FLECS: A Federated Learning Second-Order Framework via Compression and Sketching

Artem Agafonov\textsuperscript{1,2}, Dmitry Kamzolov\textsuperscript{2,1}, Rachael Tappenden\textsuperscript{3}, Alexander Gasnikov\textsuperscript{1,4,5}, and Martin Takáč\textsuperscript{2}

\textsuperscript{1}Moscow Institute of Physics and Technology, Dolgoprudny, Russia
\textsuperscript{2}Mohamed bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE
\textsuperscript{3}University of Canterbury, Christchurch 8041, New Zealand
\textsuperscript{4}Institute for Information Transmission Problems, Moscow, Russia
\textsuperscript{5}HSE University, Moscow, Russia

Abstract

Inspired by the recent work FedNL (Safaryan et al, FedNL: Making Newton-Type Methods Applicable to Federated Learning), we propose a new communication efficient second-order framework for Federated learning, namely FLECS. The proposed method reduces the high-memory requirements of FedNL by the usage of an L-SR1 type update for the Hessian approximation which is stored on the central server. A low dimensional ‘sketch’ of the Hessian is all that is needed by each device to generate an update, so that memory costs as well as number of Hessian-vector products for the agent are low. Biased and unbiased compressions are utilized to make communication costs also low. Convergence guarantees for FLECS are provided in both the strongly convex, and nonconvex cases, and local linear convergence is also established under strong convexity. Numerical experiments confirm the practical benefits of this new FLECS algorithm.

1 Introduction

This paper presents a novel second-order method for Federated Learning called FLECS. In this setup, multiple agents/devices/workers collaborate to solve a machine learning problem by communicating over a central node (server). It is assumed that the central node is a large and powerful machine, but the agents/devices are much smaller (for example, a cellphone, or laptop). Moreover, raw data is stored locally on each worker and cannot be exchanged or transferred. The finite sum minimization, or Federated Learning, problem can be formalized in the following way:

\[
\min_{w \in \mathbb{R}^d} \left\{ F(w) := \frac{1}{n} \sum_{i=1}^{n} f_i(w) \right\},
\]

where \( f_i(w) \) is the loss function (parametrized by \( w \in \mathbb{R}^d \)) associated with the data stored on the \( i \)-th machine, \( F(w) \) is the empirical loss function, and both the problem dimension \( d \), and the number of machines \( n \), can be large.

Problems of the form (1) arise not only in distributed machine learning \([23, 24]\), but also in robotics, resource allocation, power system control, control of drone or satellite networks, distributed statistical inference and optimal transport, and multi-agent reinforcement learning \([29, 33, 28, 14, 37, 35]\).

A variety of first-ordered methods have been introduced for the problem (1) under the stated computational set-up (a central server connected to multiple smaller devices). Early work includes that of \([10, 11, 26]\), which focused on distributed first order optimization algorithms that were designed with the computational structure in mind. Recent advances include works focusing on reducing the communication burden via compression techniques \([2, 19, 9]\), including momentum for algorithm speed up \([27, 33]\), variance reduction techniques \([20]\), the use of adaptive learning rates \([40]\), and

Preprint. Under review.
works that focus on maintaining client data privacy [16].

Beyond first-order methods, second-order methods have also been developed for Federated Learning. Some of these algorithms [42, 15, 12, 5, 1] utilize statistical similarity (the homogeneous data setting), which, roughly speaking, means that the local functions $f_i$ are similar, and they approximate the overall empirical risk function $F$ well. FedNL [36] for the second-order method for Federated Learning in the case of heterogeneous data. While FedNL makes a valuable step toward the applicability of second-order methods to Federated Learning, this framework seems to be impractical for large-scaled problems, since $d \times d$ Hessian approximations are stored locally on workers and transferred to the server via compressed matrix communications. (This violates the assumption in the current work, that the individual workers have small memory.)

