **steepest descent**, where

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \eta_t \nabla f(\mathbf{x}_t). \tag{A.8}$$

That is $\mathbf{d}_t = -\nabla f(\mathbf{x}_t)$.

A.1.3 Second order derivative - Curvature

For function $f : \mathbb{R}^d \to \mathbb{R}$ at a point $\mathbf{x} \in \mathbb{R}^n$, if all second partial derivatives exist and continuous over the dom(f), the local curvature can be described by the **Hessian** matrix H:

$$H = (\frac{\partial^2 f}{\partial x_i \partial x_j})_{i,j} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots \quad \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots \frac{\partial^2 f}{\partial x_d^2} \end{pmatrix}$$

If a function $f : \mathbb{R}^d \to \mathbb{R}$ is convex (that is $f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)), \lambda \in [0,1], \forall x_1, x_2$) then the Hessian $H = \nabla^2 f$ is always positive semidefinite: $x^T H x \geq 0, \forall x \in \mathbb{R}^d$

Curvature and GD Curvature in some way describes how different is a function from linear. When the regions of the parameter-space are very different, with a fixed learning rate the convergence of the GD becomes slow: in high curvature regions the small step size makes faster progress but the same time in low curvature regions it becomes really slow. On the other hand large learning rate leads to good progress in low curvature regions, but may end up in oscillation at high curvature (See Figure A.1).

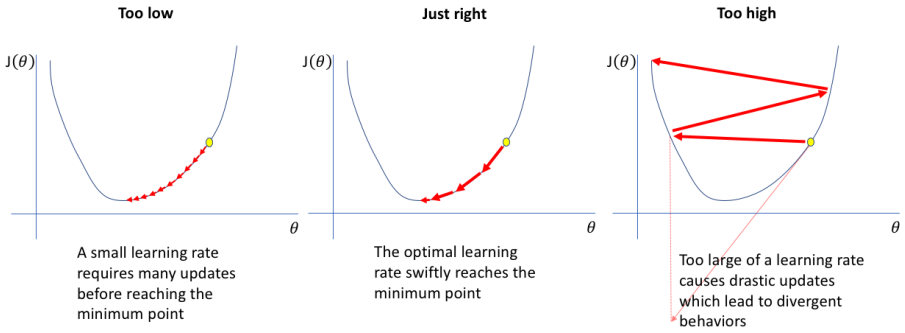


Figure A.1 Step size and learning rate in GD [**?**]

The oscillation in higher dimension also turns into a zigzagging behaviour, where we repeatedly overshoot the optimum in the update direction $\mathbf{d}$ ($= \nabla f(x)$ in gradient descent).

A.1.4 Momentum or Heavy Ball method

The idea of momentum [170] is to use a buffer to smooth the trajectory, that can be understood as adding inertia to a ball that moving down on an uneven terrain (Fig A.2). This inertia is includes all the past descent directions and influences the direction of the descent step. In a single update rule the momentum step can be :

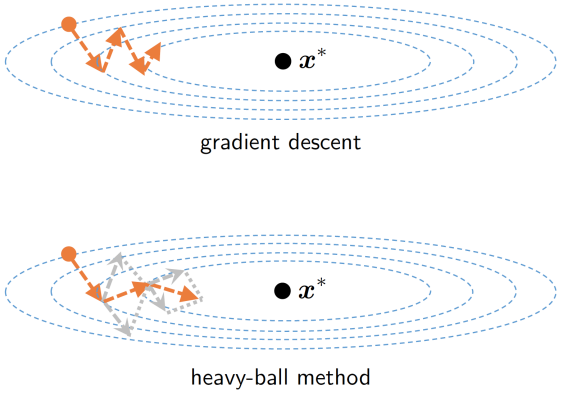


Figure A.2 Zigzagging behaviour of SGD and its mitigation by momentum [42]

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \overbrace{\eta_t \nabla f(\mathbf{x}_t) + \underbrace{\alpha_t(\mathbf{x}_t - \mathbf{x}_{t-1})}_{\text{momentum term}}}^{\mathbf{v}_{t+1}}, \tag{A.9}$$

or according to the more common formulation separating into two steps:

$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta_t \nabla f(\mathbf{x}_t) \tag{A.10}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{v}_{t+1} \tag{A.11}$$

Nesterov's Accelerated Gradient descent Nesterov's accelerated gradient [207][164] methods makes a simple change on momentum algorithm, that in practise has proven to help at a significant extent in a lot of cases. Instead of computing the gradient and then add momentum term to it, it takes to gradient in the position after adding the momentum(Figure A.3):

$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta_t \nabla f(\mathbf{x}_t + \mu \mathbf{v}_t) \tag{A.12}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{v}_{t+1} \tag{A.13}$$

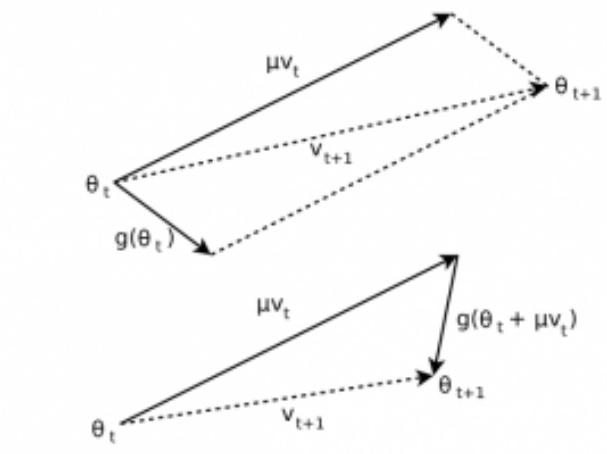


Figure A.3 Momentum (top) and Nesterov's accelerated gradient[207]

A.1.5 Stochastic Gradient Descent methods

In ML training our goal is to minimize the **risk**, or expected loss $f(y,m(x))$ of a model (that is a classification or regression function) $m(x)$:

$$R_{f,P}(m) = \mathbb{E}[f(y,m(x))] = \int_{X\times Y} f(y,m(x))dP(x,y) \tag{A.14}$$

The actual goal in training however, what can we hope to achieve is to find model m such that it minimizes

$$\tilde{m} = \arg\min_m \frac{1}{n}\sum_{i=1}^{n_{\text{Test}}} f(y_i,m(x_i)),\ (x_i,y_i) \in \mathcal{D}_{\text{Test}} \tag{A.15}$$

for the *test examples* (x_i,y_i) But since at training the test examples are unknown, what we do is **minimize empirical risk** for *training examples* (x_i,y_i) :

$$\arg\min_m \sum_{i=1}^{n_{\text{Train}}} f(y_i,m(x_i))\ (x_i,y_i) \in \mathcal{D}_{\text{Train}} \tag{A.16}$$

For A.16 the gradient descent method would look as follows:

$$x_{t+1} = x_t - \eta_t \sum_{i=1}^{n} f(x_t). \tag{A.17}$$

Since this can be very expensive,the idea of SDG is to pick an example x_{i_t} uniformly random, and execute update with that single gradient:

$$x_{t+1} = x_t - \eta_t f(x_{i_t}). \tag{A.18}$$

For a randomized picking of the index $\mathbb{E}[\nabla f(x_{i_t})] = \nabla f(x)$, thus the stochastic gradient i an unbiased estimate of the full gradient.

Minibatch Stochastic Gradient Descent (MBSGD) As a compromise between the costly full gradient descent and SGD that involves high variance we commonly use MBSGD, that in practice often improves convergence rate of SGD. In MBSGD before the

update the gradients of a random subset $I_t \subset 1,\ldots,m$ of the dataset are evaluated, and the mean applied on the parameters:

$$x_t = x_{t-1} - \frac{\eta_t}{b}\sum_{i\in I_t}\nabla f_i(x_{t-1}) \tag{A.19}$$

where b is the size of the minimatch.

A.1.6 Accelerated Stochastic Methods

Randomized Coordinate descent [165] in each iteration t picks a coordinate j of the parameter vector $\mathbf{w}$, and performs update with stepsize $\eta_t$on that single coordinate based on the partial derivative w.r.t $\mathbf{w}_{j_t}$:

$$w_{t+1} = w_t - \eta_{j_t}\nabla_{j_t} f(\mathbf{w}_t)\mathbf{e_{j_t}} \tag{A.20}$$

Generalized linear models where with certain sparsity evaluation of partial derivative $\nabla_j f(\mathbf{w})$ very efficiently, as in [178]. Further works for proximal setting, single processor parallelism [30] [179] and acceleration [139], all of these connected into a single algorithm [65].

A.2 Variance reduced algorithm

Variance reduced methods in first place aim at reducing the inherent bias and variance of the algorithms class of *stochastic mini batch gradient descent* :

$$\text{Bias}(g(x_t)) = \mathbb{E}[g(x^t)] - \nabla f(x^t) \tag{A.21}$$

$$\text{Var}(g(x)) = \mathbb{E}[g(x^t) - \mathbb{E}[g(x^t)]]^2 \leq \mathbb{E}[g(x^t)]^2 \tag{A.22}$$

Since the variance is large, we usually apply decaying η_t step size, that keeps the path closer to the ideal decent direction, on the other hand however it makes the convergence slow. As in [15] is pointed out, variance reduced algorithm as *SAG, SVRG* can actually perform better as learning algorithms than SGD in certain sittings.

Variance reduction Having an estimator X for a parameter θ, estimator X is unbiased if $\mathbb{E}[X] = \theta$. With a modified estimator $Z \stackrel{\text{def}}{=} X - Y$, with Y such that $\mathbb{E}[Y] \approx 0$, the bias of Z will be also close to 0:

$$\mathbb{E}[Z] = \mathbb{E}[X] - \mathbb{E}[Y] \approx \theta. \tag{A.23}$$

And

$$\text{Var}(X - Y) = \text{Var}(X) + \text{Var}(X) - 2\text{Cov}(X, Y), \tag{A.24}$$

Thus through creating an estimator Y for which $\mathbb{E}[Y] \approx 0$, and that is highly correlated with X, the variance can be reduced significantly.

This general template has been followed at the construction of SAG, SAGA, SVRG and SDCA algorithms.

Stochastic Average Gradient (SAG) [182][188] has been developed based on the idea of incremental gradients [23]. The main goal in SAG is reducing the variance the stochastic gradient algorithm by in a way hybridizing full gradient descent and stochastic gradient descent. SAG essentially maintains a table for gradients $g_t^i = \nabla f_i(\mathbf{w}_t)$ per data points x_i, $(i = 0, \ldots, n)$, that is initialized by the staring point w_0. In each iteration then the gradient is updated at a random point x_i, while the other gradient values will stay the same,

and the full updated will be calculated by averaging over all g_i:

$$g_{i_t} = \nabla f_i(\mathbf{w}_t, x_i), \qquad \text{if } i = i_k \tag{A.25}$$

$$g_{i_t} = g_{i_{t-1}}, \qquad \text{if } i \neq i_k \tag{A.26}$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta_t \frac{1}{n} \sum_{i=1}^{n} g_t^{(i)} \tag{A.27}$$

That can be implemented slightly more efficiently:

$$\mathbf{w}_t = \mathbf{w}_{t-0} - \eta_t \underbrace{\left(\frac{g_t^{i_t}}{n} - \frac{g_{t-1}^{i_t}}{n} + \overbrace{\frac{1}{n} \sum_{i=1}^{n} g_{t-1}^{(i)}}^{\text{old table average}} \right)}_{\text{new table average}}. \tag{A.28}$$

Thus the stochastic gradient in SAG:

$$\underbrace{g_t^{i_t}}_{X} - \left(\underbrace{g_{t-1}^{i_t} - \sum_{i=1}^{n} g_{t-1}^{(i)}}_{Y} \right), \tag{A.29}$$

where $\mathbb{E}[X] = \nabla f(\mathbf{w_t})$. However $\mathbb{E}[Y] \neq 0$, thus it is a biased estimator. On the other hand Y and X are correlated, that is $X - Y \to 0$ as $k \to \infty$. Since $\mathbf{w}_{t-1}$ and $\mathbf{w}_t$ converge to the optimum $\mathbf{w}^*$, thus $g_t^{i_t} - g_{t-1}^{i_t} \to 0$, and $\sum_{i=1}^{n} g_{t-1}^{(i)} \to \nabla f(\mathbf{w}^*) = 0$. Thus the l_2 norm of the overall estimator, and with that its variance decays to zero as well.

SAGA An follow up amendment for SAG is SAGA [52], that replaces the biased gradient estimator of SAG with an unbiased one:

$$\underbrace{g_t^{i_t}}_{X} - \left(\underbrace{g_{t-1}^{i_t} - \frac{1}{n} \sum_{i=1}^{n} g_{t-1}^{(i)}}_{Y} \right), \tag{A.30}$$

where $\mathbb{E}[Y] \neq 0$, so this is unbiased. The variance moreover similarly to SAG goes to zero as well.

Stochastic Variance Reduced Gradient (SVRG) SVRG [117] working in a similar way to SAG and SAGA, with the difference that instead of maintaining a table of all gradients it only stores an average of them, that is the full gradient $\tilde{g} = \nabla f(\tilde{\mathbf{w}}_r)$ of the loss f at an initial point $\tilde{\mathbf{w}}_r$. The stochastic update then is computed similarly to SAGA for random points $\mathbf{x}_{i_t}$ and:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta_t \left(\nabla f_{i_t}(\mathbf{w}_{t-1}) - \nabla f_{i_t}(\tilde{\mathbf{w}}_r) + \tilde{g}\right) \tag{A.31}$$

The algorithm runs in in two nested loops, in the outer loop the new estimation of the optimum $\tilde{w}_r$ will be updated from the result of multiple execution of the inner loop $\tilde{w}_{r+1} = \mathbf{w}_t$, then the full gradient will be recomputed $\tilde{g} = \nabla f(\tilde{\mathbf{w}}_r)$, and starts the inner loop again.

A.2.1 Dual methods

Dual spaces

In linear algebra a **functional** is a linear mapping from a vector space $\mathcal{V}$ into its field of scalars $\mathcal{V} \to \mathbb{F}$ (a field $\mathbb{F}$ for example $\mathbb{C}, \mathbb{R}$)
The **dual space** $\mathcal{V}^*$ of a vector space $\mathcal{V}$ is the space of all continuous linear functionals on that space $\mathcal{V}^* = \mathcal{L}(\mathcal{V}, \mathbb{F})$. The elements of the dual are the linear functionals.

Dual optimization

Dual optimization is at first place used for constrained optimization, that is when we are looking for optimal values of a function f within a restricted domain space.

The Lagrange dual problem can be used when we cannot solve the primal as $\frac{\partial f(x)}{\partial x} = 0$, because we have to take into account the feasible region of the constraints.

$$\min_x f(x) \tag{A.32}$$

$$\text{subject to } h_i(x) \leq 0, i = 1, \dots m \tag{A.33}$$

$$l_j(x) = 0, j = 1, \dots, r \tag{A.34}$$

$$\tag{A.35}$$

In **Lagrange multiplier method** we add constraints as a perturbation the original problem(inequalities $h(x) \leq 0$ with weights $\lambda \geq 0$, and equalities $l(x) = 0$ with weights $\nu \geq 0$)):

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^T h(x) + \nu^T l(x) \tag{A.36}$$

Since equalities $l(x)$ can be transformed to inequalities by adding $l(x) \geq 0$ and $-l(x) \geq 0$, it is common to omit this part and use only $h(x)$ to denote the constraint set.

Taking the minimum of this Lagrangian of the problem (Equation A.36) w.r.t. the original variables x gives us the **Lagrangian dual function** $g(\lambda)$ in terms of the **Lagrange multipliers** λ:

$$g(\lambda) = \inf_{x \in \mathcal{D}} \mathcal{L}(x, \lambda) \tag{A.37}$$

where $\mathcal{D} = \text{dom}(f)$ is the domain of f without any constraints.

Then, the optimum of the dual g^* will be :

$$g^* = \max_{\lambda} g(\lambda), \text{ s.t.} \lambda \geq 0 \tag{A.38}$$

Weak duality The g^* is a lower bound for f, since we decrease f in any case with the constraints terms(adding λhs having $h \leq 0$ and $\lambda > 0$. The dual $g()$ is at the minimum in

f so in any case the dual problem gives a lower bound. The task therefore is to give *best lower bound* that is equivalent to maximize g.

Strong duality

In some cases $f^* = g^*$, this is called *strong duality*.

Slater constraint qualification : Strong duality holds for a convex problem in a form:

$$\min f(x) \tag{A.39}$$

$$\text{s.t.} h_i(x) \leq 0, i = 1, \ldots, m \tag{A.40}$$

$$Ax = b, \tag{A.41}$$

if it is *strictly feasible*, that is there is at least one point in the domain for f_0 that strictly satisfies the conditions. Formally:

$$\exists x \in \text{int}\mathcal{D} : h_i(x) < 0, i = 1, \ldots, m \tag{A.42}$$

$$Ax = b \tag{A.43}$$

Complimentary slackness If strong duality holds x^* optimal primal and (λ^*, ν^*) optimal dual

$$f_0(x^*) = g(\lambda^*, \nu^*) = \inf_x \left(f_0(x) + \sum_{i=1}^{m} \lambda_i^* f_i(x) + \sum_{i=p}^{m} \nu^* h_i(x) \right) \tag{A.44}$$

$$\leq f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* f_i(x^*) + \sum_{i=p}^{m} \nu^* h_i(x^*) \tag{A.45}$$

$$\leq f_0(x^*) \tag{A.46}$$

thus $\leq$ holds as $=$,

- and x^* minimizes $L(y, \lambda^*, \nu^*)$,
- moreover $\lambda_i^* f_i(x^*) = 0$ for $i = 1, 2, \ldots$, that is:

- $\lambda_i^* > 0 \Rightarrow f_i(x^*) = 0$
- $f_i(x^*) < 0 \Rightarrow \lambda_i^* = 0$

(kind of a stopping criterion)

Karush-Kuhn-Tucker(KKT) conditions If for a problem with differentiable f_i, h_i strong duality holds, and $x,\ \lambda,\ \nu$ are optimal, then they also satisfies **KKT conditions**, and these conditons are sufficient:

- *primal constraints:*

$$h_i(x) \leq 0, i = 1, \ldots, m, \tag{A.47}$$

$$l_j(x) = 0, j = 1, \ldots, p \tag{A.48}$$

- dual constraints $\lambda_i \geq 0,\ i = 1, \ldots, m$
- complimentary slackness: $\lambda_i h_i(x) = 0,\ i = 1, \ldots, m$
- and gradient of the Lagrangian vanishes:

$$0 \in \partial_x \left(f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{j=1}^{p} \nu_j l_j(x) \right) \tag{A.49}$$

that is, if f is differentiable: (A.50)

$$\nabla_x f(x) + \sum_{i=1}^{m} \lambda_i \nabla_x h_i(x) + \sum_{j=1}^{p} \nu_j \nabla_x l_j(x) = 0 \tag{A.51}$$

if f strongly convex dual is smooth.

Lagrangian in the dual space This f^* is already defined in the dual space, that is the space of all functional over the vector-space V: $f^* : V \to \mathbb{F}$, where $\mathbb{F}$ is afield, as $\mathbb{C}$ or $\mathbb{R}$. And this is what conjugates do.

Conjugate

In convex analysis the basic idea of using duality is to think of a convex function, or the epigraph of the function as a convex set, that is an intersection of the *supporting hyperplanes*(Figure A.4).

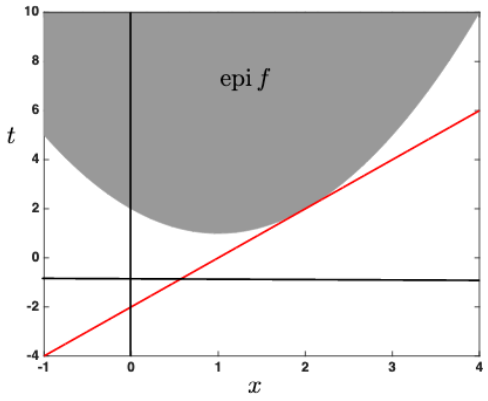


Figure A.4 The epigraph of function f, with a supporting hyperplane (From: [180]).

These supporting hyperplanes are described by affine functions, that are majorized by f:

$$\forall x, l(x) = \langle \alpha, x \rangle - a, \tag{A.52}$$

$$\text{with } l(x) \leq f(x) \tag{A.53}$$

Fenchel conjugate To recover these supporting hyperplanes, we can use f^*, the so-called Fenchel-conjugate of f. This conjugate is a function, that for each slope α assigns a (the y-interception of the hyperplane) providing us all these hyper-plane equation.

That is for a given slope α the "best" choice for a:

$$f(x) \geq \langle \alpha, x \rangle - a, \forall x \in V \tag{A.54}$$

$$\iff a \geq \langle \alpha, x \rangle - f(x), \forall x \in V \tag{A.55}$$

$$\iff a \geq \sup_{x \in V} \langle \alpha, x \rangle - f(x) \tag{A.56}$$

And the best minorant hyperplane with slope α:

$$f^*(\alpha) = \sup_{x \in V} \langle \alpha, x \rangle - f(x), \tag{A.57}$$

if this suprenum is finite. If $f^*(\alpha) = \infty$ that means that there is no minorant hyperplane for f with the given slope. f^* is called the *(Fenchel-)conjugate* or *Legendre-transform* of f. (See Figure A.5 for a geometrical intuition.)