The goal here is to build upon the work in [36] and develop a new second order framework for Federated Learning that is based upon more realistic assumptions about the available computational architecture. That is, we assume that individual workers are devices with relatively low memory availability and computational capacity (e.g. mobile phones), but they can connect to a large and powerful central machine. The typical situation, that downloading is much faster than uploading, is also assumed. We introduce the FLECS second-order framework for strongly convex and non-convex optimization, which places no additional memory on workers and reduces the number of Hessian-vector products per node. Therefore, our algorithm can be viewed as the lightweight FedNL, since it provides similar theoretical guarantees and has almost the same global. FLECS uses sketching and matrix compression for efficient communication and allows for several Hessian learning techniques (including truncated LSR1 update) and two iterate updates, namely Truncated inverse Hessian approximation and FedSONIA, inspired by [30] and [22].

1.1 Contributions

We propose FLECS: a Federated LEarning Second-Order Framework via Compression and Sketching with the following properties.

- **No additional storage on workers.** In large-scale machine learning problems, it is impossible to store the $d \times d$-dimensional Hessian approximations. For a dataset with $d = 50,000$ features, the local Hessian approximation requires more than $10$ GB of memory. For FLECS, the Hessian approximations are stored on the server, so memory requirements for the individual workers remain low.

- **Efficient communication via sketching and compression.** To reduce the cost of communication, we use a sketching technique. This allows the exchange of $d \times m$ matrices instead of $d \times d$, where $m \ll d$ is the user defined memory size, which could be set as $m = 1$. Moreover, we exploit compression to make the algorithm even more communication-efficient. This strategy heavily improves the one in [36], since we decrease the dimension of the transferred matrices $d/m$ times. Moreover, sketching reduces the number of Hessian-vector products from $d$ to $m$.

- **Heterogeneous data setting.** Our framework is flexible and does not assume that data on the workers is independent and identically distributed.

- **Hessian Learning via Quasi-Newton updates.** Motivated by quasi-Newton methods, sketching techniques and inverse Hessian truncation, we propose the Truncated L-SR1 update and Direct update for our Hessian approximations.

- **Theoretical Analysis.** Convergence guarantees in both the strongly convex, and nonconvex cases, are provided, as well as a local linear rate of convergence under strong convexity independent of problem conditioning.

- **Competitive Numerical Experiments.** Our numerical experiments highlight the practical benefits of our proposed framework.

Organization. The remainder of the paper is organized as follows. Related works are described in Section 2. Section 3 presents the proposed framework, including all algorithmic details. In Section 4 we establish theoretical convergence guarantees (all proofs can be found in the appendix). Numerical experiments are provided in Section 5 (additional experiments are provided in the appendix), and concluding remarks and potential future research directions are given in Section 6.

---

1 An example of such a setting is (e.g., battery powered) wireless network [34, 41] where the cost of sending data from devices to the centralized server is energy demanding, implying severely limited communication bandwidth to the server [18, 8]. On the other hand, the server is usually connected to an energy network and has no constraints for sending data to other devices in the network.
whether a homogeneous or heterogeneous data setting is assumed. Algorithms in the first group rely
where
M
where
which incorporates quasi-Newton type updates to learn a good approximation to the Hessian.

Arguably, some of the most popular algorithms that use approximations to the Hessian are quasi-
The Nonconvex Newton method of [30] is a different approach, which is an adaptation of Newton’s
The work [3] proposed the use of a sampled
viewed as sketching matrix [38], and the Newton Sketch algorithm in [31], shows that even randomly
approximation of the global objective function
F
on the concept of statistical similarity, which, in practice, means that each local function
curvature information in one subspace and gradient information in the orthogonal complement.

Second-order methods are well established in the optimization literature. Such methods employ
the Hessian (or an approximation to it) and thereby important (possibly approximate and/or partial)
curvature information is available for utilization. The benefits include ‘better’ search directions, and
faster rates of convergence that may be independent of the condition number, although the algorithms
often have higher memory and computational requirements than first order methods. To balance the
trade off between improved convergence rates versus increased costs, approaches that involve
modifying or approximating the Hessian have been proposed, and several of these are discussed now.
Arguably, some of the most popular algorithms that use approximations to the Hessian are quasi-
Newton methods. They employ information from the iterates to build an approximation to the Hessian
that improves in accuracy as the algorithm progresses. Among these are algorithms that use BFGS,
SR1, and DFP updates [17, 29, 13], which are now considered to be classics in optimization due to
their effectiveness and practicality, and also because, under certain conditions, theoretical results
related to the accuracy of the Hessian approximations are available. This motivates the current work,
which incorporates quasi-Newton type updates to learn a good approximation to the Hessian.

Of particular interest is the SR1 update, which does not guarantee positive definiteness of the Hessian
approximation [13]. Previously, this was considered to be a disadvantage, but in the case of non-
convex optimization, it would appear to be advantageous. L-SR1, a limited memory variant of SR1,
belongs to the class of limited memory quasi-Newton algorithms [25, 4]. These methods store only the
last m gradient differences
Y
k
= [y
k
−m+1,...,y
k
] and iterate differences
S
k
= [s
k
−m+1,...,s
k
],
where m is the memory size, y
k
= ∇F(w
k
) − ∇F(w
k−1
) and s
k
= w
k
− w
k−1
. The compact form
of the L-SR1 update is

\[ B_k = B_0 + (Y_k - B_0 S_k) M_k^{-1} (Y_k - B_0 S_k)^T, \]

where
M_k = L_k + D_k + U_k^T - S_k^T B_0 S_k, \ S_k^T Y_k = L_k + D_k + U_k, \ D_k \text{ is diagonal, } L_k, U_k \text{ are }
strictly lower and upper triangular, and B_0 is the initial approximation.

The work [3] proposed the use of a sampled S_k, rather than one based upon the previous m iterate
differences. The idea is, given the current iterate w_k, simply sample m random directions, and use the
differences with w_k as the columns of s_k (forming Y_k appropriately). An S_k with this form can be
viewed as sketching matrix [38], and the Newton Sketch algorithm in [31], shows that even randomly
projected or a sub-sampled Hessian provides enough curvature information for fast convergence.

The Nonconvex Newton method of [30] is a different approach, which is an adaptation of Newton’s
method for nonconvex problems. At each iteration, the singular value decomposition of the Hessian is
computed, small eigenvalues (in absolute value) are truncated, and then a positive definite modification
of the Hessian is constructed. This is then used to create Newton-type direction.

SONIA [22] bridges the gap between first-order and second-order methods, combining the ideas of
quasi-Newton algorithms, sketching [31] and truncation [30]. It constructs the search directions using
curvature information in one subspace and gradient information in the orthogonal complement.

Second-order methods for Federated Learning, can generally be divided in two groups, based upon
whether a homogeneous or heterogeneous data setting is assumed. Algorithms in the first group rely
on the concept of statistical similarity, which, in practice, means that each local function f_i is a ‘good’
approximation of the global objective function F. (More formally, \|\nabla^2 F - \nabla^2 f_i\| \leq \beta \text{ for some}
\[ \beta > 0. \] Usually, in this case, methods utilize the part of the data stored on the server and use it to approximate the full Hessian \[ \{ \mathbf{R}^2 \mathbf{f}_i \mathbf{s}_k^i \} \mathbf{S}_k \].

No such statistical similarity (i.i.d.) assumption is made in the heterogeneous setting, which is perhaps more realistic (e.g., it may be impossible/impractical to have local data (on a mobile phone say), that approximates the data on the server well). The main work in this setting is FedNL \[ 36 \]. The authors proposed a way to learn Hessian approximations and then take an approximate Newton-type step. However, the main drawback is that the Hessian approximations are stored locally and communications performed via compressed \( d \times d \) matrices. For large-scale optimization problems, it is impossible to store the \( d \times d \) matrix locally, because of memory limitations. Extensions of FedNL can be found in \[ 21 \], \[ 32 \].