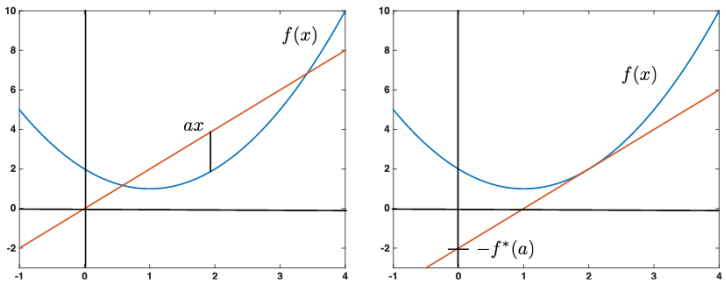


Figure A.5 **Left:** A hyperplane with slope a.**Right:**Fenchel conjugate $f^*(\alpha)$ of a convex function $f(x)$: the $y-$intercept for the supporting hyperplane with slope a(From: [180]).

Useful properties of conjugate For a conjugate of $f : \mathbb{R} \to \mathbb{R}$ the conjugate

$$f^*(y) = \max_x y^T x - f(x) \tag{A.58}$$

- The conjugate is always convex.

- A useful formulation multiplying by -1:

$$-f^*(y) = \min_x f(x) - y^T x \tag{A.59}$$

- if f is closed convex function, then $f^{**} = f$ and

$$x \in \partial f^*(y) \iff y \in \partial f(x) \iff x \in \arg\min_z f(z) - y^T z, \tag{A.60}$$

since

$$x \in \arg\min_z f(z) - y^T z \iff 0 \in \partial f(x) - y \iff y \in \partial f(x) \iff y \in \partial f^*(y) \tag{A.61}$$

(other direction follows from $f^{**} = f$)

- If f is strictly convex, then f^* is differentiable and $\nabla f^*(y) = \arg\min_z f(z) - y^T z$.

These properties are pretty useful since it is possible optimize the dual(conjugate), without the direct computation of the gradient, that is very helpful if the conjugate cannot be obtained in a closed form.

Compute the Fenchel conjugate The Fenchel conjugate

$$f^*(\alpha) = \sup_{x \in \mathcal{C}} [\langle x, \alpha \rangle - f(x)]. \tag{A.62}$$

can be expressed fairly simply (at least for locally convex functions) as function of α. Maximizing term $x^T\alpha - f(x)$ w.r.t x, x can be expressed by the dual variable $alpha$.

$$f^*(\alpha) = \max_x \{x^T\alpha - f(x)\} \tag{A.63}$$

For example:

$$f(x) = x^2 - 2x + 2 \tag{A.64}$$

$$f^*(\alpha) = \sup_{x\in\mathcal{R}}(\alpha x - f(x)) = \sup_{x\in\mathcal{R}}(\underbrace{\alpha x - x^2 + 2x - 2}_{:=g(x,\alpha)}) \tag{A.65}$$

$$\tag{A.66}$$

since $g(x,\alpha)$ is concave (in x), its minimum can be given by setting $\nabla_x g(x,\alpha) = 0$, then express x with α

$$\nabla_x g(x,\alpha) = \alpha - (2x-2) = \alpha - 2x + 2 = 0 \tag{A.67}$$

$$x = \frac{\alpha}{2} - 1 \tag{A.68}$$

$$\tag{A.69}$$

plugging this back into $f^*(\alpha)$:

$$f^*(\alpha) = \alpha(\frac{\alpha}{2} - 1) - (\frac{\alpha}{2} - 1)^2 + 2(\frac{\alpha}{2} - 1) - 2 \tag{A.70}$$

$$= \frac{\alpha^2}{2} - \alpha - \frac{\alpha^2}{4} + \alpha - 1 + \alpha + 2 - 2 = \frac{\alpha^2}{4} + \alpha - 1 \tag{A.71}$$

$$\tag{A.72}$$

Lagrangian and Fenchel dual For convex optimization with affine equality constraints:

$$\min_x f(x), \text{ s.t. } Ax = b \tag{A.73}$$

the Lagrangian is given by

$$L(x,u) = f(x) + u^T(Ax + b), \tag{A.74}$$

and the dual is, by minimizing the Lagrangian in x:

$$g(u) = \min_x L(x,u) = \min_x f(x) + u^T(Ax-b) \tag{A.75}$$

$$= \underbrace{\min_x f(x) - (-A^T u)^T x}_{\text{def of } f^* \text{ with slope } -A^T b} - u^T b \tag{A.76}$$

$$= -f^*(-A^T u) - u^T b \tag{A.77}$$

using the definition of conjugate from which the dual problem to solve is :

$$\max_u -f^*(-A^T u) - u^T b. \tag{A.78}$$

whose subgradient(subdifferential) is thus simply

$$\partial g(u) = A\partial f^*(-A^T u) - b = Ax - b, \tag{A.79}$$

with $x \in \arg\min_z f(z) + u^T Az$.

Fenchel Dual of Regularized convex linear model

$$\arg\min_{w\in\mathbb{R}^d} f(Xw) + r(w)) \tag{A.80}$$

introducing equality constraints: (A.81)

$$\arg\min_{v=Xw} f(v) + r(w) \tag{A.82}$$

, where $f : \mathbb{R}^d \to \mathbb{R}$, is a convex loss functions. The dual of Equation A.82 will take a special form that makes optimization easier:

$$L(v,w,z) = f(v) + r(w) + z^T(Xw - v) \qquad \text{Lagrangian} \tag{A.83}$$

$$D(z) = \inf_{v,w}\{f(v) + r(w) + z^T(Xw - v)\} \qquad \text{Lagrangian dual} \tag{A.84}$$

$$\text{inf w.r.t. } v: \tag{A.85}$$

$$\inf_v\{f(v) - z^T v\} = -\sup_v\{v^T z - f(v)\} = -f^*(z) \tag{A.86}$$

$$\text{inf w.r.t. } w: \tag{A.87}$$

$$\inf_w\{r(w) + z^T Xw\} = -\sup_w\{-z^T Xw - r(w)\} = -r^*(X^T z) \tag{A.88}$$

$$\tag{A.89}$$

That gives the dual objective:

$$\arg\max_{z\in\mathbb{R}^n} D(z), \text{where} \tag{A.90}$$

$$D(\mathbf{z}) = f_i^*(-z) - r^*(X^T\alpha), \tag{A.91}$$

A.2.2 Dual first order methods

Dual gradient ascent Using the formulation for the subgradient $\partial g(u) = Ax - b$, and KKT stationary condition gives the possibility to develop a subgradient method to optimize the dual variable u (maximizing the dual objective in u). This method is applicable for the case of strictly convex f, where f^* is differentiable and thus $\partial f^*(x) = \nabla f^*(x)$, based on which we can use *Dual gradient ascent* method, without the need to compute an expression explicitly for the dual function or for the conjugate:

1. start with an initial guess u_0

2. repeat for $t = 1, 2, \ldots$

$$x_k = \arg\min_z f(z) + (u_{k-1})^T Az \qquad \text{this is KKT stationary condition} \tag{A.92}$$

$$u_k = u_{k-1} + \eta_k(Ax_k - b) \qquad \text{Since } (Ax_k - b) = \nabla g(u_{k-1}) \tag{A.93}$$

Stochastic Dual Coordinate Ascent (SDCA): SDCA[190] has been developed for solving generic ridge regularized optimization problem for $x_1, \ldots x_n$ data points in $x_i \in \mathbb{R}^d$

$$\arg\min_{\mathbf{w}\in\mathbb{R}^d} F(w) = \left[\frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w}^T\mathbf{x}_i) + \frac{\lambda}{2}||\mathbf{w}||^2\right] \tag{A.94}$$

$$(\text{or } \arg\min_{w\in\mathbb{R}^d} f(Xw) + r(w))) \tag{A.95}$$

$$(\text{introducing constraints: } \arg\min_{v=Xw} f(v) + r(w))), \tag{A.96}$$

where $f_i : \mathbb{R} \to \mathbb{R}, i = 1, \ldots, n$ are scalar convex loss functions. That is we maximizing the negative convex conjugates of the original problem.Taking $\mathbf{w}^*$ for the optimum of (A.94) a $\mathbf{w}$ solution is defined to be ε_F-suboptimal, if $F(\mathbf{w}) - F(\mathbf{w}^*) < \varepsilon_F$. For SVM SGD sjown to reach an ε_F-suboptimal solution in time $O(1/\lambda\varepsilon_F)\log(1/\lambda\varepsilon_F))$ for given λ stepsize parameter. according the arguments of authors of [190], solving a problem with the most common SGD has a number of drawbacks:

- No clear stopping criterion
- it tends to be aggressive at the beginning
- reaches a moderate accuracy quite fast but convergence then becomes rather slow around the optima.

To address these issues, at least for the convex case it is possible to use Dual Coordinate Ascent(DCA) for the dual of the problem (A.94):

$$\arg\max_{\alpha\in\mathbb{R}^n} D(\alpha), \text{where} \tag{A.97}$$

$$D(\alpha) = \left[\frac{1}{n}\sum_{i=1}^{n} -f_i^*(-\alpha_i) - \frac{\lambda}{2}\underbrace{\left\|\frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i x_i\right\|^2}_{\text{from A.91}}\right], \tag{A.98}$$

$$\text{Or more generally:} \tag{A.99}$$

$$D(\alpha) = f_i^*(-\alpha) - r^*(X^T\alpha), \tag{A.100}$$

In (A.98) each α_i is associated with a single training example x_i, and in each iteration of DCA a dual objective will be optimized with respect to single coordinate, that is using a single training example.:

$$\arg\max_{\alpha_i}\left[-f_i^*(-\alpha_i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda n}\alpha_i x_i\right\|^2\right] \tag{A.101}$$

Then defining

$$w(\alpha) = \frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i \mathbf{x}_i, \tag{A.102}$$

$$\text{for strong duality } F(w^*) = D(\alpha^*) \text{ with} \tag{A.103}$$

$$w^* = \frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i^* x^{(i)}\ ,\ \alpha_i^* = -f'(w^{*T}x^{(i)}), \tag{A.104}$$

thus $w(\alpha^*) = \mathbf{w}^*$ with α^* is defined as the optimal solution of (A.98). Since we know that $F(\mathbf{w}) \geq D(\alpha)\forall\mathbf{w}\in\mathbb{R}^d \text{and}\alpha\in\mathbf{R}^n$, the duality gap $F(w(\alpha) - D(\alpha)$ is an upper bound for the primal sub-optimality $F(w(\alpha)) - F(\mathbf{w}^*)$. So if we want to solve for ε accuracy we just need to check whether this gap is smaller than ε.

Since in the dual formulation each coordinate of the dual variable $\alpha\in\mathbb{R}^n$ is associated with a single training example x, in DCA optimization we need to optimize $D(\alpha)$ w.r.t. a

single coordinate in each iteration, while the leftovers can be kept constant. SDCA choose the coordinate to update uniformly at random:

1. Let $w_0 = w(\alpha_0)$
2. repeat for $t = 1, 2, \ldots$

$$\Delta\alpha^i = \max_{\delta\alpha} -f_i^*(-(\alpha_{t-1}^i + \delta\alpha)) - \frac{\lambda n}{2}||w_{t-1} + \frac{\delta\alpha x_i}{\lambda n}||^2 \tag{A.105}$$

$$\alpha_t = \alpha_{t-1} + \delta\alpha^i \tag{A.106}$$

$$w_t = w_{t-1} + \frac{1}{\lambda n}\Delta\alpha^i x^i \tag{A.107}$$

A.2.3 Alternating Direction Method of Multipliers (ADMM)

Dual decomposition of the Lagrangian Elements of the primal variable vector **x** can be decomposed into B blocks: $x = [x_1, \ldots, x_B] \in \mathbb{R}^n$, and thus the primal problem with $x_i \in \mathbb{R}^{n_i}$ can be transformed into the following form:

$$\min_x \sum_{i=1}^{B} f_i(x_i) \text{ s.t.} Ax = b. \tag{A.108}$$

Naturally the constraints cannot be decomposed similar way, but we can partition coefficient matrix A accordingly: $A = [A_1, \ldots, A_B]$, where $A_i \in \mathbb{R}^{m \times n_i}$,

$$x \in \arg\min_z f(z) + u^T Az = \arg\min_z \sum_{i=1}^{B} (f_i(z_i) + u^T A_i z_i) \tag{A.109}$$

$$\iff x_i \in \arg\min_{z_i} f_i(z_i) + u^T (A_i z_i) \tag{A.110}$$

Thus actually the optimization objective also decomposed according to the blocks. Using this the *dual decomposition algorithm* works in the following way:

For $k = 1, 2, \ldots$

1. $x_i^{(k)} \in \arg\min_{z_i} f_i(z_i) + (u^{(k-1)})^T (A_i z_i),\ i = 1, 2, \ldots, B$
2. $u_i^{(k)} = u^{k-1} + \eta_k \left(\sum_{i=1}^{B} A_i x_i^{(k)} - b \right)$

A big advantage of dual decomposition is that it allows parallelized updates of each blocks, by (1) first broadcasting u to each process, that optimizes x_i, and then (2) collecting the local optimization results update u.

Augmented Lagrangian A disadvantage of Dual Gradient Ascent method is that it requires strong convexity of f to ensure convergence. Augmented Lagrangian method (or *method of multipliers*) has better convergence guarantees by transforming the primal problem:

$$\min_x f(x) + \frac{\rho}{2}\|Ax-b\|_2^2 \text{ s.t. } Ax = b. \tag{A.111}$$

This is the same problem, since it only means adding $0 = \|Ax-b\|$ in the feasible region. For $\rho > 0$ and a full column-rank A matrix this objective is fully convex that ensures convergence of the dual gradient ascent method.:

For $k = 1, 2 \cdots$:

1. $x^{(k)} = \arg\min_z f(z) + (u^{(k-1)})^T Az + \frac{\rho}{2}\|Az-b\|_2^2$
2. $u^{(k)} = u^{(k-1)} + \rho(Ax^{(k)} - b)$

with a stepsize $\eta = \rho$. Since choosing this value results in the stationarity condition in the primal problem, assuming that $Ax^{(k)} - b \to 0$ as $k \to \infty$:

$$x^{(k)} = \arg\min_z f(z) + (u^{(k-1)})^T Az + \frac{\rho}{2}\|Az-b\|_2^2 \tag{A.112}$$

$$\iff 0 \in \partial f(x^{(k)}) + A^T(u^{(k-1)+\rho(Ax^{(k)}-b)}) = \partial f(x^{(k)}) + A^T u^{(k)} \tag{A.113}$$

However, while the *augmented Lagrangian method* gives better convergence properties thank to strong convexity, it excludes the property of decomposability.

Alternating Direction Method of Multipliers (ADMM) ADMM method has been designed to combine the advantageous properties of the *dual decomposition* and *augmented*

Lagrangian method. For that the first task is to bring the problem in the form (if it is possible):

$$\min_{x,z} f(x) + g(z) \text{ s.t.} Ax + Bz = c \tag{A.114}$$

Then we can augment the the objective:

$$\min_{x,z} f(x) + g(z) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2 \text{ s.t.} Ax + Bz = c \text{ s.t.} Ax + Bz = c \tag{A.115}$$

for some $\rho > 0$. then the augmented Lagrangian:

$$L_\rho(x,z,u) = f(x) + g(z) + \tag{A.116}$$

$$+u^T(Ax + Bz - c)\frac{\rho}{2}\|Ax + Bz - c\|_2^2 \tag{A.117}$$

The *augmented Lagrangian method* would jointly minimize this over x and z:

$$\left(x^{(k)}, z^{(k)}\right) = \arg\min_{x,z} L_\rho(x,z,u^{(k-1)}) \tag{A.118}$$

ADMM instead splits the minimization into two steps: first over x, then z (or in reversed order) using the updated value of the previous step:

For k =1,2,...:

- $x^{(k)} = \arg\min_x L_\rho(x, z^{(k-1)}, u^{(k-1)}$
- $z^{(k)} = \arg\min_x L_\rho(x^{(k)}, z, u^{(k-1)})$
- $u^{(k)} = u^{(k)-1} + \rho(Ax^{(k)} + Bz^{(k)} - c)$

The update in this algorithm does not depend on gradients anymore, since the minimization happened before separately.

A.2.4 Some second order methods

Newton-type methods

Newton method is originally designed for zero finding,by iteratively progressing to the point where the value of a function f is 0.:

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)} \tag{A.119}$$

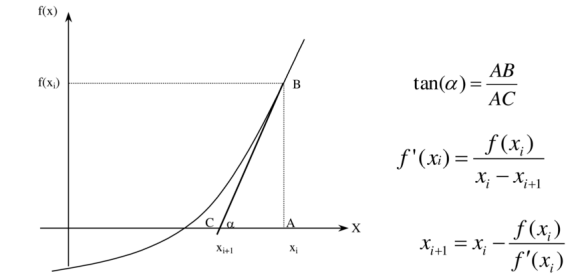


Figure A.6 Newton method for finding 0 [119]

For optimization it can be simply applied substituting f with its derivative, yielding a *second order method* :

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)} \tag{A.120}$$

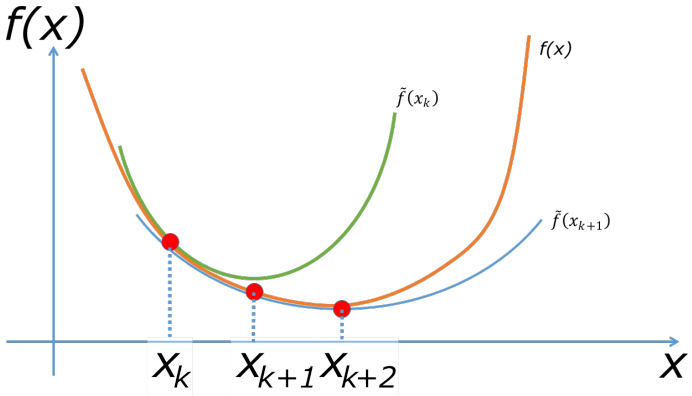


Figure A.7 Second Order Newton method [1]

Newton type methods yield asymptotic super-linear convergence using a better, quadratic approximation of the loss. A function that is locally convex in a neighbourhood of a can be approximated by second order Taylor polinom $q(x)$. Based on this, for an x being close enough to to a we can make the following approximation:

$$f(x) \approx f(a) + \overbrace{g^T}^{=\nabla f(a)^T}(x-a) + \frac{1}{2}\underbrace{(x-a)^T \overbrace{H}^{=\nabla^2 f(a)} (x-a)}_{x^THx-2a^THx+a^THa} \tag{A.121}$$

$$\text{with } b = x = g - Ha, \text{ and } c \text{ stands for the leftover terms} \tag{A.122}$$

$$f(x) \approx q(x) = \frac{1}{2}x^THx + b^Tx + c. \tag{A.123}$$

The loss f(Eq. A.123) can be exactly optimized by setting x to be the minimum of $q(x)$ (assuming that that the Hessian is invertable):

$$0 = \nabla q(x) = Hx + b \Rightarrow x = -H^{-1}b = -H^{-1}g + a \tag{A.124}$$

$$\tag{A.125}$$

To see that it is also a minimum if the second derivative is positive semidefinite:

$$\nabla^2 q = H \tag{A.126}$$

Thus the second derivative of q is also a second derivative of f. As long as f is a convex function (at least around a), then it is positive semidefinite, so it is a minimum.

So the algorithm:

- initialize $x_0 \in \mathbb{R}^n$
- iterate:

  $x_{t+1} = x_t - H^{-1}g$, where $g = \nabla f(x_t)$, and $H = \nabla^2 f(x_t)$

Damped newton method The so-called Damped newton method instead of jumping to the minimum of the approximation as above, uses an η learning rate:

$$x_{t+1} = x_t - \eta \nabla^2 [f(x_t)]^{-1} \nabla f(x_t), \tag{A.127}$$

$$\text{or} \tag{A.128}$$

$$x_{t+1} = x_t - \eta y, \tag{A.129}$$

Line search There are two basic iterative approaches in optimization to find a local minimum x^* of an objective function $f : \mathbb{R}^n \to \mathbb{R}$:

- line search
- trust regions

Line search is about finding a *descent direction*, along which f can be reduced, and then computes a step size to be taken in that direction. The descent direction can be computed by various methods as GD and quasi-newton methods. Step size can be determined exactly or inexactly.

make an initial guess x_0, $k = 0$ Repeat

1. compute a descent direction d_k
2. choose α_k to more or less minimize in $\alpha \in \mathbb{R}_+$ $h(\alpha) = f(x_k + \alpha d_k)$
3. update $x_{k+1} = x_k + \alpha d_k$

until $||\nabla f(x_{k+1})||$ <threshold

Exact line search Finding α_k stepsize can be given exactly: by solving $h'(\alpha) = 0$(as in conjugate gradient method) or loosely by asking a sufficient decrease in f.

So a way to find this is to calculate the first and second derivative and apply newton method:

$$h'(\alpha) = \nabla f(x_k + \alpha d_k) d_k \text{ directional derivative} \tag{A.130}$$

$$h''(\alpha) = \nabla^2 f(x_k + \alpha d_k) d_k \text{ curvature along } d_k \tag{A.131}$$

applying newton for reach iteration a takes a lot of time to calculate these derivatives. Even if it is a unidimensional problem ∇f and $\nabla^2 f$ are not.

For example So quadratic approximation a method that only uses h, and not h' or h''.

Wolfe condition In inexact line search Wolfe-conditions provide an efficient way to compute acceptable α, that reduces f sufficiently.

Wolfe conditions can be used as requirements for an α for being a good guess:

1. *Armijo rule*:

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k d_k^T \nabla f(x_k) \tag{A.132}$$

2. curvature condition

$$-d_k^T \nabla f(x_k + \alpha_k d_k) \leq -c_2 d_k^T \nabla f(x_k) \tag{A.133}$$

where $0 < c_1 < c_2 < 1$,c_1 is usually very small, and c_2 way bigger (Nocedal and Wright[228]: $c_1 = 10^{-4}, c_2 = 0.1$)

(We can ensure d_k to be a descent direction if $d_k^T \nabla f(x_k) < 0$, as in the case of gradient descent $d_k = -\nabla f(x_k)$)

Preconditioned steepest descent The essence *descent methods* is generating an algorithm that fits the optimization problem:

1. find the direction descent direction
2. how far in the direction - step size

The most natural choice for the descent direction is the gradient as it is in GD, however as discussed before it tends have s zigzag behaviour, depending on local curvature of the function.