The approach proposed here combines ideas from quasi-Newton methods, sketching, and Hessian learning, and develops a second-order Federated Learning framework with efficient communications and without an additional memory burden on workers. All Hessian approximations are stored on the server, not the workers. Communications are based on a user-defined memory hyperparameter \( m \) and utilize compression to reduce communication. Sketching allows to transfer \( d \times m \) matrices and reduce the number of Hessian-vector products on the worker. The proposed framework allows for two Hessian learning techniques and two iterate update rules. The Truncated Inverse Hessian approximation is based on the idea of truncating eigenvalues (as in \[ 30 \]) of Hessian approximate, while FedSONIA is based upon SONIA \[ 22 \] and reduces the iteration complexity of the step from \( O(d^3) \) to \( O(nmd^2) \).

3 FLECS

Inspired by the works \[ 22 \], \[ 36 \], we propose a general framework for second-order methods for federated learning problems \[ 1 \] that allows for different forms for the iterates and different Hessian approximations/updates. One of main goals of this framework is to create fast communication-efficient second-order methods that do not place additional memory requirements on the workers.

To make broadcasting between the server and nodes cheap we employ sketching and compression. Throughout the paper, \( \mathbf{S}_k \in \mathbb{R}^{d \times n^m} \) denotes a sketching matrix, where \( m \) is the memory size \((m \ll d)\). Any operator that compresses a matrix is denoted by \( \mathbf{C} : \mathbb{R}^{d \times m} \rightarrow \mathbb{R}^{d \times m} \), e.g. random dithering, Rank-R, Top-K, Rand-K (description of compressors can be found in Appendix C). The Hessian approximation of the \( i \)-th worker at iteration \( k \) is denoted by \( \mathbf{B}_i^k \), and is stored on the central node.

Our framework can be described in three steps (further details follow in later subsections):

1. **Worker step.** Each worker receives the current iterate \( \mathbf{w}_k \), and their local Hessian approximation multiplied by a sketching matrix, i.e., the product \( \mathbf{B}_i^k \mathbf{S}_k \in \mathbb{R}^{d \times n} \). The matrix is sent directly (i.e., no compression), since, typically, downloading is faster than uploading. Each worker then computes their local Hessian-sketch product \( \mathbf{Y}_i^k := \nabla^2 \mathbf{f}_i(\mathbf{w}_k) \mathbf{S}_k \) and sends back the compressed difference \( \mathbf{C}_i^k := \mathbf{C}(\mathbf{Y}_i^k - \mathbf{B}_i^k \mathbf{S}_k) \in \mathbb{R}^{d \times m} \).\

2. **Hessian approximation update.** The server receives \( \mathbf{C}_i^k, \mathbf{M}_k^i := \mathbf{S}_k^T (\nabla^2 \mathbf{f}_i(\mathbf{w}_k)) \mathbf{S}_k \), and \( \nabla \mathbf{f}_i(\mathbf{w}_k) \) from all \( i = 1, \ldots, n \) workers. Using this information, as well as the current (local) Hessian approximation \( \mathbf{B}_k^i \), the new approximation \( \mathbf{B}_{k+1}^i \) is formed. Our framework allows \( \mathbf{B}_{k+1}^i \) to be computed by a (1) Truncated L-SR1 update, or (2) Direct update.