Ideally for each direction we would like to go to the *Cauchy point*, that is the minimum along the descent direction.

If the curvature of a function is very different in specific directions, that leads to slow convergence of descent methods because it increases zigzaging. By Multiplying the variables by a preconditioner we would like to achieve the Curvature to be about the same around a single step

Taking a matrix H_t, that is symmetric and positive definite (this is the only thing required) we can calculate *Cholesky factorization* $H_k = L_k L_k^T$, where L_k is a lower rectangular matrix, that can be seen as square root of H. This matrix can be used to change the variables:

$$x' = L_k^T x. \tag{A.134}$$

We can then apply the steepest descent iteration in the new variables:

$$x'_{k+1} = x'_k - \alpha_t \nabla \tilde{f}(x'_k), \tag{A.135}$$

where $\tilde{f} = f(x')$, the function in the new variables.

With the correspondences

$$x = L_k^{-T} x', \text{ and } \tilde{f}(x'_k) = f(L_k^{-T} x'), \tag{A.136}$$

we can calculate $\nabla \tilde{f}(x'_k)$:

$$\nabla \tilde{f}(x'_k) = L_k^{-1} \nabla f(L_k^{-T} x') \tag{A.137}$$

Thus steepest descent iteration can be calculated the following way:

$$x'_{k+1} = x'_k - \alpha_k L_k^{-1} \nabla f(L_k^{-T} x'). \tag{A.138}$$

Then we can go back to the original variables applying A.136:

$$L_k^T x_{k+1} = L_k^T x_k - \alpha_k L_k^{-1} \nabla f(x_k) \tag{A.139}$$

Then multiplying everything with L_k^{-T}:

$$\underbrace{L_k^{-T}L_k^T}_{I}x_{k+1} = \underbrace{L_k^{-T}L_k^T}_{i}x_k - \alpha_k \underbrace{L_k^{-T}L_k^{-1}}_{H_k^{-1}}\nabla f(x_k) \tag{A.140}$$

Thus what we obtain, is the preconditioned steepest descent in the original variables:

$$x_{k+1} = x_k - \alpha_k H_k^{-1}\nabla f(x_k). \tag{A.141}$$

To prove that $-\alpha_k H_k^{-1}\nabla f(x_k)$ is actually a descent direction, we can calculate the directional derivative in the direction d_t(inner product between the gradient and the direction) $\nabla f(x_k)^T d_t$, for $d_t = -H_k^{-1}\nabla f(x_t)$:

$$\nabla f(x_k)^T d_k = -\nabla f(x_k)^T H_k^{-1}\nabla f(x_k) \tag{A.142}$$

Here H_t^{-1} is positive definite, because H_k is positive definite. When a positive definite matrix pre- and post-multiplied by a non-zero vector, then it will be positive(by definition of positive definiteness),and with the minus the derivative will be negative indeed. As a result it must be a descent direction in x_k.

For H_k the only restriction is that it should by *symmetric positive defininte*, thus PCG is actually a family of methods.

Descent and Newton If Newton method works it works pretty well. So it might be advantageous to combine somehow the robustness of descent methods with the fast convergence of Newton method.

For a d_k direction, both are of type

$$x_{k+1} = x_k + \alpha_k \tag{A.143}$$

Descent method:

$$x_{k+1} = x_k - \alpha_k D_k \nabla f(x_k) \tag{A.144}$$

$$d_k = -D_k \nabla f(x_k) \text{ minus the gradient multiplied by PSD preconditioner} \tag{A.145}$$

Newton:

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k) \; \alpha_k = 1 \tag{A.146}$$

$$d_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k) \text{ minus the gradient multiplied by inverse of Hessian} \tag{A.147}$$

It can be seen, that in both case matrix times minus the gradient. Thus newton method can be seen as a descent method as well, which works however only if the inverted Hessian $H = \nabla^2 f(x_k)^{-1}$ is PSD.

(also descent allows to use any step size in the direction, in Newton $\alpha_k = 1$)

If this is not the case, that is the Hessian is not PSD, then we have use something else, that in similar to H:

1. $D_k = I$ steepest descent algorithm - slow compared to Newton

2. include curvature information D_k diagonal preconditioner:(TODO notation)(for psd it is enough to have elements in the diagonal to be positive)

$$D_k(i,i) = \left(\max \varepsilon, \frac{\partial^2}{\partial x_i^2} f(x_k) \right)^{-1} \tag{A.148}$$

   for $\varepsilon > 0$ to have sufficiently positive, thus $D_k(i,i) \geq \frac{1}{\varepsilon} > 0$

3. more sophisticated : "inflating "$\nabla^2 f(x)$:

$$D_k = (\nabla^2 f(x_k) + \tau I)^{-1} \tag{A.149}$$

   where τ is given such that D_k PSD. Its advantages is that it uses the full Hessian.

Conjugate Gradient Descent [95]

When we do exact line-search along the gradient, each time we try to find the minimum $x_t$along the direction $d_t = \nabla f(x_{t-1})$. If this descent directions are not orthogonal to d_{t-1}, that means that there is a component that belongs to the previous direction,that is we could have travelled further more along d_{t-1}. This is again the zigzag. So in n dimension the idea is to search in n conjugate directions. In this case progress made in one direction does not affect the progress made in the other. So only search in n dimensions to find the optimal points. Two vectors are *conjugate* or *Q-orthogonal* if defining an inner product with Q Hessian $\langle u,v\rangle = u^T Qv = 0$, and a set of pairwise Q-orthogonal vectors form a basis of $\mathbb{R}^n$. So for an n dimensional function we can transform the function to the following form:

$$l(x) = \frac{1}{2}x^T Qx - x^T b + c, \tag{A.150}$$

whose gradient is $\nabla l(x) = Qx - b$, so if we minimize it $Qx = b$.

CGMethod : For use the Conjugate gradient method we need $\{d_1,...\} \in \mathbb{R}^n Q$-orthogonal vectors, since they form a basis $x* = \sum_{i=1}^n \alpha_i d_i$, since $d_k Q d_i = 0$ for $k \neq i, \alpha_k = \frac{d_k^T b}{d_k^T Q d_k}$, thus after calculating α_i-s we have found the minimum $x^* = \sum_{i=1}^n \alpha_i d_i$. What is needed to start the method is to find d_i orthogonal vectors.

$$Qx^* = \sum_{i=1}^{n} \alpha_i Q d_i \tag{A.151}$$

where $Qx^* = b$ at the minimum, thus $b = \sum_{i=1}^n \alpha_i Q d_i$ a linear combination of Q-times the original Q-orthogonal vectors. Multiliyng both side by one of these original vectors:

$$d_k^T b = \sum_{i=1}^{n} \alpha_i d_k^T Q d_i \text{ by orthoganility of} d_i \text{ and } d_k \text{ for } i \neq k \tag{A.152}$$

$$d_k^T b = \alpha_k d_k^T Q d_k \tag{A.153}$$

$$\alpha_k = \frac{d_k^T b}{d_k^T Q d_k} \tag{A.154}$$

Having the αs allows to calculate the minimum value of f directly.

How to calculate the n Q othogonal vevtors? Could use $Qv = \lambda v$, eigenvectors, that is inefficient. Instead generate them dynamically from an initial guess x_1:

$$\mathbf{d_1} = -\nabla f(x_1) = b - Qx_1 \tag{A.155}$$

$$\alpha_k = \frac{-\nabla f(x_k)^T d_k}{d_k^T Q d_k} \qquad \Rightarrow\ x_{k+1} = x_k + \alpha_k d_k \tag{A.156}$$

$$\beta_k = \frac{\nabla f(x_{k+1})^T Q d_k}{d_k^T Q d_k} \qquad \Rightarrow d_{k+1} = -\nabla f(x_{k+1}) + \beta_k d_k \tag{A.157}$$

That is d_i-s are the gradient directions, while α_i-s are the corresponding stepsizes.

When we do this iteratively we will find a solution close enough to x^*

A.2.5 Bregmann divergence and mirror descent

Projected Gradient Descent (PGD) Projected gradient method can be viewed as a generalization of GD that is usable for constrained optimization as well. If first takes a step in the direction of the negative gradient

$$y_{t+1} = x_t - \eta_t \nabla f(x_t); \tag{A.158}$$

then projects it back to the feasible region C:

$$x_{t+1} = \arg\min_{x \in C} \|y_{t+1} - x\|. \tag{A.159}$$

In a single step:

$$x_{t+1} = \arg\min_{x \in C} = \left\{ f(x_t) + \langle \nabla f(x_t), x - x_t \rangle + \frac{1}{2\eta_t} \|x - x_t\|_2^2 \right\} \tag{A.160}$$

that can be interpreted as keeping the new new point in a proximity of the old one, and making the gradient step along the tangent with the slope of the sub-gradient (Figure

A.8). In other words, we assume that in the proximity of the current approximation x_t the gradient or tangent line is close enough to the real function, but as we go further there is a high uncertainty, which we want to penalize.

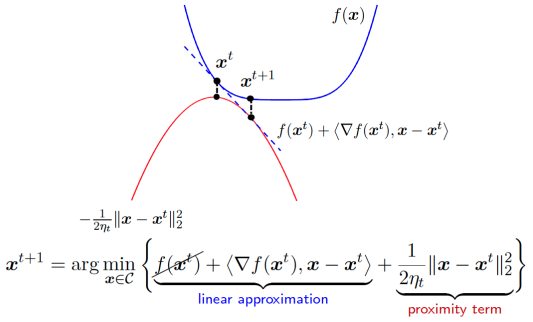


Figure A.8 Projected Gradient descent [43]

To sum up, in PGD the proximity term $\|x - x^{(t)}\|_2^2$ is based on the assumption, that the discrepancy between the f and its tangent might be well approximated by this homogeneous penalty. Since this belief often does not hold, *mirror descent* uses a generalization of the proximity term, the so-called *Bregman-divergence*

Bregman-divergence Bregman divergence (first used in [31]) is a generalization of squared Euclidean distance:

$$\|x-y\|^2 = \langle x-y, x-y\rangle = \|x\|^2 - \|y\|^2 - \langle \underbrace{2y}_{\nabla\|y\|^2}, x-y\rangle \tag{A.161}$$

Here $2y$ equals to the derivative of $\|y\|^2$, moreover $\|y\|^2 + \langle 2y, x-y\rangle$ is equal to the equation of the tangent line at y, thus the whole expression measures the difference between the function $f(x) = \|x\|^2$ and its tangent line at y evaluated at x (Figure A.9). The non-negativity of this distance in any point x and y is equivalent with the distance measure being convex.

Thus for a ϕ strictly convex and differentiable function on C:

$$\phi(z) \geq \phi(x) + \langle z-x, \nabla\phi(x)\rangle, \ \forall z, \text{ (Convexity definition).} \tag{A.162}$$

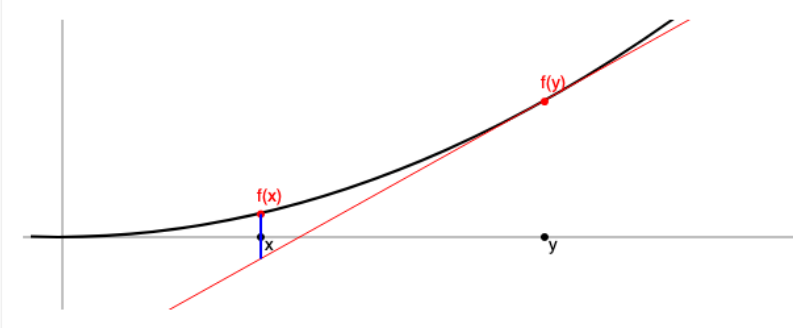


Figure A.9 Geometric interpretation of Euclidean distance from [175] Red line is the tangent of $\|x\|^2$ at y, and length of the blue line is the distance $d^2(x,y) = \|x-y\|^2$

.

we can derive a distance function D_ϕ, and these distance measures (or divergences) are called together *Bregman-divergence*:

$$D_\phi(z,x) = \phi(z) - \phi(x) - \langle \nabla\phi(x), z-x \rangle. \tag{A.163}$$

Examples that can be derived in this form could be :

- Mahalanobis distances, a "distorted Euclidean norm": $\phi_A = \frac{1}{2}x^T Ax$, for a matrix $A \in \mathbb{R}^n$
- KL-divergence: $\phi_{\mathrm{KL}} = \sum_{i=1}^n p_i \log p_i$(negative Shannon entropy)

The goal of using the in optimization divergence is to adjust the step-size of the gradient descent based on a "distance" along a function ϕ that fits better the local curvature of function f to be minimized, potentially incorporating knowledge about some constraints as well .

Mirror descent Mirror descent (Figure A.10) is essentially a projected gradient descent that instead of squared Euclidean distance uses a Bregman-divergence D_ϕ to penalize going to far from the current point. The concrete function ϕ should be chosen such way, that it

- fits to the local curvature of f
- fits to the geometry of the constraint set C

- makes sure that projection in the step $\arg\min_z D_\phi(z,x)$ can be computed easily.

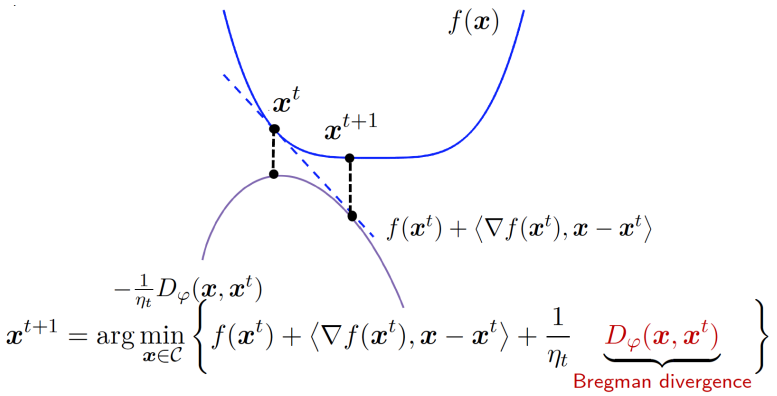


Figure A.10 Mirror descent with Bregman divergence from [43]

A.2.6 Quasi Newton methods

Quasi newton methods are approximating the inverse Hessian B_t using gradients ($g \overset{\text{def}}{=} \nabla f$), thus saving the not negligible cost of explicitly calculating the Hessian and its inverse, as it is done in Newton methods and conjugate gradients .

The main steps of these Quasi Newton methods thus:

1. Compute an approximate Newton direction: $d_t = -B_t g_t$
2. line search (inexact): $\eta_t = \arg\min_\eta f(w_t + \eta_t d_t)$
   $w_{t+1} = w_t + \eta_t d_t$
3. Compute gradient at w_{t+1}: $g_{t+1} = \nabla f(w_{t+1})$
4. Update the approximate inverse Hessian :

$$B_{T+1} = \text{UpdateFormula}(B_t, \underbrace{w_{t+1} - w_t}_{\overset{\text{def}}{=} p_t}, \underbrace{g_{t+1} - g_t}_{\overset{\text{def}}{=} q_t}) \tag{A.164}$$

For quadratic functions, exact Hessian and exact line-search quasi newton methods become equivalent to conjugate gradient. From the definition of Hessian: $H = \frac{dg}{dx} \rightarrow H \cdot dx = dg$

follows the *secant equation*

$$Hp_t = q_t \tag{A.165}$$

Rank one update for keeping the update of the Hessian simple (rank one thus can be a product of vectors: u and v)

$$H_{t+1} = H_t + uv^T. \tag{A.166}$$

This update should satisfy the secant equation:

$$(H_t + uv^T)p_t = q_t \text{ (secant equation)} \tag{A.167}$$

$$u(v^T p_t) = q_t - H_t p_t \tag{A.168}$$

$$u = \frac{1}{V^T p_t}(q_t - H_t p_t), \tag{A.169}$$

where v can be any vector, still the secant equation will be satisfied with the resulted u. We can put additional requirements on v. Since the Hessian is symmetric, we want the update to be symmetric as well. This means u and v should be collinear. Using this

$$u = \underbrace{\frac{1}{v^T p_t}}_{\text{scalar}}(q_t - H_t p_t) \tag{A.170}$$

$$v = q_t - H_t p_t \text{ and } u = \frac{1}{v^T p_t} v \tag{A.171}$$

$$H_{t+1} = H_t - \frac{1}{v^T p_t} vv^T \tag{A.172}$$

Then what we want to approximate the newton step in the inverse $B = H^{-1}$

$$H \cdot dx = dg \;\Rightarrow\; Hp_t = q_t \;\Rightarrow\; Bq_t = p_t,\; B = H^{-1} \tag{A.173}$$

$$v = p_t - B_t g_t \tag{A.174}$$

$$B_{t+1} = B_t - \frac{1}{vq_t} vv^T \tag{A.175}$$

For Rank two update: Broyden–Fletcher–Goldfarb–Shanno (BFGS) In BFGS [145], U, and V two rank, that is two column matrices, with which we can satisfy more requirements, not only the secant equation but also we want the approximate inverse Hessian to be close to previous one:

$$||B_{t+1} - B_t||_W \rightarrow \min \tag{A.176}$$

$$B_{t+1} > 0 \tag{A.177}$$

for closeness: "W-norm": $||A||_W = ||W^{1/2} A W^{1/2}||_F$(where for PSD W:$W^{1/2} = A$, for $AA = A^T A = W$).

If we use $W \approx H$ results in the BFGS methodm while for the coice $W \approx H^{-1}$ David-Fletchel-Powel (DFP) method.

Limited memory BFGS (LBFGS) L-BFGS [33] is a verison of BFGS, that does not require storing if the Hessian on its inverse thus applicable for solving problem with huge number of variables.

$$B_{t+1} = B_t - \frac{1}{v q_t} v v^T \text{ BFGS} \tag{A.178}$$

$$d_t = B_t g_t = (\underbrace{B_0}_{:=\alpha I} + U_1 V_1^T + U_2 V_2^T + \dots) g_t \tag{A.179}$$

Where we don't actually need B_t only the product with g_t:

$$d_t = \alpha g_t + U_1 (V_1^T g_t) + U_2 (V_2^T g_t) + \dots \tag{A.180}$$

Stochastic Quasi Newton methods To mimic the LBFGS methods a range of stochastic algorithm has been proposed as in [27] and [131][128][231], that are seeking to model local curvature information using inexact gradients coming from SGD procedure. [162] tries to combine these with SVRG, while [87] makes an attempt to utilize stochastic matrix inversion techniques to develop stochastic LBFGS that can be used along with SVRG.

A.3 Information theory and Bayesian Learning

The most fundamental concept of information theory is the amount of information some data sample carries. The **Shannon information content** of an event x is defined : $I(x) = -\log(p(x)),$ for $x \in X$(X all events), that is if an event occurs with the higher probability, means the lower the information content, or the smaller surprise factor.

A common practice is to measure information content with *bits*, that means using base 2 logarithm. $I(x) = 1$ for example for equally probably binary events $p(X = \text{"head"}) = \frac{1}{2} \rightarrow I = \log_2(\frac{1}{2}) = 1\text{bit}$

Shannon Entropy describes expected information content of random variable $X \sim p$ drawn from a distribution p. For discrete random variables it is defined as:

$$H(X) = \mathbb{E}_{x\sim p}[I(x)] = -\sum_{x\in X} p(x)\log_2(p(x)). \tag{A.181}$$

That is for continuous random variables equivalent with:

$$H(X) = \mathbb{E}_{x\sim p}[I(x)] = -\int p(x)\log_2(p(x))dx \tag{A.182}$$

Cross Entropy measures the relative entropy between two distribution, calculated by summing over all events the product of the probability of an event x according to distribution p with the negative log of its probability according to the other distribution q:

$$H_p(q) = -\sum_x q(x)\log_2(p(x)) = \mathbb{E}_{x\sim q}[-\log p(x)] \tag{A.183}$$

As a loss function *cross entropy* measures the difference between *q predicted distribution* and *p ground truth*, so the better the approximation of p, the closer the two entropy is.

Morover it is easy to see that:

$$H_p(q) \geq H_p(p) \tag{A.184}$$

Kullback-Leibler divergence (KL-divergence) describes how much one distribution diverges from another, that can be used to measure how much information we lose when we approximating a complex *true distribution* with our *model distribution*.

KL-divergence can be calculated by multiplying the probability of an event happening in reality (p distribution) with the difference of the of the logs of this real-world with its predicted probability in q:

$$D_{KL}(p||q) = \mathbb{E}[p(x)(\log p(x) - \log q(x))] \sum_{i=1}^{N} p(x_i)(\log p(x_i) - \log q(x_i)) \tag{A.185}$$

$$= \sum_{i=1}^{N} p(x_i) \log \frac{p(x_i)}{q(x_i)} \tag{A.186}$$

Properties of KL-divergence:

1. non negative - $p(x) = q(x) \rightarrow \log \frac{p(x)}{q(x)} = 1 \rightarrow KL(p||q) = 0$
2. non symmetric "relative distance"

Information loss also can be derived by the difference between the cross entropy $H_p(q)$ and the *ideal entropy* $H(q)$:

$$H_p(q) - H(q) = -\sum_x q(x) \log_2(p(x)) - (-\sum_x q(x) \log_2(q(x)) \tag{A.187}$$

$$= \sum_x q(x) \log_2(q(x) - \sum_x q(x) \log_2(p(x)) \tag{A.188}$$

$$= \sum_x q(x)(\log_2(q(x)) - \log_2(p(x)))) \tag{A.189}$$

$$= \sum_x q(x) \log_2(\frac{q(x))}{p(x)}) \tag{A.190}$$

$$= KL(q||p) \tag{A.191}$$

From Equation A.184 it follows, that optimizing cross entropy equivalent with optimizing KL divergence.

Conditional entropy $H(Y|X)$ is the entropy of Y given X known

$$H(Y|X) = \sum_{x \in X} p(x) H(Y|X = x) = -\sum_{x \in X, y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)}. \tag{A.192}$$

That is, it quantifies the amount of information needed to describe the value of a random variable Y when X is observed.

We use the term **Categorical cross entropy** in machine learning when evaluating the model performance on a C-way classification task. In a typical case when we want to predict a class of the input data the target is a categorical or generalized Bernoulli distribution given the input as a one-hot vector. To convert the output of the model into a probability distribution over the possible classes we usually use `softmax` function on the unnormalized model output often called as `logit` vector : $S(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}$

Thus the categorical cross entropy loss for a single data point (x,y):

$$CE = -\sum_{c=1}^{C} y_c \log(\hat{y}_c) = -\sum_{c=1}^{C} y_c \log \left(\frac{e^{\hat{y}_c}}{\sum_{j=1}^{C} e^{\hat{y}_j}} \right) \tag{A.193}$$

for y_i is the true label of sample , while $\hat{y}_i$ is the predicted, for N samples and C possible classes. If the target is a one hot vector this is simplified down to

$$CE = -\log \left(\frac{e^{\hat{y}_t}}{\sum_{j=1}^{C} e^{\hat{y}_j}} \right) \tag{A.194}$$

for index t denotes the coordinate corresponding to the true label, since $y_c = 1$ for the true label t, and $y_c = 0$ in any other case.

Likelihood and probability Likelihood is the likelihood of the parameters given the data that is the probability of the data given the parameters.

$\mathcal{L}(\mathbf{w}) \propto p(y|\mathbf{w}, \sigma^2)$ $(= p(y|\theta))$ with a Gaussian prior.

In a stochastic process, when we have a parameter θ that describes the process, the probability of outcomes/events O is $p(O|\theta)$. When we want to model a real world process we have the observed outcomes the goal is to estimate θ. The natural way to approximate θ is to choose one that maximizes the probability of observing O:

$$\max_{\theta} P(O|\theta) \stackrel{\text{def}}{=\!=} \max_{\theta} L(\theta|O) \tag{A.195}$$

Thus likelihood measures how good the parameters fit to the observed data, while probability, how probable is to draw a given sample given the parameters.

Log-likelihood In ML we try to maximize likelihood of the parameters, that is the probability of the data given the parameters $p(\mathcal{D}|\theta) = \prod_{(x,y)\in\mathcal{D}} p(y|x,\theta)$. that is in fact equals to $l(\theta|\mathcal{D}) = \prod_{(x,y)\in\mathcal{D}}(\theta|x,y)$, as it is stated in Equation A.195.

Since being the probability (and thus likelihood) independent across all the training examples, that is often a reasonable assumption, we usually can decompose the likelihood into product of likelihood over all examples Thus for for N samples the likelihood and log likelihood :

$$L(\theta|\mathcal{D}) = L(\mathcal{D}|\theta) = \prod_{i=1}^{N} p(y_i|x_i,\theta) \tag{A.196}$$

When one seeks to find the best parameters θ with some method that is based on gradients, the product becomes slightly inconvenient, thus a much easier method to work with the logarithm of the likelihood.

$$LL(\theta|\mathcal{D}) = \sum_{i=1}^{N} \log(p(y_i|x_i,\theta)) \tag{A.197}$$

Since the natural logarithm is a monotonically increasing function the optimization will lead to the same optimum:

$$\theta^* = \arg\max_{\theta} L(\theta|\mathcal{D}) = \arg\max_{\theta} \sum_{i=1}^{n} \log L(\theta|x_i, y_i) \tag{A.198}$$

Cross Entropy and Log-Likelihood let p be the true distribution, and q the predicted distribution with y_i is the target probability distribution for x_i feature vector of the ith sample

$$p(y|x_i) = \begin{cases} 1 & \hat{y}_i = y_i \\ 0 & otherwise \end{cases}$$

Thus for a sample x_i the cross entropy loss $H_i(p,q) = -\sum_{y_i \in Y} p(y_i|x_i) \log(q(y_i|x_i))$. The loss computed from Equation A.3 reduces to $H_i(p,q) = -\log(q(y_i|x_i))$ (see Equation A.194, thus the expected cross entropy becomes $H(p,q) = -\sum_{i \in N} \log(q(y_i|x_i))$, that is in fact the equals the negative log likelihood. Thus maximizing the log likelihood is equivalent to minimizing cross entropy.

The derivativeof the `softmax` function, that can be used in an gradient based maximum likelihood estimation, that is the predicted distribution for w.r.t. the `logit` vector **a**: (wrong notation)$q(y_i|x_i) = S(x_i) = \frac{e^{a_i}}{\sum_{j=1}^{C} e^{a_j}}$

$$q_i = \frac{e^{a_i}}{\sum_{j=1}^{C} e^{a_j}} \tag{A.199}$$

$$\frac{\partial q_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{j=1}^{C} e^{a_j}}}{\partial a_j} \tag{A.200}$$

Applying Quotient rule if $f(x) = \frac{g(x)}{h(x)}$, $f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{h^2(x)}$:

For $i = j$:

$$\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{C} e^{a_k}}}{\partial a_j} = \frac{e^{a_i} \sum_{k=1}^{N} e^{a_k} - e^{a_j} e^{a_i}}{(\sum_{k=1}^{N} e^{a_k})^2} \tag{A.201}$$

$$= \frac{e^{a_i} (\sum_{k=1}^{N} e^{a_k} - e^{a_j})}{(\sum_{k=1}^{N} e^{a_k})^2} \tag{A.202}$$

$$= \frac{e^{a_i}}{(\sum_{k=1}^{N} e^{a_k})} \frac{(\sum_{k=1}^{N} e^{a_k} - e^{a_j})}{(\sum_{k=1}^{N} e^{a_k})} \tag{A.203}$$

$$= q_i(1 - q_j) \tag{A.204}$$

For $i \neq j$:

$$\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{C} e^{a_k}}}{\partial a_j} = \frac{0 - e^{a_j} e^{a_i}}{(\sum_{k=1}^{N} e^{a_k})^2} \tag{A.205}$$

$$= \frac{-e^{a_j}}{(\sum_{k=1}^{N} e^{a_k})} \frac{(e^{a_i})}{(\sum_{k=1}^{N} e^{a_k})} \tag{A.206}$$

$$= -q_j q_i \tag{A.207}$$

Cross entropy loss indicates distance between the models output and and the target distribution: $H(y,q) = \sum_i y_i \log(q_i)$, for $\mathbf{y}$ target vector and $\mathbf{q}$ prediction. It is widely used as an alternative of squared error, when output activations can be understood as a probability of the different hypotheses might be true, that is a distribution over the possibilities. It is used as the loss function for NN, with softmax output activation functions. So the derivative of the *Cross entropy loss* w.r.t. activation $\nabla_{\mathbf{a}} f(x,y)$ using the prediction calculated by the `softmax`:

$$f(\mathbf{x},\mathbf{y}) = -\sum_i y_i \log(q_i) \tag{A.208}$$

$$\frac{\partial f(\mathbf{x},\mathbf{y})}{\partial a_i} L = -\sum_k y_k \frac{\log(q_k)}{\partial a_i} \tag{A.209}$$

$$= -\sum_k y_k \frac{\log(q_k)}{\partial q_k} \frac{\partial q_k}{\partial a_i} \tag{A.210}$$

$$= -\sum_k y_k \frac{1}{a_k} \frac{\partial q_k}{\partial a_i} \tag{A.211}$$

$$\tag{A.212}$$

From the derivative of softmax w.r.t pre-nonlinearity activation :

$$\frac{\partial L}{\partial a_i} = -y_i(1-q_i) - \sum_{k\neq i} y_k \frac{1}{q_k}(-q_k p_i) \tag{A.213}$$

$$= -y_i(1-q_i) + \sum_{k\neq i} y_k q_i \tag{A.214}$$

$$= q_i(\underbrace{y_i + \sum_{k\neq i} y_k}_{=1}) - y_i \tag{A.215}$$

$$= q_i - y_i \tag{A.216}$$

A.4 Bayesian Learning

Bayesian view of ML/DL In Bayesian perspective[24] NN/ML probabilistic model $P(\mathbf{y}|\mathbf{x},\mathbf{w})$, given an input $\mathbf{x} \in \mathbb{R}^d$ the model assigns a probability to each possible output $\mathbf{y} \in \mathcal{Y}$, using a set of parameters $\mathbf{w}$.

Bayesian learning techniques build on Bayes theorem in looking for the best *hypothesis* H for explaining the *evidence* E:

$$p(H|E) = \frac{P(E|H)P(H)}{P(E)}. \tag{A.217}$$

In ML terms of ML this looks as follows:

$$p(\mathbf{w}|\mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{w})P(\mathbf{w})}{P(\mathbf{x})} \tag{A.218}$$

or going even more specific, we want to model $p(y|\mathbf{x},\mathbf{w})$, where task is

- classification $y \in C$ for C a set of classes,and $p(y|\mathbf{x},\mathbf{w})$ is a categorical distribution
- or regression y being a continuous variable and the model describing a Gaussian distribution.

The most fundamental problem in ML is **Density estimation** , that is selecting a probability distribution $p(x)$ and its parameters that best explains our data x. It is actually an unsupervised method, that can be used for supervised tasks.

Being $p(x)$ known enables to

- identify outliers
- fill in missing data
- make quatization - shorter code to x with high p(x)
- compute association rules from consitionals $p((x_i)_j,(x_i)_k)$
- latent factor
- perform clustering

Or with joint density estimation $p(x_i,y_i)$

- supervised learning
- feature relevance

Since estimating density $p(x)$ (or $p(x|w)$)it is usually quite challenging, a range of tricks are being used focusing on different aspects of Bayes theorem.

Maximum Likelihood Estimation Given a data set $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}$ we want to maximize the likelihood of the parameter $\mathbf{w}$ (that is to find the set of parameters ($\mathbf{w}$) that maximize the likelihood function, e.g. result in the largest likelihood value):

$$\max_{\mathbf{w}} p(\mathcal{D}|\mathbf{w}), \text{ for} \tag{A.219}$$

$$p(\mathcal{D}|\mathbf{w}) = \prod_i p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) \tag{A.220}$$

For training such a model the optimization method used is *negative log likelihood*, that is in classification (categorical distribution) corresponds to the *cross entropy* error function, while for regression problems (Gaussian distribution) to *mean squared error (MSE)*.

The way to learn weights by **maximum likelihood estimation** is to maximize the log likelihood of observing $\mathcal{D}$:

$$\mathbf{w}^{\text{MLE}} = \arg\max_{\mathbf{w}} \log P(\mathcal{D}|\mathbf{w}) = \arg\max_{\mathbf{w}} \sum_i \log P(\mathbf{y}_i|\mathbf{x}_i\mathbf{w}) \tag{A.221}$$

This is typically done by GD assuming that $P(\mathcal{D}|\mathbf{w})$ is differentiable in $\mathbf{w}$.

However, MLE provides us with point estimation, and ignores uncertainty of the values of the weights, that might lead to overfitting, that is in fact happens too often in NNs.

Maximum a Posteriori (MAP) To tackle the problem of overfitting the most popular way is to apply regularization, in which case the optimization process will be similar to searching for a Maximum a Priory estimation, by introducing prior on the weights, such as we want the weights to be relatively low and mostly 0 (normal distribution $\mathcal{N}(0, \sigma^2)$)

MAP is about "approximating" the distribution omitting the usually intractable partition function/marginal likelihood $p(\mathcal{D})$, since that is a constant:

$$P(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})} \propto p(\mathcal{D}|\mathbf{w})p(\mathbf{w}) \tag{A.222}$$

Thus the maximum a posteriori estimation:

$$\mathbf{w}^{\text{MAP}} = \arg\max_{\mathbf{w}} \log P(\mathbf{w}|\mathcal{D}) = \arg\max_{\mathbf{w}} \log P(\mathcal{D}|\mathbf{w}) + \log P(\mathbf{w}). \tag{A.223}$$

With Gaussian prior MAP results in L2 regularization, while Laplace prior yields L1 regularization.

The optimization methods for finding a MAP estimate are basically the same as in MLE adding a log prior probability.

Posterior Inference With a full posterior distribution $P(\mathbf{w}|\mathcal{D})$,we can make predictions that takes into account the uncertainty of the weights as well:

$$p(y|\mathbf{x},\mathcal{D}) = \int p(y|\mathbf{x},\mathbf{w})p(\mathbf{w},\mathcal{D})d\mathbf{w}, \tag{A.224}$$

where the parameters $\mathbf{w}$ are marginalized out, thus turning prediction into an expectation, or in other words into a average over an ensemble of predictors weighted by the posterior probabilities for their parameters $\mathbf{w}$.

The problem is however, that computing analytically the this posterior $p(\mathbf{w}|\mathcal{D})$ is intractable. The idea of variational inference is therefore to approximate somehow the posterior with a *variational distribution* $q(\mathbf{w}|\theta)$, with a known functional form with parameters θ, that will estimate the distribution of over $\mathbf{w}$.

From a Bayesian perspective the correct way is to do posterior inference these are Bayesian NNs. First approaches where Laplace method (low complexity) and MCMC (long convergence and difficult to train), now Variational inference methods are considered state-of-the-art [24].

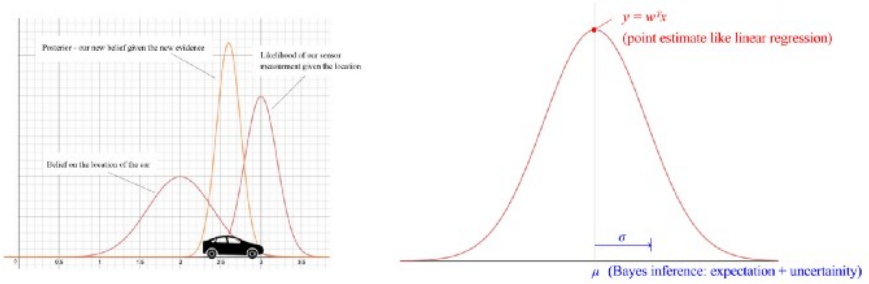


Figure A.11 Bayesian inference vs point estimation, borrowed from [104]

A.5 Training from Bayesian perspective

Given a model m the probabilistic prediction of a target variable is $p(y|m) = \int p(y|\theta,m)p(\theta|m)d\theta$.

Even for the case, when θ is a Gaussian this is a very hard problem to solve. Morover the possible presence of some latent variables z results in additional dimensions that need to be marginalized out $p(y|m) = \int\int p(y,z|\theta,m)p(\theta|m)dzd\theta$

Solving these very high dimensional integrals is analytically not thus we need numerical methods to approximate the posteriors and marginal likelihoods. There are a number of methods developed for making these approximations as:

- Laplace approx
- Bayesian Information Criterion
- variational approximation
- expectation propagation
- MCMC
- exact sampling

Here we will only focus on the state-of-the art so-called *variational approximations*, that are considered accurate and econmomical:

- Expectation propagation
- Variational Inference

Expectation Propagation (EP)

We assume that the data $\mathcal{D} = \{x^{(1)}, \ldots, x^{(N)}\}$, is drawn iid from some distribution. We want to have a model $p(x|\theta)$, with prior $p(\theta)$. The pararameter posterior, that we want to find during the training involves then the likelihood of each individual datapoints:

$$p(\theta|\mathcal{D}) = \frac{1}{p(\mathcal{D})} p(\theta) \prod_{i=1}^{N} p(x^{(i)}|\theta), \tag{A.225}$$

using the iid assumptions over datapoints : $\prod_{i=1}^{N} p(x^{(i)}|\theta) = p(\mathcal{D}|\theta)$. Ignoring the normalizing constant we can rewrite this, as

$$p(\theta) \prod_{i=1}^{N} p(x^{(i)}|\theta) = \prod_{i=0}^{N} f_i(\theta), \tag{A.226}$$

with $f_0(\theta) \stackrel{\text{def}}{=} p(\theta)$ and $f_i(\theta) \stackrel{\text{def}}{=} p(x^{(i)}|\theta)$, for some functions f of θ, thus the first factor is the prior and the others correspond to the contributions of each data points. So the posterior is product of a bunch of functions over θ.

The goal is to approximate this with simpler terms: $q(\theta) = \prod_{i=1}^{N} \tilde{f}_i(\theta)$.

Since $f_i(\theta)$ can be complicated and multiplying them together is very hard, and for complicated distributions each data point adds some complicated thing to the posterior.

It might be a solution to approximate with some distribution with which it is easier to work as Gaussians or exponential distributions in general, where multiplication and division are relatively simple.

To find the appropriate approximation to the true likelihood is $f_i(x_i|\theta)$, possible ways could be:

- $\min_{q(\theta)} KL(\prod_{i=0}^{N} f_i(\theta) || \prod_{i=0}^{N} \tilde{f}_i(\theta))$ -intractable - globally minimize KL divergence - this would be the best approximation, exact mean and variance of true posterior.
- $\min_{\tilde{f}_i(\theta)} KL(f_i(\theta) || \tilde{f}_i(\theta))$ simple non-iterative, inaccurate. for each factor only go through once. but each one produces some error, that will be multiplied together.

- $\min_{\tilde{f}_i} KL(f_i(\theta)\prod_{i\neq j}^{N}\tilde{f}_j(\theta)||\tilde{f}_i(\theta)\prod_{i\neq j}^{N}\tilde{f}_j(\theta))$ simple iterative, accurate $\rightarrow$ EP - iterates approximating terms but multiplies by all the other approximating terms - can be applied to many things from graphical models, belief propagation, kernel machines etc..

So the EP algorithm for an exponential prior over θ, and sequential updating order looks the following:

- having an original input model $f_0(\theta)\dots f_N(\theta)$
- initialize $\tilde{f}_0(\theta) = f_0(\theta,\dots,\tilde{f}_N(\theta) = f_N(\theta)$
- repeat
  - For i=1...N do
    - Deletion $q_{\setminus i} \leftarrow \frac{q(\theta)}{\tilde{f}_i(\theta)}$
    - Projection $\tilde{f}_i^{\text{new}} \leftarrow \arg\min_{f(\theta)} KL(f_i(\theta)q_{\setminus i}(\theta)||f(\theta)q_{\setminus i}(\theta))$
    - Inclusion $q(\theta) \leftarrow \tilde{f}_i^{\text{new}} q_{\setminus i}(\theta)|$

So the essence of EP it is about updating our model, that gives the estimation of the likelihood in each step to get closer to the actual likelihood. This corresponds to take a single example into consideration in each step and fit the model to give the true likelihood for the point.

Minimization is done via *matching moments*. Since in the inclusion step multiplying together probability distributions, the new exact moments will be incorporated into the common distribution, thus improving the shared estimated parameters.

Evidence Lower Bound (ELBO) If cannot compute a posterior distribution $p(z|x) = \frac{p(z,x)}{p(x)} = p(z|x)$, we can approximate with $q(z)$, using KL divergence to evaluate quality of

the estimation:

$$KL(q(z)||p(z|x)) = -\sum q(z)\log\frac{p(z,x)}{p(x)}\frac{1}{q(z)} \tag{A.227}$$

$$= -\sum q(z)\log\frac{p(z,x)}{q(z)}\frac{1}{p(x)} \tag{A.228}$$

$$= -\sum q(z)[\log\frac{p(z,x)}{q(z)} + \log\frac{1}{p(x)} \tag{A.229}$$

$$= -\sum q(z)[\log\frac{p(z,x)}{q(z)} - \log p(x) \tag{A.230}$$

$$= -\sum q(z)\log\frac{p(z,x)}{q(z)} + \sum q(z)\log p(x) \tag{A.231}$$

$$KL(q(z)||p(z|x)) + \underbrace{\sum q(z)\log\frac{p(z,x)}{q(z)}}_{\underset{=\mathcal{L}:\text{lower bound}}{\text{def}}} = \underbrace{\sum q(z)\log p(x)}_{=\log p(x)} \tag{A.232}$$

$$\underbrace{\underbrace{KL}_{\text{always}\geq 0} + \underbrace{\mathcal{L}}_{\text{always}\leq 0}}_{\mathcal{L}\leq -KL} = \underbrace{\log p(x) and fixed}_{\text{always}\leq 0} \tag{A.233}$$

Since x is given we are looking for $p(z|x)$ through manipulating KL and the *lower bound*$\mathcal{L}$.

Thus the idea of evidence lower bound method for approximating $p(z|x)$ with $q(z)$, is instead of minimizing KL we equivalently can maximize $\mathcal{L}$, that is much easier task, since it works only with jkoint probability instead of conditional:

$$\mathcal{L} = \sum q(z)\log\frac{p(x,z)}{q(z)} \tag{A.234}$$

$$KL = -\sum q(z)\log\frac{p(x|z)}{q(z)} \tag{A.235}$$

Thus the term *variational* means using variation inference that is a method that approximates maximum likelihood, when probability density is complicated and

To sum up, ELBO uses evidence lower bound as a proxy, for example using KL divergence:

$$\log(p(x)) \leq \underbrace{\mathbb{E}_q[\log(p(x,Z))] - \mathbb{E}_q[\log(q(Z))]}_{\mathcal{L}\text{- ELBO}}. \tag{A.236}$$

The point is to optimize this ELBO there are also other type of divergences as in [243].

Bayesian Neural Networks The essence of Bayesian deep learning is to look at the network as a conditional model $p_{\mathbf{w}}(y|x)$ parametrized by weights $\mathbf{w}$, that returns an y when some input x is given.

Training NN/ML models from Bayesian perspective means calculating $P(\mathbf{w}|\mathcal{D})$, the posterior distribution of weights given the training data,

$$P(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w}))}{p(\mathcal{D})} \tag{A.237}$$

$$\tag{A.238}$$

The prediction using the Bayesian model then is done by taking an expectation of the distribution of $\hat{\mathbf{y}}$ given the parameters $\mathbf{w}$and data $\mathbf{x}$:

$$P(\hat{\mathbf{y}}|\mathbf{x}) = \mathbb{E}_{P(\mathbf{w}|\mathcal{D})}[P(\hat{\mathbf{y}}|\mathbf{x},\mathbf{w})] \tag{A.239}$$

$$= \int p(\hat{\mathbf{y}}|\mathbf{x},\mathbf{w})p(\mathbf{w}|\mathcal{D})d\mathbf{w}. \tag{A.240}$$

That means that each possible configuration of weights corresponds a different prediction of the unknown label, weighted by the parameter posterior. This is equivalent to using an ensemble of uncountably infinite number of NNs. This is however intractable for reasonably sized NN.

To answer this problem form Hinton and Van Camp [100] and Graves [89] is to use variational approximation to the Bayesian posterior. Variational learning is about finding parameters θ of a distribution of weights $q(\mathbf{w}|\theta)$ that minimizes KL divergence with the

true Bayesian posterior on the weights:

$$\theta^* = \arg\min_{\theta} KL[q(\mathbf{w}|\theta)||P(\mathbf{w}|\mathcal{D})] \tag{A.241}$$

$$= \arg\min_{\theta} \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathbf{w})P(\mathcal{D}|\mathbf{w})} dw \tag{A.242}$$

$$= \arg\min_{\theta} KL[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})] \tag{A.243}$$

The cost function defined above is called the *variational free energy* or *expected lower bound*:

$$\mathcal{F}(\mathcal{D},\theta) = \underbrace{KL[q(\mathbf{w}|\theta)||P(\mathbf{w})]}_{\text{prior dependent - "complexity cost"}} - \underbrace{\mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})]}_{\text{data dependent "likelihood cost"}} \tag{A.244}$$

$$= \mathbb{E}_{q(\mathbf{w}|\theta)} \log q(\mathbf{w}|\theta) - \mathbb{E}_{q(\mathbf{w}|\theta)} \log p(\mathbf{w}) - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log p(\mathcal{D}|\mathbf{w})] \tag{A.245}$$

where all three terms are expetations w.r.t. the variational distribution $q(\mathbf{w}|\theta)$, thus the cost function can be approximated by drawing samples $\mathbf{w}^{(i)}$ from $q(\mathbf{w}|\theta)$.

$$\mathcal{F}(\mathcal{D},\theta) \approx \frac{1}{N}\sum_{i=1}^{N} [\log q(\mathbf{w}^{(i)}|\theta) - \log p(\mathbf{w}^{(i)}) - \log p(\mathcal{D}|\mathbf{w}^{(i)})] \tag{A.246}$$

Using a Gaussian variational posterior $\theta = (\mu, \sigma)$, with μ mean vector and σ standard variation vector (whose elements are from the diagonal covariance matrix, since weights are assumed to be uncorrelated). The NN thus will be parametrized by θ instead of $\mathbf{w}$.

Bayes by backprop [24]The training consists of a forward and backward pass, in the forward one we draw a sample from variational posterior $q(w|\theta)$ as a Monte Carlo estimate of the expectation. This will be used to evaluate the cost function $\mathcal{F}(\mathcal{D},\theta)$ with eq A.246. Here the first two term are data independent and can be evaluated layer wise, while the last term is evaluated at the end of the fw pass. In the backward pass gradients of μ and σ are calculated.

For the backpropagation we must use the so-called *reparametrization trick* [124]to be able to learn μ and σ.

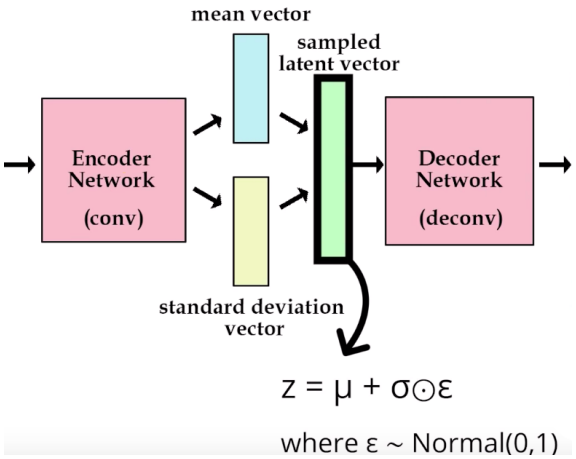


Figure A.12 semantics of VAE[2]

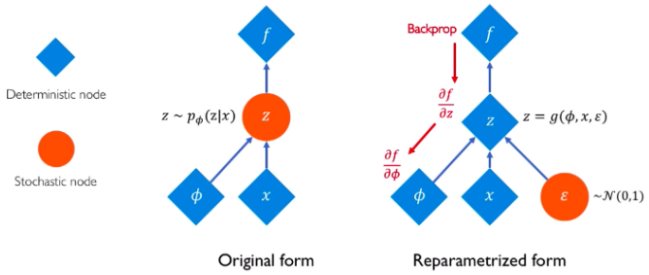


Figure A.13 VAE reparamtrization trick, with ϕ being the distribution to learn (?)[3]

For a *Gaussian variational posterior* over the weights $\mathbf{w}$ we can define ε as a sample from a standard Gaussian distribution, that will be combined by a deterministic, differentiable function g:

$$\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{A.247}$$

$$\mathbf{w} = g(\theta, \varepsilon) = \mu + \sigma \odot \varepsilon, \text{ for } \odot = \text{element-wise multiplication} \tag{A.248}$$

To ensure the standard deviation σ to be positive, it can be parametrized by ρ point-wise:

$$\sigma = \log(1 + \exp(\rho)) \tag{A.249}$$

thus the variational posterior parameters : $\theta = (\mu, \rho)$, and

$$w = \mu + \log(1 + \exp(\rho)) \odot \varepsilon \tag{A.250}$$

With

$$f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - logP(\mathbf{w})P(\mathcal{D}|\mathbf{w}) \tag{A.251}$$

This deterministic function moves the probabilistic part "out of the way" of the backward step, and thus for $g(\theta, \varepsilon) = t(\mu, \sigma, \varepsilon)$ the derivatives can be calculated: The gradients with respect to the mean μ and the standard deviation parameter ρ will be :

$$\Delta_\mu = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu} \tag{A.252}$$

$$\Delta_\rho = \underbrace{\frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\varepsilon}{1 + \exp(-\rho)}}_{(g(h(x))' = G'(h(x)) \cdot h'(x)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho} \tag{A.253}$$

The updates of the variational parameters:

$$\mu \leftarrow \mu - \alpha \Delta_\mu \tag{A.254}$$

$$\rho \leftarrow \rho - \alpha \Delta_\rho \tag{A.255}$$

The term $\frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}}$ is shared for the gradients of mean and variance parameter ρ and are exactly the gradients found by the traditional backpropagation algorithm on an NN.

Variational inference To sum up the idea of variational inference, we approximate the intractable posterior weight distribution $p(\mathbf{w}|\mathcal{D})$ with some $q_\varphi(\mathbf{w})$, thus optimization aims to find φ parametrs to minimize KL divergence $KL(q_\varphi(\mathbf{w})||p(\mathbf{w}|\mathcal{D}))$. This is equivalent to maximizing the *variational lower bound*

$$\mathcal{L}(\varphi) = -KL(q_\varphi(\mathbf{w})||p(\mathbf{w})) + L_\mathcal{D}(\varphi), \tag{A.256}$$

with denoting the *expected log-likelihood* as $L_{\mathcal{D}}(\varphi) = \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \mathbb{E}_{q_\varphi}[\log p(y|x,w)]$.

Since the (conditional) marginal log-likelihood

$$\mathcal{L}(\phi) + KL(q_\varphi(\mathbf{w})||p(\mathbf{w}|\mathcal{D})) = \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \log p(y|x), \tag{A.257}$$

which is constant w.r.t. ϕ, thus maximizing the lower bound w.r.t. ϕ will minimize $KL(q_\varphi(\mathbf{w})||p(\mathbf{w}|\mathcal{D}))$. *Stochastic gradient variational Bayes* method introduced (SGVD) in [124] is based on the reparametrization trick, that is on parametrization the randomized model parameters $w \sim q_\phi$, as $\mathbf{w} = g(\varepsilon,\phi)$, with a differentiable function g, and a random noise variable $\varepsilon \sim p(\varepsilon)$. With this *reparametrization* a differentiable,mini-batch based Monte Carlo estimator of the expected log likelihood can be formed as follows:

$$L_{\mathcal{D}}(\phi) \approx L_{\mathcal{D}}^{\text{SGVB}}(\phi) = \frac{N}{M}\sum_{i=1}^{M} \log p(y^i|x^i, w = g(\varepsilon,\phi)), \tag{A.258}$$

where $(x^i,y^i)_{i=1}^M$ is a mini-batch of data with M random data points . If the KL divergence part of the variational lower bound cannot be computed analytically to compute $D_{KL}(q_\phi(\mathbf{w})||p(\mathbf{w}))$ using MC as well Thus having an unbiased estimate the gradient $\nabla_\phi L_{\mathcal{D}}(\phi) \approx \nabla_\phi L_{\mathcal{D}}^{\text{SGVB}}(\phi)$ ill be also unbiased thus after a random initialization of ϕ, the optimization can be performed using stochastic gradient ascent on $\mathcal{L}(\phi)$.

In [123] Kingma et. al introduced a range of methods, that can make bayesian trainig even more effective, namely *local reparametrization* and *variational dropout*.

Local reparametrization trick According to the theory stochastic gradient based methods should eventually converge to some local optimum, high variance of the gradients can have a very disadvantageous effect on practical performance.

Then variance of the log likelihood in SGVB is given by

$$\mathrm{Var}[L_{\mathcal{D}}^{\mathrm{SGVB}}(\phi)] = \frac{N^2}{M^2}\left(\sum_{i=1}^{M}\mathrm{Var}[L_i] + 2\sum_{i=1}^{M}\sum_{j=i+1}^{M}\mathrm{Cov}[L_i, L_j]\right) \tag{A.259}$$

$$= N^2\left(\frac{1}{M}\mathrm{Var}[L_i] + 1\frac{M-1}{M}\mathrm{Cov}[L_i, L_j]\right), \tag{A.260}$$

Where the variances and covariances are w.r.t. the data distribution and also the noise distribution :

$$\mathrm{Var}[L_i] = \mathrm{Var}_{\varepsilon, \mathbf{x}_i, y_i}[\log p(y_i|x_i, w = g(\varepsilon, \phi))]. \tag{A.261}$$

Here as the minibatch size M grows, the dominant term will become the covariance. And since for the purpose of efficient training the forward and backward passes are executed over the whole mini-batch, for the entire minibatch ε will be sampled only once, that results in a not negligible effect on the covariance.

Thus to increase the performance of variational inference method, [123] proposes *local reparametrization*, that yields $\mathrm{Cov}[L_i, L_j] = 0$, through sampling random activations directly instead of adding noise to the weights.

Variational Dropout Dropout is regarded the most efficient regularization technique for NN training. The idea behind the method shortly is, that the strong tendency of NNs for overfitting can be reduced by randomly randomly disabling neurons, thus forcing the network to learn more general features.

Formally, for a fully connected layer with input activations forward pass with droupout is computed as follows:

$$B = (A \odot \xi)\theta. \tag{A.262}$$

Here $\mathbf{A}$ is an $M \times K$ matrix of input features,θ is the $K \times L$ weight matrix B $M \times L$ output matrix, and $\xi_{i,j} \sim p(\xi_{i,j})$ random variables. For $p(\xi_{i,j})$ [99] originally proposed a Bernoulli distribution with $1 - p$ probability, and p stands for *droupout rate*.

In [200] has been later proved that it can work with continuous noise as well, as with a Gaussian distribution, for example: $N(1,\alpha),\ \alpha = p/(1-p)$.

[123] then reinterprets this continuous noise as a variational method introducing the *varational dropout*, and also provide a way to adapt $p(\xi)$ to the data, that is to train the parameters of their dropout distributions.

In [223] Wang et. al proposes that instead of applyingA.262, activation can be directly drawn from an approximate or exact marginal distribution with good empirical results, even ignoring dependencies between neurons of $\mathbf{B}$

They argue that this Gaussian noise comes naturally from the Bayesian nature of the network. With weights $\mathbf{W}$ where the posterior distribution of the weight is given by factorized Gaussian: $q_\phi(w_{i,j}) = N(\theta_{i,j}, \alpha\theta_{i,j}^2)$

$$q_\phi(b_{m,j}|\mathbf{A}) = N(\gamma_{m,j}, \delta_{m,j}) \tag{A.263}$$

$$\text{with } \gamma_{m,j} = \sum_{i=1}^{|K|} a_{m,i}\theta_{i,j},\ \text{and } \delta_{m,j} = \alpha \sum_{i=1}^{K} a_{m,i}^2 \theta_{i,j}^2. \tag{A.264}$$

where K is the number of datapoints.

Derivation of γ and δ In original dropout $\mathbf{b^{(i)}} = \mathbf{a^{(i)}W}$, $\mathbf{a^{(i)}}$, where a column vector of activation for datapoint i.

This can be replaced by stochastic operator:

$$\mathbf{b}^{(\mathbf{i})} = (\mathbf{a}^{(i)} \odot (\mathbf{d}^{(i)}/(1-p)))\mathbf{W} \tag{A.265}$$

for a single activation: (A.266)

$$b_k^{(i)} = \sum_j W_{jk} a_j^{(i)} d_j^{(i)}/(1-p) d_j^{(i)} \sim \text{Bernoulli}(1-p),\ p \text{ dropout rate} \tag{A.267}$$

$$\mathbb{E}[d_j^{(i)}] = (1-p) \tag{A.268}$$

$$\mathbb{E}[d_j^{(i)}/(1-p)] = 1 \tag{A.269}$$

$$\text{Var}[d_j^{(i)}/(1-p)] = \text{Var}[d_j^{(i)}]/(1-p) \tag{A.270}$$

thus the expected value and variance for$\mathbf{b}^{(i)}$: (A.271)

$$\mathbb{E}[b_k^{(i)}] = \mathbb{E}\left[\sum_j W_{jk} a_j^{(i)} d_j^{(i)}/(1-p)\right] \tag{A.272}$$

$$= \sum_j W_{jk} a_j^{(i)} \mathbb{E}\left[d_j^{(i)}/(1-p)\right] \tag{A.273}$$

$$= \sum_j W_{jk} a_j^{(i)} \tag{A.274}$$

$$\text{Var}[b_k^{(i)}] = \text{Var}\left[\sum_j W_{jk} a_j^{(i)} d_j^{(i)}/(1-p)\right] \tag{A.275}$$

$$= \sum_j \text{Var}\left[W_{jk} a_j^{(i)} d_j^{(i)}/(1-p)\right] \tag{A.276}$$

$$= \sum_j W_{jk}^2 (a_j^{(i)})^2 \text{Var}\left[d_j^{(i)}/(1-p)\right] \tag{A.277}$$

$$= p/(1-p) \sum_j W_{jk}^2 (a_j^{(i)})^2 \tag{A.278}$$

$$\tag{A.279}$$

Equation A.263 then can be derived from the local reparametrization (that is sampling activations B directly).

Learning the dropout rate The posterior $q_\phi(\mathbf{W})$ can be decomposed into parameter vector θ, that corresponds to the mean ,and a multiplicative noise term that is determined by α. Any posterior over $\mathbf{W}$ with a multiplicative noise can be seen as a *dropout posterior*.

NN training maximizes expected likelihood $\mathbb{E}_{q_\alpha}[L_\mathcal{D}(\theta)]$.

From the optimization of variational lower bound:

$$\mathbb{E}_{q_\alpha}[L_\mathcal{D}(\theta)] - KL(q_\alpha(\mathbf{w}), p(\mathbf{w}))) \tag{A.280}$$

where $p(\mathbf{w})$ should be independent from θ, thus [123] argues that that prior should be log uniform: $p(\log(|w_{i,j}|)) \propto c$.

The objective in term A.280 explicitly depends on θ and α. α than can be adaptive, by maximize A.280 w.r.t. α.

A.6 Attacks on privacy

Vulnerabilities of models are strongly connected to semantics of ML process, especially in the case NNs. A strong opinion these these days that actually all NNs work as Encoders, or more or less equivalently they basically memorize the training data.

A.6.1 Poisoning

In a poisoning attack someone, who has access to the training data can bias the model, and train it to expose unexpected behaviour. This means that we alternate the data to change the behaviour of the learned model.

1. It is connected to adversarial examples,
2. The goal is to trick the model to give a given answer for specific data
3. To make it undetectable, the attacker should avoid degrading the overall performance.

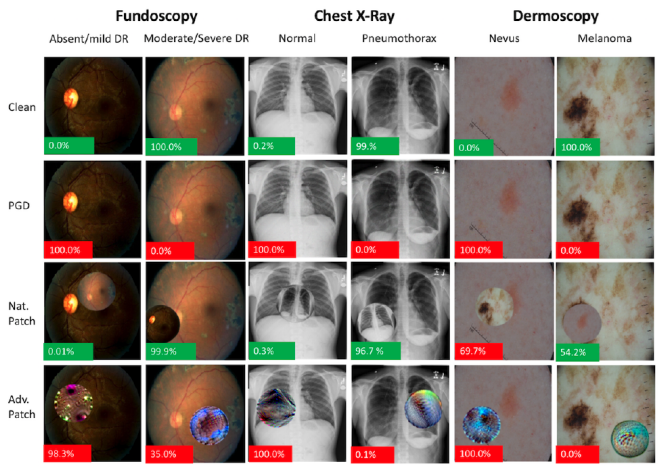


Figure A.14 Adversial example in medical imaging NN [66]

An example for this method is the pixel-wise backdoor presented in [90], that results in the canonical example of adversarial and backdoor attacks, in which a stop sign is tricked to be classified as a speed limit. This can be achieved by

- adding poisoned data during training
- modified test data during inference.

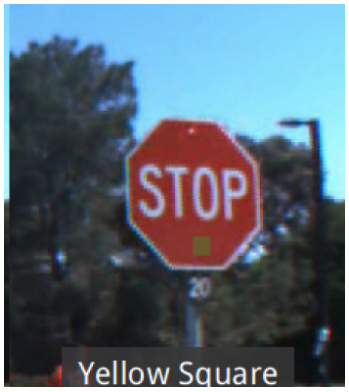


Figure A.15 Adversial example in self-driving [90].

Backdoor vs adversarial examples Adversarial transformation are applied on test examples exploiting boundaries between the different classes. Backdooring[41] shifts these boundaries intentionally to misclassify certain inputs.

Backdoor attacks in FL: Trough naive data poisoning it is hard to achieve success, since the effect might vanish by the aggregation. However [16] proposes *model replacement attacks*, in which it aims at completely replacing the global model ($\mathbf{w}_{t+1}$) with a malicious one $\tilde{\mathbf{w}}$:

$$\tilde{\mathbf{w}} = \mathbf{w}_t + \frac{\eta}{n} \sum_{k=1}^{K} (\mathbf{w}^{(k)} - \mathbf{w}_t) \tag{A.281}$$

for which it needs to solve

$$\tilde{\mathbf{w}}^m = \frac{n}{\eta}\tilde{\mathbf{w}} - \left(\frac{n}{\eta} - 1\right)\mathbf{w}_t - \sum_{\substack{k=1 \\ k \neq m}}^{K} (\mathbf{w}_{t+1}^k - \mathbf{w}_t) \approx \frac{n}{\eta}(\tilde{\mathbf{w}} - \mathbf{w}_t) + \mathbf{w}_t. \tag{A.282}$$

That is the point is to scale up the updates to survive the model averaging, moreover it is designed to be a single-shot attack.

Result:

The attacker forces the model to behave as he wants, by hiding some patterns in the input to be processed.

Defense

It is virtually impossible to detect this attack.

Summary

Even if there is no real defense, the motivation for this kind of attack would be unclear. Even if the attacker, that is the node is able to achieve the trained model to behave as it wishes, it is only true for tampered examples. The only goal might be to discredit the trained model and the method, but there is no too much chance that anyone else can learn the built-in backdoor, so it is self-revealing.

A.6.2 Data reconstruction from the gradients

Under some circumstances from the updates an adversary can reconstruct the data.

Single data-point For single gradients a method is introduced in [11][10]. The update in SGD training of NNs works in the following way (**w** parameter vector including bias vector **b**):

$$\mathbf{w} = \mathbf{w} - \eta \nabla_{\mathbf{w}} f \tag{A.283}$$

For a single neuron classifier with $\mathbf{w}_k$ denoting the weights for the kth input feature and b the bias: (Here for squared loss, but works for cross-entropy as well)

$$\Delta \mathbf{w}_k = \frac{\partial f(\mathbf{W}, b, \mathbf{x}, y)}{\partial \mathbf{w}_k} = \frac{\partial \|\phi(\mathbf{W}\mathbf{x}+b) - y\|^2}{\partial \mathbf{w}_k} = 2(\phi(\mathbf{W}\mathbf{x}+b) - y)\phi'(\mathbf{W}\mathbf{x}+b)) \cdot x_k \tag{A.284}$$

$$\text{Where } \frac{\partial \mathbf{W}\mathbf{x}+b}{\partial \mathbf{w}_k} = \frac{\partial \sum_{i=1}^{d} \mathbf{w}_i^T \mathbf{x}_i + b}{\partial \mathbf{w}_k} = x_k \text{ for the last term} \tag{A.285}$$

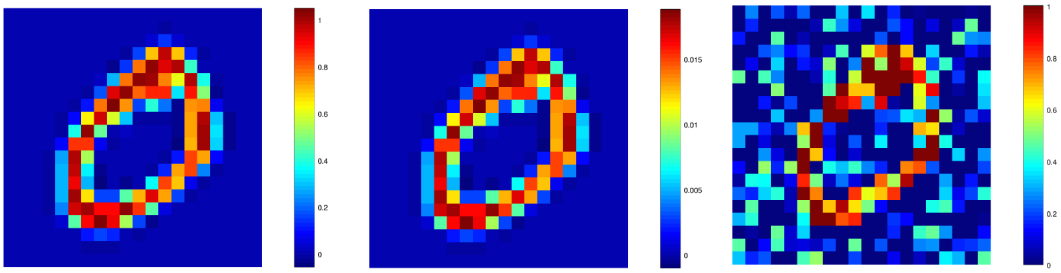
$$\Delta b = \frac{\partial f(\mathbf{W}, b, \mathbf{x}, y)}{\partial b} = \frac{\partial \|\phi(\mathbf{W}\mathbf{x}+b) - y\|^2}{\partial b} = 2(\phi(\mathbf{W}\mathbf{x}+b) - y)\phi'(\mathbf{W}\mathbf{x}+b) \cdot 1 \tag{A.286}$$

Thus one can notice, that dividing these two equations gives us back the value of the input at coordinate k: $x_k = \Delta \mathbf{w}_k / \Delta b$.

This reconstruction method has been tested for single layer fully connected NNs (Figure A.16 left image), and proven to be working, even if only part of the gradients is known by the server(middle image), or with weight regularized loss(rightmost image).

On the other hand this method cannot be used for CNNs given the number of features are much higher than the number of features

Deep leakage from Gradient (DLG) [255](and improved deep leakage, iDLG[250]) uses a different method, namely they run optimization on the pixels of a randomly generated image (or text) matching the gradients on the random image to those of the real data point. Denoting the NN with $m(\mathbf{w})$, the gradient obtained by backpropagation from the loss



(**a**) Original 20x20 image of handwritten number 0, seen as a vector over $\mathbb{R}^{400}$ fed to a neural network.

(**b**) Recovered image using 400/10285 (3.89%) gradients (see Sect.3, Example 2). The difference with the original (**a**) is only at the value bar.

(**c**) Recovered image using 400/10285 (3.89%) gradients (see Sect.3, Example 3). There are noises but the truth label 0 can still be seen.

Figure A.16 Reconstruction from single gradient in fully connected NN [11]

function f is

$$\Delta \mathbf{w} \overset{\text{def}}{=} \frac{\partial f(m(\mathbf{x}), \mathbf{y})}{\partial \mathbf{w}} \tag{A.287}$$

To reconstruct a single data point $\mathbf{x}$ then, starting from a random input $\tilde{\mathbf{x}}$ and random target $\tilde{\mathbf{y}}$ the following method can be used:

for $t = 1, \ldots$:

1. $\Delta \tilde{\mathbf{w}} \leftarrow \frac{\partial f(m(\tilde{\mathbf{x}}), \tilde{\mathbf{y}})}{\partial \mathbf{w}}$
2. $\Delta \leftarrow \|\Delta \tilde{\mathbf{w}} - \Delta \mathbf{w}\|^2$
3. $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} - \eta \frac{\partial \Delta}{\partial \tilde{\mathbf{x}}}$
4. $\tilde{\mathbf{y}} \leftarrow \tilde{\mathbf{y}} - \eta \frac{\partial \Delta}{\partial \tilde{\mathbf{y}}}$

The efficiency can be boosted by extracting extracting the true label in a single step as it is proposed in iDLG [250]. To get the ground truth label (in case of $\mathbf{y}$ is *one hot vector*) it is enough to find the output index of that has a negative gradient.

Classification is usually trained via cross-entropy loss, where for the correct class c, and the logits $\mathbf{y} = [y_1, \ldots]$:

$$f(x, c) = -\log \frac{e^{y_c}}{\sum_j e^{y_j}}. \tag{A.288}$$

Thus

$$g_i \overset{\text{def}}{=} \frac{\partial f(x)}{\partial y_i} = -\frac{\partial \log e^{y_c} - \log \sum_j e^{y_j}}{\partial x_i} \tag{A.289}$$

$$= \begin{cases} -1 + \frac{e^{y_i}}{\sum_j e^{y_j}} & < 0, \text{ if } i = c \\ \frac{e^{y_i}}{\sum_j e^{y_j}} & > 0, \text{ otherwise.} \end{cases} \tag{A.290}$$

Although this gradients w.r.t. to the output $\mathbf{y}(=\mathbf{a}^{(L)})$ cannot be seen from the update (we only update the weights, thus they are not included in the update), we still can compute it from the gradients for the weights that lead to the output $\nabla \mathbf{w}^{(L)}$. For the the incoming weights of the ith output:

$$\nabla \mathbf{w}_i^{(L)} = \frac{\partial f(x,c)}{\partial \mathbf{w}_i^{(L)}} = \frac{\partial f(x)}{\partial y_i} \cdot \frac{\partial y_i}{\partial \mathbf{w}_i^{(L)}} \tag{A.291}$$

$$= g_i \cdot \frac{\partial (\mathbf{w}_i^{(L)} \mathbf{a}^{(L-1)} + b_i^{(L)})}{\partial \mathbf{w}_i^{(L)}} \tag{A.292}$$

$$= g_i \cdot \mathbf{a}^{(L-1)} \tag{A.293}$$

The correct class then can be identified by picking the output, whose incoming weights has a gradient vector that pushes those in a different direction than the other:

$$c = i, \text{ iff } \nabla \mathbf{w}_i^{(L)^T} \cdot \nabla \mathbf{w}_j^{(L)} \leq 0, \ \forall j \neq i. \tag{A.294}$$

Another method has been presented in [177], that works for convolutional NNs as well, but also seems like they only recover a single image. In this the authors use a GAN to recover the update from a single client at a compromised server.

Reconstruction of a batch of data The DLG method [255] also reported to be working (the label guess of iDLG is not yet), when an aggregated gradient is given over a mini-batch

of data points $\mathcal{B} = \{(\mathbf{x}_1, \mathbf{y}_1, \dots\}$:

$$\Delta \mathbf{w} \overset{\text{def}}{=} \sum_{i=1}^{b} \frac{\partial f(m(\mathbf{x}_i), \mathbf{y}_i)}{\partial \mathbf{w}} \tag{A.295}$$

In this case the method is modified to optimize a single data-point in each step: for $t = 1, \dots$:

1. $j = t \mod b$
2. $\Delta \tilde{\mathbf{w}} \leftarrow \sum_{i=1}^{b} \frac{\partial f(m(\tilde{\mathbf{x}}_i), \tilde{\mathbf{y}}_i)}{\partial \mathbf{w}}$
3. $\Delta \leftarrow \|\Delta \tilde{\mathbf{w}} - \Delta \mathbf{w}\|^2$
4. $\tilde{\mathbf{x}}_j \leftarrow \tilde{\mathbf{x}}_j - \eta \frac{\partial \Delta}{\partial \tilde{\mathbf{x}}_j}$
5. $\tilde{\mathbf{y}}_j \leftarrow \tilde{\mathbf{y}}_j - \eta \frac{\partial \Delta}{\partial \tilde{\mathbf{y}}_j}$

The result of these method is illustrated in Figure A.17.

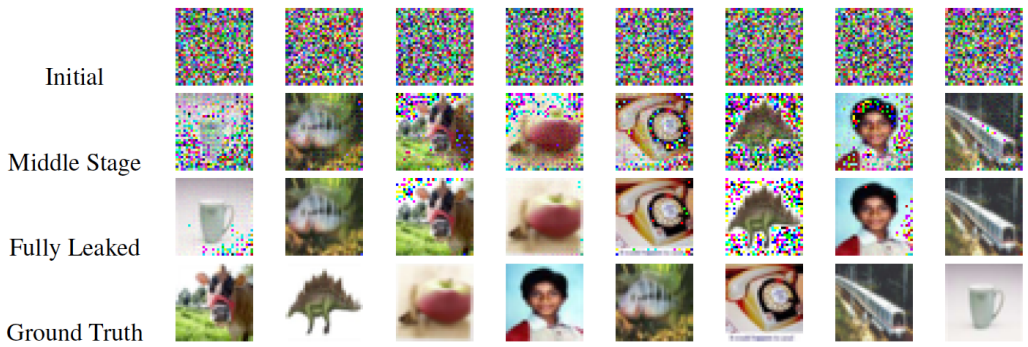


Figure A.17 Batch leakage [250]

In [76] the authors analyse the above methods with respect to their usability in different learning scenarios, and also make the following observations:

- The methods in [11][10] have limits for general NNs (few layers, non CNN etc)
- DLG is reported in the paper to be working for batch sizes of 8 and resolution of 64×64.

- For p number parameters and d input pixel if $p < d$ reconstruction is as complex as image recovery from incomplete data. Even if $p > d$ non-linearities cause problems
- For fully connected layers : input for those can always be computed analytically (assuming that grad $\neq 0$)

The main contribution of the work instead of euclidean matching $\arg\min_x \|\nabla_\theta f_\theta(x,y) - \nabla_\theta f_\theta(x^*,y)\|^2$. Thus they decompose a parameter gradients into norm magnitude and direction , magnitude captures the phase of training, how far we are from the optimum

[76] proposes the cost function for the optimizing the distance of the gradient to use cos distance, adding total variance[183]($TV(x)$) as prior on the image:

$$d(x,y) = \frac{\langle x,y\rangle}{\|x\|\|y\|} \tag{A.296}$$

Thus the objective, that aims at finding images that lead to similar change in model prediction as an unobserved ground truth image(this is equivalent to euclidean distance if the magnitude of both gradient is normalized to 1):

$$\arg\min_{x\in[0,1]^p} 1 - \frac{\langle \nabla_\theta f_\theta(x,y), \nabla_\theta f_\theta(x^*,y)\rangle}{\|\nabla_\theta f_\theta(x,y)\|\|\nabla_\theta f_\theta(x^*,y)\|} + \alpha TV(x) \tag{A.297}$$

This attack has no restriction on architecture, and works across multiple epoch training, or local gradient averaging up to 100 images. The method also works on trained networks as well where previous approaches might fail due to low magnitude of the gradients

Even if the averaged gradients are composed into update vector after multiple epochs, they still experience some leakage.

Thus to sum up both the most secure way is to mask updates completely - **secure aggregation**

Result:

If the training uses frequent updates computed from gradients over few data points, and/or small epoch number the **training data can be reconstructed completely**.

Defense

1. *secure aggregation* or *Homomorphic encryption* (LWF, Pailler,[11]) to obfuscate the distinct updates
2. *differential privacy* to change appearance of training data in the update
3. multiple local epochs

.

Summary

Gradient recovery methods aim at completely reconstructing the training data, thus these can be considered the most harmful attacks in an FL system. The methods we found up to know in the literature promise a rather limited success in a real federated learning scenario (long local training at the nodes, multiple pass over the training data for each update), the possibility of developing more powerful attacks in the future makes it necessary to address the problem. The good new is in our intuition, the existing countermeasures as secure aggregation and deferentially private transformations seem to be sufficient to handle the problem.

A.6.3 Membership inference - Black box

[114] presents an analysis for inferring presence of records in the training data in realistic scenarios. The most important works that they have collected can be grouped into five group:

1. Membership inference [197][150][185][238]

2. Attribute inference [68],[67][238]

3. Property inference [13][70]

4. Model stealing [151] [211]

5. Hyperparameter stealing [219] [233]

All these attacks are connected to the definition of differential privacy, since they build on differences in the work of models that either used ore not used specific data points in the training.

In general evaluation of the attacks are building on the privacy budget of the learning algorithms, thus application of deferentially private transformation can be understood as balancing between effectiveness of membership attacks, and usability of final model.

We found two important ideas for the attacks:

- The first general membership attack is presented in [197]. The attack trains shadow models that are similar to the attacked one, that is performing a similar task over a similar data-sets. These models are fed by different input data, and over their outputs a binary classifier attack model will be trained whose task is to decide whether the input as part of the training data or not. (Figure A.18)

- The method of [238] simplifies this method at a great extent. Omitting the shadow and attack model, the decision is made based on the expected value of confidence over the data points.

Membership inference In a Membership Inference Attack ($\texttt{Exp}(\texttt{Att}, A, n, \mathcal{D})$) (def in [238]) the adversary is given a data point $z = (x, y)$ and the task to decide, whether it has been included in the training data ($z \in S$) or not($z \in \mathcal{D} \setminus S$).

Formally the attack is a *membership experiment* $\texttt{Exp.Att}$ is the attack n is the size of training data S, Training algorithm A that produces model $a = A(S)$. The attack proceeds the following way:

- Sample a training dataset $S \sim \mathcal{D}^n$

- choose a $b \sin 0,1$ uniformly random
- Draw a z
  - $z \sim S$ if $b = 0$
  - $z \sim \mathcal{D}$ if $b = 1$
- if $\texttt{Att}(z,a,n,A,\mathcal{D}) = b$ return 1(success), otherwise 0

Membership advantage is the way to measure the performance of `Att`:

$$\texttt{Adv}(\texttt{Att},A,n,\mathcal{D}) = 2.\Pr(\texttt{Exp}(\texttt{Att},A,n\mathcal{D}) = 1) - 1, \tag{A.298}$$

that goes form 0(0.5 `Exp` success with random coin flip) to 1 total success.

[113] presents a comparison of performance of different MIAs under various DP mechanisms, while [19] gives theoretical upper bound on the success an adversary can achieve in MIA.

Uncertainties in performance evaluation

As [105] points out in many case the guarantees and empirical performance measurements are not sufficient. The theoretical upper bounds of this effectiveness given by differential privacy and empirical or theoretical lower bounds of the attacks are building on the very artificial setup, that is implicitly present in A.298 :

- In this scheme z that has been drawn from $\mathcal{D}$ can also be part of S. Other works use therefore $z \sim \mathcal{D} \setminus S$ for $b = 1$ [196]
- Assumption of independence of records in the input, which is unrealistic, thus privacy levels do not give realistic evaluation of risks [144][6]. When samples have dependency leaking one reveals information on the other one [214].
- [114] more realistic scenario, unbalanced case: $\Pr(b = 0) \neq \Pr(b = 1)$
- The attacker has access to the same data distribution.

These assumptions leads to the theoretical bounds uncertain.

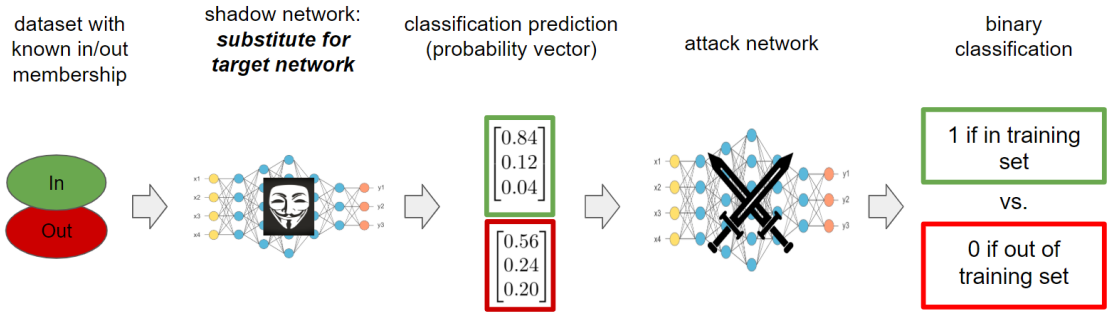


Figure A.18 membership inference with shadow models [197][4]

Missing attributes

The essence of attribute inference attack [68][238] is to use the data distribution, that is modelled by our ML model to infer the most probable values of the missing attributes $\mathbf{x}'$, conditioned on the known attributes $\mathbf{x}$:

$$\arg\max_{\mathbf{x}'} \Pr_m(\mathbf{x}'|\mathbf{x}) = \arg\min_{\mathbf{x}',\mathbf{y}} f((\mathbf{x}',\mathbf{x}),\mathbf{y}). \tag{A.299}$$

This method might be even more powerful, if we already know that someone belongs to the training data. Intuitively since the loss has been minimized on the given example, we can be almost certain that the attribute that minimizes the loss of the model is the true value. (If we do not know that, we still can believe that the input with the predicted missing attribute can be some outlier with respect to the model distribution)

Specifically in the field of medicine, one of the first application of the attack presented in [68] shows, that one could identify "genetic markers based on warfarin dosage output". Building on white-box information [229], it is possible to achieve more challenging attacks as well such as recovering faces from training data[67].

Result

Successful MIAs attacks might lead to leakage sensitive data of the patients.

Defense

The way to reduce the success rate of the attack is the blurring out the contribution of individual data points with *Differentially private transformations*

Summary

These attacks on one hand might reveal some users being included in the training data set. This, in combination with a data catalog that allows detailed screening of data on its own might lead to leakage of sensitive information. With a bit of simplification:

1. filter the dataset using some medical conditions
2. check whether the person of interest is in the training data

Attribute inference also can be used for detecting sensitive data. We can assume that someone is included in the training set, then take the following steps:

1. Publicly available data - construct a base vector
2. (Filtering conditions - add to the base vector)
3. Remaining - possibly using the prediction vector as well - use the optimization to maximize the probability that the constructed record was in the training data.

The countermeasure we can take is applying deferentially private transformations during the training, that usually promises some guarantees w.r.t maximal success rate of the attacks. These theoretical guarantees are building however on simplifying assumptions, that can change actual success rate in both directions

A.7 Predecessors of FedAvg – Distributed Learning for Convex Problems

As the amount of data to be processed exceeded the storage capacities (or memory) of the most powerful machines, new training methods have been developed to support parallel

processing. These *data parallel* methods focus on efficiency of communication of training, that aims at minimizing overall empirical loss across the training data for convex loss functions. The one extremity of distributed training is one-shot averaging [245] where the local sub-problems are solved perfectly, then the global optimum is given by a single average step. The other extreme is parallel SGD [256] where, after every single per-data-point optimization step, the local updates are averaged. In [192], the performance and the fundamental limitations of SGD-based methods have been studied in terms of runtime, communication costs and number of samples used. They found that the best convergence guaranties can be given for accelerated gradient descent [164] with the biggest possible batch sizes. Over the family of SGD-based methods, under some conditions on the problem (mostly convexity of the loss), a range of other methods based on duality or curvature information can be used as well.

In general this class of algorithms is aiming at training better models in fewer synchronisation steps, varying the local solvers and aggregation methods, exploiting the convexity of the loss function.

A.7.1 Distributed Approximate Newton (DANE)

DANE [193] solves locally available general sub-problems exactly to minimize communication needs, similarly to one-shot averaging. This method, in base case, can only work if the local sub-problems are the same, in which case, naturally, we would not need distributed training. On the other hand, if this is not the case, then nothing guarantees that the global optimum will be the average of the local solutions. The actual task for workers, therefore, is to solve a problem in each iteration which is perturbed by a quadratic term of the form $-(\mathbf{a}_t^k)^T\mathbf{w} + \frac{\mu}{2}\|\mathbf{w}-\mathbf{w}_t\|^2$. Thus the optimization follows the following steps in each iteration:

$$\mathbf{w}_{t+1}^k = \arg\min_{\mathbf{w}} f^k(\mathbf{w}) - (\mathbf{a}_t^k)^T\mathbf{w} + \frac{\mu}{2}\|\mathbf{w}-\mathbf{w}_t\|^2 \tag{A.300}$$

$$\mathbf{w}_{t+1} = \frac{1}{K}\sum_{k=1}^{K}\mathbf{w}_{t+1}^k \tag{A.301}$$

Here, the idea is to choose $\mathbf{a}_t^k$ such that $\mathbf{w}^*$, the optimal parameter value of the original task, should be also the solution of the perturbed local problem:

$$\nabla f^k(\mathbf{w}) - \mathbf{a}_t^k + \mu(\mathbf{w} - \mathbf{w}_t) = 0 \tag{A.302}$$

Thus, the perturbation ideally would be the vector that points from the global optimum $\mathbf{w}^*$ to the optimum of f^k, i.e.

$$\mathbf{a}_t^k = \nabla f^k(\mathbf{w}^*) + \mu(\mathbf{w}^* - \mathbf{w}_t) \approx \nabla f^k(\mathbf{w}^*). \tag{A.303}$$

Since the actual $\mathbf{w}^*$ is unknown, the authors suggest, instead, to use

$$\mathbf{a}_t^k = \nabla f^k(\mathbf{w}_t) - \eta \nabla f(\mathbf{w}_t), \tag{A.304}$$

where $\nabla f(\mathbf{w}_t) = \sum_{k=1}^K \nabla f^k(\mathbf{w}_t)$. The local problem, thus, becomes

$$\mathbf{w}^k = \arg\min_{\mathbf{w}} \{ f^k(\mathbf{w}) - (\nabla f^k(\mathbf{w}_t) - \eta \nabla f(\mathbf{w}_t))^T \mathbf{w} + \frac{\mu}{2} ||\mathbf{w} - \mathbf{w}_t||^2 \}. \tag{A.305}$$

By adding the term $f^k(\mathbf{w}_{t-1}) + \langle \nabla f^k(\mathbf{w}_{t-1}), \mathbf{w}_{t-1} \rangle$, that does not affect the optimization since it does not depend on $\mathbf{w}$, the local objective in Equation A.305 can be rewritten as

$$\mathbf{w}_t^k = \arg\min_{\mathbf{w}} \{ f(\mathbf{w}_{t-1}) + \langle \nabla f(\mathbf{w}_{t-1}), \mathbf{w} - \mathbf{w}_{t-1} \rangle + \frac{1}{\eta} D_k(\mathbf{w}, \mathbf{w}_{t-1}) \}, \tag{A.306}$$

where D_k denotes the *Bregman divergence* (A.2.5) for the regularized local objective $f^k(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w}\|^2$ given as

$$D_k(\mathbf{w}, \mathbf{w}_{t-1}) = D_{f^k}(\mathbf{w}, \mathbf{w}_{t-1}) + \frac{\mu}{2} ||\mathbf{w} - \mathbf{w}_{t-1}||^2 \tag{A.307}$$

$$= f^k(\mathbf{w}) - f^k(\mathbf{w}_{t-1}) - \langle \nabla f^k(\mathbf{w}_{t-1}), \mathbf{w} - \mathbf{w}_{t-1} \rangle + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}_{t-1}\|^2. \tag{A.308}$$

The first two terms in the Equation A.306 relate to a linear approximation of the loss function around the current solution $\mathbf{w}_{t-1}$ that is common to all machines (does not depend on k) and the difference between the nodes is in the local loss function f^k. This update is, in fact, a *mirror-descent* step (see Section A.2.5 and the Figure A.19) that minimizes in each round the estimation of $\mathbf{w}$ based on the local data and the current common estimate of the loss function.

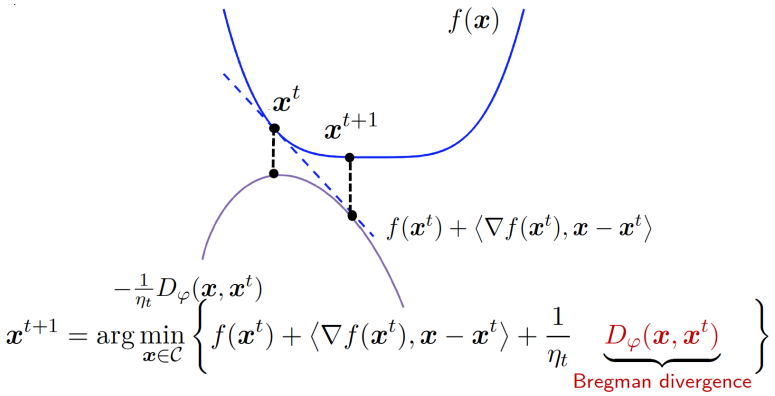


Figure A.19 Mirror descent with Bregman divergence from [43](the upper indices t denotes the iteration counter) for the optimization of function f w.r.t. variable $\mathbf{x}$

For $\eta, \mu \to \infty$ the update becomes a standard gradient descent with learning rate $\tilde{\eta} = \frac{\eta}{\mu}$. While for $\eta = 1$, if the local objectives are equal, i.e. $f^k(\mathbf{w}) = f(\mathbf{w})$, then $\mathbf{w}_{t+1}^k = \arg\min_{\mathbf{w}} f^k(\mathbf{w}) = \arg\min_{\mathbf{w}} f(\mathbf{w}) = \mathbf{w}^*$. That is, DANE converges in a single, Newton-type iteration to the empirical minimum.

DANE has a number of drawbacks which make it hardly applicable for settings of FL, though. It assumes that minimization of the local objective (Equation A.305) should be exact. It also assumes the access to iid data. Moreover, according to [193], for large number of nodes K it requires using strong regularization constant μ that involves significantly slower convergence.

A.7.2 Distributed Optimization for Self-Concordant Empirical Loss (DiSCO)

In [247], authors are aiming at exploiting the super-linear convergence of Newton methods (in the form of implementing inexact damped Newton method for distributed setting). An advantage of applying Newton methods is that these have much weaker dependence on condition number of the Hessian ($\kappa = \frac{\mu_{\max}}{\mu_{\max}}$ for μ eigenvalues), in other words, on how different is the sensitivity of the function for small changes in the input in different directions. In exact Newton methods the update is given by $\frac{\nabla f(\mathbf{w}_t)}{\nabla^2 f(\mathbf{w}_t)} \stackrel{\text{def}}{=} \Delta \mathbf{w}_t$. For the distributed problem the obvious implementation would be to compute the update as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \left(\sum_{k=1}^{K} \nabla^2 f^{(k)}(\mathbf{w}_t) \right)^{-1} \left(\sum_{k=1}^{K} \nabla f^{(k)}(\mathbf{w}_t) \right) \tag{A.309}$$

However, in this case the communication of Hessians in each round can involve a prohibitively large communication cost.

The DiSCO algorithm [247] bypasses this problem by using a distributed inexact damped Newton method where in each iteration t, given a non-negative sequence of expected precision $\{\varepsilon_t\}$, the task is

- to find an approximate update direction $\mathbf{v}_t$, such that $\|\nabla^2 f(\mathbf{w}_t)\mathbf{v}_t - \nabla f(\mathbf{w}_t)\| \leq \varepsilon_t$
- Compute $\delta_t = \sqrt{\mathbf{v}_t^T \nabla^2 f(\mathbf{w}_t)\mathbf{v}_t}$
- Update to approximated optima $\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{1+\delta_t}\mathbf{v}_t$

Thus, the key of implementing distributed inexact Newton methods is to find the direction of the inexact Newton step $\mathbf{v}_t$ for which we need the gradient $\nabla f(\mathbf{w}_t)$ and Hessian $\nabla^2 f(\mathbf{w}_t)$:

$$||\nabla^2 f(\mathbf{w}_t)\mathbf{v}_t - \nabla f(\mathbf{w}_t)||_2 \leq \varepsilon_t, \tag{A.310}$$

That is equivalent to solve approximately $\nabla^2 f(\mathbf{w}_t)\mathbf{v}_t = \nabla f(\mathbf{w}_t)$ for which [247] uses a distributed version of *Preconditioned Conjugate Gradient Descent* (PCG) (See Sections

A.2.4, A.2.4, A.2.6 and A.2.4). PCG achieves better convergence rate through (1) modifying the shape of the function to optimize (preconditioning, introduced in Section A.2.4) and (2) finding better update directions (conjugate gradient, introduced in Section A.2.4). The method can be very fast[5] if the data is distributed in iid fashion and, then, communicates only first order information (thus, the update vector has a size of only $\mathcal{O}(d)$)).

To achieve this, in the outer loop, the coordinator executes the PCG method starting from the average of the exact solutions of the local regularized problems

$$\mathbf{w}_0 = \frac{1}{m}\sum_{k=1}^{K}\mathbf{w}_0^{(k)} \tag{A.311}$$

$$\mathbf{w}_0^{(k)} = \arg\min_{\mathbf{w}}\left\{f^k(\mathbf{w}) + \frac{\rho}{2}\|\mathbf{w}\|_2^2\right\} \tag{A.312}$$

Based on this, the coordinator computes the *preconditioner* $\mathbf{P} = \mathbf{H}_1 + \mu\mathbf{I} = \nabla^2 f_1(\mathbf{w}) + \mu\mathbf{I}$, that is the Hessian of the loss at the first node in the starting point. Using preconditioner $\mathbf{P}$, an initial direction is computed as $v_0 = \mathbf{P}^{-1}\nabla f^1(\mathbf{w}_0)$.

In each round t the direction $\mathbf{v}_t$, along with the recent parameters $\mathbf{w}_t$, is broadcasted to the nodes which compute the exact values for $\nabla f^k(\mathbf{w}_t)$ and $\nabla^2 f^k(\mathbf{w}_t)\mathbf{v}_t$ is sent back to the coordinator. Then, the server computes the global product $\nabla^2 l(\mathbf{w}_t)v_t = \sum_{k=1}^{K}\nabla^2 f^k(\mathbf{w}_t)v_t$, together with $\mathbf{w}_{t+1}$ and $\mathbf{v}_{t+1}$, using PCG and starts the next loop until having a good enough solution.

A.7.3 Accelerated Inexact Dane (AIDE)

An inexact version of DANE and its accelerated variant are presented in [174] where the local sub-problem should be solved only to fulfill $||\mathbf{w}_t^{(k)} - \hat{\mathbf{w}}_t^{(k)}|| \leq \gamma\|\mathbf{w}_{t-1} - \hat{\mathbf{w}}_t^{(k)}\|$, denoting by $\mathbf{w}_t^{(k)}$ an approximate solution of the local problem (Equation A.305) and by $\mathbf{w}_t^{(k)}$ the exact solution with $\mathbf{w}_t = \frac{1}{K}\sum_{k=1}^{K}\mathbf{w}_t^{(k)}$. To solve the local problem one can use for instance Stochastic Variance Reduced Gradient method (SVRG, see section A.2).

[5] They propose to use DiSCO for self-concordant loss functions. An $f : \mathbb{R}^n \to \mathbb{R}$ function is said to be self-concordant [166] for means that the third derivative is controlled by the second derivative (for example linear regression).

A.7.4 Communication Efficient Distributed Coordinate Ascent (CoCoA)

All of the above presented methods build on "iid-ness", while CoCoA provably converges over any distribution.

The main idea of *CoCoA* [112] and CoCoA$^+$ [153] is that if the training examples are dispersed across multiple worker nodes, then applying dual optimization is a natural choice since, in this setting, the different nodes are working on different subsets of dual variables and the weight update $\Delta\mathbf{w}$ is eventually a linear combination $X^T\Delta\alpha$ of the refinements $\Delta\alpha_i$ of coefficients $\alpha_\mathbf{i}$ belonging to data points $\mathbf{x}_i$ (Appendix A.2.1). Thus, CoCoA is built on Stochastic Coordinate Ascent, that is applying Randomized Coordinate Ascent (Section A.2.2) [190] on the Fenchel dual space (Sections A.2.1 and A.2.1). The original problem is to minimize a convex loss function on linear predictors with $r(\mathbf{w})$ convex regularization term

$$\arg\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{X}\mathbf{w}) + r(\mathbf{w}) \tag{A.313}$$

can be turned, by introducing equality constraints, into

$$\arg\min_{\mathbf{v}=\mathbf{X}\mathbf{w}} f(\mathbf{v}) + r(\mathbf{w}), \tag{A.314}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a convex loss function. The dual of Equation A.314 will take a special form that makes optimization easier (see Section A.2.1 for details)

$$\arg\max_{\alpha\in\mathbb{R}^n} D(\alpha), \text{where} \tag{A.315}$$

$$D(\alpha) = f_i^*(-\alpha) - r^*(\mathbf{X}^T\alpha), \tag{A.316}$$

That is, we can maximize easier the negative convex conjugates of the original problem.The convex conjugate of f is $f_i^* : \mathbb{R} \to \mathbb{R}$, standing for disjoint sets of α_i dual variables

$$f_i^*(\alpha) = \max_{\mathbf{w}} \{\mathbf{w}\alpha - f(\mathbf{w}, \mathbf{x}_i)\}. \tag{A.317}$$

To find the actual convex conjugate (Fenchel dual) function we have to express α from $\mathbf{w}$ by finding the maximum of the Equation A.317 in $\mathbf{w}$ of the (always) concave conjugate function

$$\nabla_{\mathbf{w}} (\mathbf{w}\alpha - f(\mathbf{w}, \mathbf{x}_i)) = 0 \Rightarrow \alpha = \nabla_{\mathbf{w}} f(\mathbf{w}, \mathbf{x}_i), \tag{A.318}$$

and, then, plug it back into f. Thus, the original minimization task with L2 regularization $(r(\mathbf{w}) = \|\mathbf{w}\|^2)$

$$\arg\min_{\mathbf{w} \in \mathbb{R}^d} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^{n} f_i(\mathbf{w}^T \mathbf{x}_i) \tag{A.319}$$

will be equivalent to maximizing the dual

$$\arg\max_{\alpha \in \mathbb{R}^n} -\frac{\lambda}{2} \|\mathbf{X}^T \alpha\|^2 - \sum_{i=1}^{n} f_i^*(\alpha_i). \tag{A.320}$$

From the first term comes a convenient mapping $\mathbf{w}(\alpha) = \mathbf{X}^T \alpha$ due to the optimality conditions [190]. This mapping shows that each coordinate of the dual vector α corresponds to a single data point.

The main idea of CoCoA is that each worker node k optimizes disjoint sets of (dual) variables $\alpha^{[k]}$, $\alpha = (\alpha^{[1]} \dots \alpha^{[K]}) \in \mathbb{R}^n$, producing an update vector $\Delta \mathbf{w}_{t+1}^k = \mathbf{X}^{[k]} \Delta \alpha^{(k)} = \mathbf{X}^T \alpha_t - \mathbf{X}^T (\alpha_t + \Delta \alpha_t^{(k)})$, after the node k has found the exact solution $\alpha_t^{[k]} = \alpha_{t-1}^{[k]} + \Delta \alpha_t^{(k)}$ by arbitrary solver for on the local dual sub-problem over the available data set $X^{[k]}$. Then various $\Delta \mathbf{w}_t^k$ coming from different machines k will be aggregated and redistributed, i.e. $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \sum_{i=1}^{K} \Delta \mathbf{w}_t^k$.

This is expected to reduce conflicts between the updates belonging to the same machine but allows to use any local solver for the local sub-problem.

Other advantages of working with this dual problem, over the primal, is that solving the problem for one coordinate completely makes the learning rate unnecessary and it has a clear stopping criterion stemming from *duality gap* $P(\mathbf{w}(\alpha)) - D(\alpha) < \varepsilon$.

That gives indeed stronger convergence properties as it has been shown in [112], using SDCA as internal optimizer. However, in practice, for more complex problems (not necessarily convex) as [130] reports, it does not perform very well.

A.7.5 Federated SVRG: The hybrid of CoCoA, DANE and SVRG (FSVRG)

The connections of DANE with SVRG have been analysed in [174] where the authors have shown that a modified version of DANE is in fact equivalent to a distributed version of SVRG. The key idea here is to use the SVRG update loop in the inexact DANE to solve the local problems approximately. Applying the stochastic update rule of SVRG actually proceeds in direction of the local solution of DANE for $\mu = 0$ proximity regularization:

- DANE local objective (repeating the Equation A.305)

$$\mathbf{w}_k = \arg\min_{\mathbf{w}} \{ \underbrace{f_i(\mathbf{w})}_{f^k(\mathbf{w})} - (\overbrace{\underbrace{\nabla f_i(\mathbf{w}_t)}_{\nabla f_i(\tilde{\mathbf{w}})} - \underbrace{\eta}_{1} \underbrace{\nabla f(\mathbf{w}_t)}_{\nabla f(\tilde{\mathbf{w}})}}^{\text{constant w.r.t. } \mathbf{w}})^T w + \underbrace{\frac{\mu}{2} ||\mathbf{w} - \mathbf{w}_t||^2}_{=0} \} \tag{A.321}$$

- For $\mu = 0$ and $\eta = 1$, renaming the actual global optimum $\mathbf{w}_t$ to $\tilde{\mathbf{w}}$ at data point i, a step of the steepest descent algorithm for the DANE local objective is

$$-(\nabla f_i(\mathbf{w}) - \nabla f_i(\tilde{\mathbf{w}}) + \nabla f(\tilde{\mathbf{w}})) \tag{A.322}$$

- This vector happens to be the stochastic update direction computed in the inner loop of SVRG for the ith single data point

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta_t \underbrace{(\nabla f_{i_t}(\mathbf{w}_{t-1}) - \nabla f_{i_t}(\tilde{\mathbf{w}}_r) + \tilde{g})}_{\text{update direction}} \text{ (Equation A.31 in Appendix A.2),} \tag{A.323}$$

since the actual global optimal value $\tilde{\mathbf{w}}$ in the SVRG stochastic local update corresponds to $\mathbf{w}_t$ in the formulation of the DANE local solver, for the full gradient $\nabla f(\mathbf{w}_t) = \tilde{g}$, $\nabla f^k(\mathbf{w}_t) = \nabla f_i(\mathbf{w}_t)$ and $\nabla f(\mathbf{w}_t) = \nabla f(\tilde{\mathbf{w}})$

Moreover, as pointed out in [130], DANE with SVRG style local updates can also be interpreted as applying the idea of CoCoA to fix the drawbacks of DANE, namely, solving the local sub-problems only to obtain a *relative accuracy*.

Based on these observations, authors of [130], who originally formulated the setup of FL, propose *Federated SVRG* for solving the linear, L2 regularized problems (that is, in the form $\min_{\mathbf{w}} f(\mathbf{X}\mathbf{w}) + \|\mathbf{w}\|^2$). In their method to such hybridization of DANE, SVRG and CoCoA they also add

- Adjusting local step size to have similar scale of local update vectors: $\eta_k = \eta / n^k$;
- Weighting the updates based on the proportion of data: $\frac{n^k}{n}(\mathbf{w}^k_{t+1} - \mathbf{w}_t)$;
- Weighting the updates by a diagonal matrix $\mathbf{S}^k$ where the elements of the diagonal are given by the proportion of local and global frequencies of non zeros at the given coordinates of the input data;
- Weighting updates by a diagonal matrix $\mathbf{A}$ where the jth diagonal elements are given by the inverse of the proportion of nodes, where the given coordinates appear with non-zero value.

The method has been tested for predicting whether a given Google+ post will generate comments, using regularized logistic regression. For this problem the Federated SVRG highly outperformed CoCoA+ and distributed Gradient Descent, however, the method

Algorithm 8 Federated Stochastic Variance Reduced Gradient

procedure FSVRG(step size η, data partition $\{\mathcal{P}_k\}_{k=1}^K$, diagonal matrices $\mathbf{A}$, $\mathbf{S}_k \in \mathbb{R}^{d\times d}$ for $k \in \{1,...,K\}$)
  for $s = 0,1,2,...$ **do** ▷ Overall iterations
    Compute $\nabla f(\mathbf{w}_t) = \frac{1}{n}\sum_{i=1}^n \nabla f_i(\mathbf{w}_t)$
    for $k = 1$ to K **do** ▷ Distributed loop
      Initialize: $\mathbf{w}^k = \mathbf{w}_t$ and $\eta_k = h/\eta_k$
      Pick $i \in \{1,2,...n\}$, uniformly at random
      Let $\{i_t\}_{t=1}^{n^k}$ be random permutation of $\mathcal{P}_k$
      for $t = 1,...,n^k$ **do** ▷ Actual update loop
        $\mathbf{w}^k = \mathbf{w}^k - \eta_k(\mathbf{S}^k[\nabla f_{i_t}(\mathbf{w}^k) - \nabla f_{i_t}(\mathbf{w}_t)] + \nabla f(\mathbf{w}_t))$
      end for
    end for
    $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{A}\sum_{k=1}^K \frac{n^k}{n}(\mathbf{w}^k - \mathbf{w}_t)$
  end for
end procedure

is strongly tailored to the specific task by exploiting very different patterns in the data generation at the nodes by the $\mathbf{A}$ and $\mathbf{S}^k$ matrices.

Moreover, since FSVRG, such as CoCoA and other similar sophisticated methods, is building on the convexity of the objective function, the much simpler methods of FedSGD and FedAvg are used in most of the cases since these are empirically proven to be more efficient on a much bigger range of models.

A.8 Compression for FL

In [133] the authors propose an asymmetric communication pattern for reducing the communication costs for image classification tasks with NN-s. The work builds on the observation according to which internet connections are set up in an asymmetric way, that is the up-link is usually much slower then down-link, thus the main goal is to reduce the amount of data to be sent by the workers.

The paper examines two ways of compressing information to be aggregated:

1. **Structured updates** utilize the observations of [54] according to which, in case of image classification, a significant portion of model parameters can be predicted based

on the generally smooth transitions along the spatial position of input parameters. With the structured updates approach the search space is going to be restricted, that is, not all the parameters but only the parameters from lower dimension are trained:

(a) In a *low rank matrix method*, in each training round, the weight update matrix $\mathbf{H}_t$ is factorized into a projection matrix $\mathbf{B}_t^i$ and a reconstruction matrix $\mathbf{A}_t^i$ which is generated randomly in each round. At node i and at round t $\mathbf{H}_t^i = \mathbf{A}_t^i \mathbf{B}_t^i$ and only train the $\mathbf{B}_t^i$ projection matrix, while $\mathbf{A}_t^i$ can be sent to the server as a random seed.

(b) The *random mask technique* means to restrict the update $\mathbf{H}_t^i$ to be sparse according to predefined random sparsity pattern, what can be done by a random seed, so only non-zero values and the random seed have to be communicated.

2. **Sketched updates** compute the whole update $\mathbf{H}_t^i$ and then compress or sub-sample it to reduce the data to be communicated ($\hat{\mathbf{H}}_t^i$).

(a) *Random sub-sampling* [132] takes a random subset of trained parameters, then sends it to the server along with the random seed. In theory, the aggregated partial updates give an unbiased estimate of the global update direction, i.e. $\mathbb{E}[\hat{\mathbf{H}}_t] = \mathbf{H}_t$

(b) *Probabilistic quantization* assigns each parameter $h_{ij} \in \mathbf{H_t^i}$ closer to one of the boundaries of the interval it falls in, with a probability proportional to the distance from them. For a 1 bit compression it is

$$\hat{h}_{ij} = \begin{cases} h_{max}, \text{with probability} \dfrac{h_{ij} - h_{min}}{h_{max} - h_{min}} \\ h_{min}, \text{with probability} \dfrac{h_{max} - h_{ij}}{h_{max} - h_{min}} \end{cases} \tag{A.324}$$

Naturally, this scheme can be generalized to more than 1 bit, in which case representing the weights on b bits can create 2^b intervals. The quantization error can be further reduced by applying structured random rotations as is suggested in [206].

A.9 Quantization

In distributed training off NNs, the use of *Quantization methods* builds on the observation that large NN parameters are sparse, thus, the updates are, not surprisingly, sparse too. In general, they are viewed as a generalization of delayed update training, uploading the most important directions the model should be changed in. These methods usually accumulate gradient residuals for those coordinates that have not been sent, and when the aggregated magnitude grows beyond the threshold they will be updated. That means that less important parameters will be less frequently and less significantly updated, thus reducing the average update size.

A general method for quantization with residuals can be formulated as

$$\tilde{\nabla f_t} = \mathcal{Q}(\nabla f_t + \Delta f_{t-1}) \tag{A.325}$$

$$\Delta f_t = \nabla f_t - \mathcal{Q}^{-1}(\tilde{\nabla f_t}), \tag{A.326}$$

where $\tilde{\nabla f_t}$ stands for the quantized gradient at time t, $\mathcal{Q}$ for the quantization function and Δf_t denotes gradient residual/quantization error. The simplest way for specification of importance of coordinates is done simply by pruning along some thresholds τ by the magnitude of the value at the coordinate.

Seide et al. [189] presents the method of *1-bit quantization* for distributed SGD training of NNs, which is one of the simplest, yet efficient method. They found that for speech processing NNs even 1-bit quantization with $\tau = 0$ can work (that is, dropping all negative gradients, or equivalently replacing them by 0, and keeping all positive gradients replacing them by 1).The *re-quantization function*, that is, reconstruction of the update value then simply substitutes the mean of the values of its column in the weight matrix, multiplied by the sign transmitted for that given coordinate.

Threshold quantization [203] replaces $\tau = 0$ in 1-bit quantization with a magnitude threshold $\tau > 0$ and sends updates with a magnitude greater than τ. Every item of the update is a 32-bit word from which one bit is the value 1 or 0, corresponding to an update length τ or $-\tau$ and 31 bit for index in the parameter array. All the residuals, i.e. the remaining parts

(above τ or under $-\tau$) of the updated variables, and the values between τ and $-\tau$ will be accumulated per index and submitted later when threshold is reached. Thus, eventually, all the updates will be sent, only the weaker ones will be transmitted less frequently.

While with 1-bit quantization the compression is of a constant rate 32, when we assume that the original vector consists of 32-bit floats, threshold quantization results in a volatile size of the updates and brings the difficulties of picking the right threshold value. However, it incurs a relatively fast convergence.

Trying to combine the upsides of this two methods, Dryden et al. [60]) propose an algorithm to dynamically specify the values for τ, adjusting it to a fixed π proportion of number of parameters to be uploaded. According to this, if k^+ denotes the number of positive update coordinates, in each round the $\pi * k^+$ largest values , and similarly the $pi * k^-$ negative coordinates with the largest absolute values will be updated. For the reconstruction on the server side, the mean of the updated values will be used per sign. Thus, keeping the constant size of the updates, a faster convergence can be achieved.

In [4], first, the method of Dryden is modified using a single threshold on the absolute values of the layers' parameters (that is πth largest absolute values will be set to 1) and, second, applying a global threshold building on *layer normalization* [14].

Another version of information compression is proposed in [20], where in both client-server and server-client direction, simply the sum of signs over the gradients taken at the different data points $\mathbf{x}_i$ are communicated such as $\Delta\mathbf{w} = \mathbf{w} - \eta \text{sign}\left[\sum_{i=1}^{n^k} \text{sign}(\nabla f(x_i))\right]$. This majority vote like scheme can also be executed at the server at the time of aggregation and, eventually, the aggregated update $\eta \text{sign}\left[\sum_{i=1}^{K} \Delta\mathbf{w}\right]$ can be applied at the worker nodes.

A.10 Ensembles and distillation

[9] addresses the problem of very large scale distributed NN training, where the most used methods are versions of synchronous and asynchronous distributed stochastic MBGD ([51][40]). Adding more workers to the training can be utilized only for a limited extent to speed up the process by distributing computation and storage needs. After a certain level

infrastructural limits and increasing problems of divergence of local training starts hampering the process above the advantages. In the synchronous settings further improvement can be achieved by increasing effective batch sizes, that results in less biased gradient estimates [138] [120]. However, this improvement vanishes quickly with the growing number of workers. The way to go, after this point, is to multiply the whole process by training an ensemble of models. That, in exchange, leads to increasing test and inference times as well as required computation capacities. To reduce these requirements arising from the use of ensemble distillation [98] is proposed.

A safe enough way to produce high performance predictors is to train multiple high capacity predictors on the training data and create an ensemble [56] that will make predictions combining output vectors by weighted average or voting methods. This, on one hand, is a simple way of combining models but, on the other hand, it might be cumbersome to make predictions based on a big number of complicated models.

For the problem of forming large ensembles, [37] describes an efficient step-wise ensemble selection method. The ensemble creation starts from an empty set and iteratively adds models to the collection in a greedy way, that is, the model that yields the biggest improvement for the ensemble will be included in each round.

A simple and elegant method to assemble the knowledge of the ensemble, called *Distillation* has been presented in [32]. Distillation builds on the empirical fact that much smaller models may have a similar expressing power to huge NNs or ensembles. The key idea is that the new, simpler model should be trained to emit the soft probabilities (in case of qualitative NNs the *logits*, i.e. the activation of the output layer before applying output activation function) of the output of the ensemble. Soft probabilities or `logits` can carry a lot more information on the distribution than the output after applying the *softmax* – if its entropy is high enough. Exploiting that NNs are universal approximators, a logical choice would be to use them for the compressed model. To transfer the knowledge however, the method needs a sufficient amount of transfer data. This can be obtained from a validation set splitted out from training data, or alternatively, some unlabelled set of data could be used as well, for which the labels are easy to produce by the teaching ensemble.

An improved and specialized version of this method is applied in [32], while [98] presents the application of *distillation* for transfer the knowledge of single huge and "cumbersome" NNs to a smaller, simpler model. If we assume that the cumbersome model generalizes well, the knowledge the big model obtained, that is the way it generalize, can be grabbed through the output vectors produced on individual data points. Maximising the average log probability of the correct answer comes with the side effect of assigning probabilities to the incorrect labels as well, that actually carries very important information. According to the example provided in the paper, an image of a car might be misclassified for a truck, but this probability will likely be much higher than the one for mistaking it for a carrot.

Generalization using logit is introduced in [32] with using a *temperated softmax*

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}. \tag{A.327}$$

If $T = 1$, Equation A.327 returns the probability q_i for class i, as it is usually done in the final layer of an NN. The higher the T, the smoother is the distribution of probabilities over the classes. The labels for the *transfer set* should be produced with high T and the training of the distilled model should be carried out with that value as well, while at inference stage, naturally, $T = 1$ should be used.

If the correct labels are also available for the transfer set, they propose to use a combined objective function according to empirical results with a much lower weight. Each training example contributes to the cross-entropy gradient in the optimisation: with z_i logit in the distilled model and v_i logit form the big model, that gives the soft target probability

$$\frac{\partial C}{\partial z_i} = \frac{1}{T}\left(\frac{e^{z_i/T}}{\sum_j e^{z_j/T}} - \frac{e^{v_i/T}}{\sum_j e^{v_j/T}}\right),$$

That is, the goal of distillation is minimizing difference between the temperated softmax of the models. In an experiment with MNIST it was demonstrated that for simple NN the method works, distillation of the same neural network from a 1.5 times bigger one (1200 ReLU vs 800 ReLU) yielded 50% performance improvement comparing to the direct

training of the small network. *Mutual learning* [248] and *Co-distillation* [9]. are very similar techniques, aiming at preventing the pipeline complexity of two stage distillation[98] (training an ensemble or a complex teacher model, then, distillation). According to their idea, the distinction between training and distillation phase (or, between student and teacher model) is unnecessary, since one can use a distillation term in the loss of the individual models, that might be disabled at very beginning of training process. After a warming up period thus the weight updates will be

$$\mathbf{w}^{(i)} = \mathbf{w}^{(i)} - \eta \nabla_{\mathbf{w}^{(i)}} \left[f(\mathbf{w}^{(i)}, \mathbf{x}) + \psi \left(\left\{ \frac{1}{N-1} \sum_{j \neq i} m^{(j)}(\mathbf{w}^{(j)}, \mathbf{x}) \right\}, m^{(i)}(\mathbf{w}^{(i)}, \mathbf{x}) \right) \right],$$

where f is the loss of the model $m^{(i)}$ and ψ is a penalty term that penalizes the difference form the average of the logits $m(w^{(j)}, \mathbf{x})$ computed by the models $\mathbf{w}^{(j)}$, that have been trained in parallel on the same examples (or batches) $\mathbf{x}$. They empirically found that the presence of stale predictions (old parallel models) in the training process has an infinitesimal influence compared to stale gradients in asynchronous SGD.

A.11 Peer-to-peer methods

In the gossip models, that have been introduced in [215] and [18], the main task is balancing between local performance of the models at nodes, and a requirement of a kind smoothness of the parameter space. The presented methods are peer-to-peer multi-task learning like algorithms for collaboratively learn personalized models over a network of nodes. The set of nodes in these works are represented as a graph, in which the neighbourhood matrix $N \in \mathbb{R}^{n \times n}$ contains the weights of edges, that describes similarities of nodes or "task relatedness". This can be based on user profiles or data itself. For normalizing relations, a diagonal matrix D can be used with $D_{ii} = \sum_{j=1}^{n} N_{ij}$. In the algorithm, the primary objective of the nodes is to obtain a model describing their data as well as possible:

$$\mathbf{w}_{\text{sol}}^{k} = min_{\mathbf{w} \in \mathbb{R}^n} f^k(\mathbf{w}) = \sum_{j=1}^{n^k} f(\mathbf{w}; x_j^k, y_j^k) \tag{A.328}$$

Above this, the proposed systems seek to achieve a kind of *smoothness* over all the models in the sense that more related task should have more similar models.

The idea of the first approach, called **model propagation**[215] is, first, to find the exact optima at the nodes and, then, make the weights smooth over the edges. For that, to each local model $\mathbf{w}_{\text{sol}}^k$ a confidence value is assigned based on the amount of data they hold ($c_k = n^k / max_j n^j$). This controls how far the model can drift away in the function of the neighboring models. Thus the objective for model propagation algorithm is the following:

$$\Omega_{MP} = \frac{1}{2}(\sum_{i<j}^{K} N_{ij}\|\mathbf{w}^i - \mathbf{w}^j\|^2 + \mu \sum_{i=1}^{K} D_{ii}c_i\|\mathbf{w}^i - \mathbf{w}_{sol}^i\|^2), \tag{A.329}$$

where μ is a trade off parameter between the grade of smoothness and the local accuracy. This problem could be solved in closed form $\mathbf{W}^* = \overline{\alpha}(I - \overline{\alpha}(I - C) - \alpha P)^{-1} C\mathbf{W}_{\text{sol}}$ for $\mathbf{W}_{sol} = [\mathbf{w}_{\text{sol}}^1; \ldots; \mathbf{w}_{\text{sol}}^K] \in \mathbb{R}^{K \times p}$, $\alpha \in (0,1)$ such that $\mu = (1-\alpha)/\alpha$ and $\overline{\alpha} = 1 - \alpha$. To compute this closed form optimum, however, we would need to know the global network and all the solitary models. Therefore, in the peer-to-peer setup, the following two-step iterative asynchronous process is proposed (with $\tilde{\mathbf{W}}_i^j(t) \in \mathbb{R}^p$ denoting agent i's last knowledge on the other models at nodes j):

1. *communication step*: i selects a random neighbour $j \in \mathcal{N}_i$, and they exchange their models, i.e.

$$\tilde{\mathbf{W}}_i^j(t+1) = \tilde{\mathbf{W}}_j^j(t) \text{ and } \tilde{\mathbf{W}}_j^i(t+1) = \tilde{\mathbf{W}}_i^i(t) \tag{A.330}$$

.

2. *update step*: for $l \in \{i, j\}$:

$$\tilde{\mathbf{W}}_l^l = (\alpha + \overline{\alpha}c_l^{-1})(\alpha \sum_{k \in \mathcal{N}_i} \frac{N_{lk}}{D_{ll}} \tilde{\mathbf{W}}_l^k(t+1) + \overline{\alpha}c_l \mathbf{w}_{\text{sol}}^l) \tag{A.331}$$

In the **collaborative learning** algorithm [215], learning and propagation are interweaved, thus the objective

$$\mathcal{Q}^i_{CL}(\mathbf{W}_i) = \frac{1}{2}(\sum_{i<j}^{K} N_{ij}\|\theta_i - \theta_j\|^2 + \mu \sum_{i=1}^{K} D_{ii} f^i(\mathbf{w}^i), \tag{A.332}$$

where the left side term is responsible for smoothness and the right side term aims at preventing too high loss of local accuracy. For solving this, they use distributed version of *Alternating Direction Method of Multipliers* (ADMM) ([29, 225]) (Section A.2.3) on a Lagrangian formulation (Section A.2.1) of the problem:

$$L^i_\rho(\tilde{\mathbf{W}}_i, Z_i, \Lambda_i) = \mathcal{Q}^i_{CL}(\tilde{\mathbf{W}}_i) + \sum_{j\in\mathcal{N}_i} \Big[\Lambda^i_{ei}(\tilde{\mathbf{W}}^i_i - Z^i_{ei}) + \Lambda^j_{ei}(\tilde{\mathbf{W}}^j_i - Z^j_{ei}) \tag{A.333}$$

$$+ \frac{\rho}{2}(\|\tilde{\mathbf{W}}^i_i - Z^i_{ei}\|^2 + \|\tilde{\mathbf{W}}^j_i - Z^j_{ei}\|)\Big]. \tag{A.334}$$

where ρ is a penalty weight for ensuring smoothness, matrix $\tilde{\mathbf{W}}_i$ is a copy for all parameters in the neighbourhood of node i, and the secondary variables Z^i_{ei} and Z^j_{ei} can be understood as some preferences of node i for the values maintained at i itself ($\tilde{\mathbf{W}}_i$) and at j ($\tilde{\mathbf{W}}_j$). Thus, to minimize f^i_ρ the nodes have to agree on each others' parameters. The Λ^i_{ei} and Λ^j_{ei} are the dual variables associated with the constraints that require these preferences to be as close as possible while $\rho > 0$ penalizes the level of "disagreement". The gossip based ADMM proceeds by

1. i updates primal variables $\tilde{\mathbf{W}}^i_i$, $\tilde{\mathbf{W}}^j_i$ and, along with the old duals Λ^i_{ei} and Λ^j_{ei}, sends them to j.

2. Upon receiving these, j updates the preference variables Z^i_{ei} and Z^j_{ei} and sends them back to i.

3. Based on the previous steps, i finally updates the dual Λ^i_{ei} and Λ^j_{ei}.

To sum up, this gossip based collaborative method is based on enforcing a consensus on the models on each other.

A bit simpler, gradient based, method for peer-to-peer learning of personalized models has been presented by Bellet et al. [18], where the objective to minimize (with the notation used in [215]) is

$$\mathcal{Q}_{\mathrm{CL}}(\Theta) = \frac{1}{2}\sum_{i<j}^{K} N_{ij}\|\mathbf{w}^i - \mathbf{w}^j\|^2 + \mu\sum_{i=1}^{K} D_{ii}c_i f^i(\mathbf{w}^i) \tag{A.335}$$

This objective is very similar to the Equation A.332, apart from adding c_i, the confidence of agent i. When there is a high confidence at a node, it is more important to have a good performance on its data. To optimize the Equation A.335, instead of ADMM, as it was done in [215], they recommend a simple decentralized coordinate descent method after arbitrary initialization of local models:

1. Update step: Coordinate descent

$$\mathbf{w}^i_{t+1} = \mathbf{w}^i_t - \frac{1}{L_i}\nabla_{\mathbf{w}^i}\mathcal{Q}_{\mathcal{L}}(\mathbf{W}_{(t)}) = (1-\alpha)\mathbf{w}^i_t + \tag{A.336}$$

$$\alpha\Big(\sum_{j\in\mathcal{N}_i}\frac{N_{ij}}{D_{ii}}\mathbf{w}^j_t - \mu c_i\nabla\mathcal{L}_i(\mathbf{w}^i_t)\Big), \tag{A.337}$$

   where $\alpha = \frac{1}{1+\mu c_i L_i^{loc}} \in (0,1]$ for all $j \in \mathcal{N}_i$, and L_i are some Lipsitz constants.

2. Broadcast step: $\mathbf{w}^i_{t+1}$ will be sent to nodes $j \in \mathcal{N}_i$

Above its simplicity, a big advantage of this method is that it can incorporate differential privacy[63] as well, adding a Laplacian noise vector to the update step.

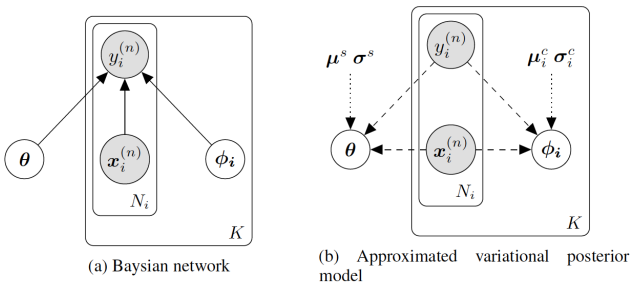
A.12 Variational Federated Multitask Learning

Variational Federated Multi-Task Learning (VIRTUAL) [47] has been developed to address federated MTL for generic, non-convex models using a star shaped Bayesian network (Figure A.20) with the server parameters θ in the center and local parameters ϕ_k at the leaves. The idea is that, similarly to *progressive networks* [184], the local model reuses

knowledge aggregated by the server through "gating" activation from the parallely running server model

$$\mathbf{a}_k^{(l+1)} = \sigma(\mathbf{U}_\mathbf{k}^\mathbf{l}\mathbf{a}_k^{(l)} + \alpha\mathbf{V}_\mathbf{k}^\mathbf{l}(\sigma(\mathbf{a}_s^{(l+1)}))), \tag{A.338}$$

where $\mathbf{U}_k^l$ and $\mathbf{V}_k^l$ are the weight matrices belonging to the client and server activation $\mathbf{a}_\mathbf{k}^{(\mathbf{l})}$ and $\mathbf{a}_\mathbf{s}^{(\mathbf{l})}$, respectively, and α is a gating weight. These weights, all together, add up to local parameters ϕ_k at machine k, that is, trained simultaneously with updates to be sent to the coordinator.



(a) Baysian network

(b) Approximated variational posterior model

Figure A.20 Variational inference model of [47]. Plates represent replicates over K clients and N_i data-points at node i. **(a)** conditional dependence of y on the shared local model parameters as well as on input $\mathbf{x}$ according to the discriminative model $p(y_i^{(n)}|\mathbf{x}_\mathbf{i}^{(\mathbf{n})},\theta,\phi_\mathbf{i})$. **(b)** the dashed arrows denote the dependencies of the parameter posteriors on data point n of client i (likelihood), (μ^s,σ^s) and (μ_i^c,σ_i^c) represent Gaussian priors on θ and ϕ_i.

The problem to be solved at the clients is to maximize the likelihood given the local data set D^i:

$$p(D^i|\theta,\phi^i) = \prod_{k=1}^{n^{(i)}} p(y_i^{(k)}|x_i^{(k)},\theta,\phi^i). \tag{A.339}$$

This factorization is equivalent with enforcing data points in D^i to be conditionally independent given θ and $\phi^1,\ldots\phi^K$. Assuming a prior $p(\theta,\phi^1,\ldots,\phi^K) = p(\theta)\prod_{i=1}^K p(\phi^i)$, given all the data-sets $D^{1:K}$, the posterior to be found by the federated multi-task learning is the

maximum of the expression

$$p(\theta, \phi^1, \ldots, \phi^K | D^{1:K}) \propto \frac{1}{p(\theta)^{K-1}} \prod_{i=1}^{K} p(\theta.\phi^i | D^i) \tag{A.340}$$

Due to thus intractable posterior in A.340, VIRTUAL uses *expectation propagation* (EP, see Section A.5) [159] to approximate it with a proxy posterior:

$$q(\theta, \phi^1, \ldots, \phi^K) = s(\theta)c(\phi^1, \ldots, \phi^K) = (\prod_{k=1}^{K} s^k(\theta))(\prod_{k=1}^{K} c^k(\phi^k)), \tag{A.341}$$

Using variational methods, as it is described in [24] and [159], (see Section A.5), finding a good approximation of the posterior is equivalent with finding the best available parameters in the model.

EP optimization refines one factor at each step, first computing a refined posterior, by replacing the refining factor of the proxy with the respective one from the true posterior. Next, it finds the new $s^k(\theta)$ and $c^k(\phi^k)$ distribution over the server and client parameters, respectively, that minimize the *KL-divergence* (Section A.3) between the full proxy and refined one.

Thus, the optimization step t at client k the goal is to find new proxies (new approximations for the factors for the true distribution) s_t^k and $c_t^k(\phi^k)$ by minimizing the following variational free energy function:

$$\mathcal{L}(s^k(\theta), c^k(\phi^k)) = KL\left(s^k(\theta)\frac{s_{t-1}(\theta)}{s_{t-1}^{(k)}(\theta)} \| p(\theta)\frac{s_{t-1}(\theta)}{s_{t-1}^{(k)}(\theta)}\right) \tag{A.342}$$

$$+ KL(c^{(k)}(\phi^{(k)}) \| p(\phi^{(k)})) - \mathbb{E}_{s_t(\theta), c^{(k)}(\phi^{(k)})} \log p(\mathcal{D}^{(k)} | \theta, \phi^{(k)}), \tag{A.343}$$

where $s_t(\theta) = s_t^{(k)}(\theta) \prod_{j \neq k} s_{t-1}^j(\theta)$, with the updated local factor $s_t^{(k)}(\theta)$ being the new posterior over the server parameters, using which the next node will start the process again.

This can be optimized by gradient descent that uses Monte Carlo estimates of the parameters with the re-parametrization trick [124] (see Section A.5).