3. **Iterate update.** The server forms the matrices

\[
\mathbf{B}_k := \frac{1}{n} \sum_{i=1}^{n} \mathbf{B}_k^i, \quad \nabla \mathbf{F}(\mathbf{w}_k), \quad \mathbf{M}_k = \frac{1}{n} \sum_{i=1}^{n} \mathbf{M}_k^i,
\]

and calculates the new iterate \( \mathbf{w}_{k+1} \) via update rule:

\[
\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mathbf{p}_k
\]

where \( \alpha_k > 0 \) is the step-size. There are several options for computing the search direction \( \mathbf{p}_k \), including (1) a truncated inverse Hessian approximation step, or (2) via a SONIA step for Federated Learning (FedSONIA). The process then repeats from Step 1.

3.1 Initialization

The algorithm is initialized with a user defined memory size \( m \ll d \), and \( n \) matrices \( \mathbf{B}_0^i \in \mathbb{R}^{d \times d} \) are stored on the central node, each representing an approximation to the local Hessian for the \( i \)-th worker. Various additional parameters are initialized depending on the selected update rule (to be described later). At iteration \( k > 0 \) of FLECS, \( m \) directions \( \{ s_1, \ldots, s_m \} \) are randomly sampled at
which we refer to as the Direct update. Then, we utilize the following learning technique

\[
\omega > 0
\]

where

\[
L
\]

we use an error-feedback technique and send

\[
Algorithm 1
\]

To ensure numerically stability of the update (6), truncation is employed. At first, we take the spectral

values. In particular, define

\[
\begin{array}{c}
\text{containing the corresponding eigenvectors. Next, we compute the inverse }
\end{array}
\]

In this case we compute the

\[
\text{truncation constant.}
\]

\[
\text{setting the random seed to be equal to the iteration number } k \text{ during sampling.}
\]

3.2 Worker Step

Each of the workers receives \( B^i_k S_k \), and the iterate \( w_k \), from the server. On each node we compute

\[
C^i_k := C(Y^i_k - B^i_k S_k), \quad \nabla f_i(w_k), \quad M^i_k := S^T_k Y^i_k = S^T_k (\nabla^2 f_i(w_k)) S_k, \quad i = 1, \ldots, n. \tag{4}
\]

The quantities in (4) are used to construct the Hessian approximation on the main node. Since \( M^i_k \) is an \( m \times m \), it can be sent directly without compression. So too can the vector \( \nabla f_i(w_k) \). For the matrix \( Y^i_k \) we use the compression operator \( C \). In order to make the variance of this compression low, we use an error-feedback technique and send \( C^i_k = C(Y^i_k - B^i_k S_k) \).

3.3 Hessian Approximation update

The server receives (4) from each worker \( i = 1, \ldots, n \). Firstly, we restore an approximation to \( Y^i_k \) as

\[
\tilde{Y}^i_k = C^i_k + B^i_k S_k. \tag{5}
\]

Then, we apply one of the following options to update the Hessian approximation.

Hessian update option 1. Truncated L-SR1 update (Algorithm 1). In this case we compute the approximate Hessian of the \( i \)-th worker using the L-SR1 update:

\[
B^i_{k+1} = B^i_k + (\tilde{Y}^i_k - B^i_k S_k)(M^i_k - S^T_k B^i_k S_k)^{-1}(\tilde{Y}^i_k - B^i_k S_k)^T.
\tag{6}
\]

To ensure numerically stability of the update (6), truncation is employed. At first, we take the spectral decomposition of the small \((m \times m)\) matrix

\[
M^i_k - S^T_k B^i_k S_k = U^i_k L^i_k (U^i_k)^T,
\]

where \( L^i_k \in \mathbb{R}^{d \times d} \) is a diagonal matrix containing the eigenvalues, and \( U^i_k \) is an orthogonal matrix containing the corresponding eigenvectors. Next, we compute the inverse \((L^i_k)^{-1}\), and truncate small values. In particular, define \((l^i_k)_{jj}^{-1} := (L^i_k)_{jj}^{-1}\), and set all elements such that \([[(l^i_k)^{-1}]_{jj}] \leq \omega \) to be 0, where \( \omega > 0 \) is a user defined parameter that must be initialized. The truncated matrix is denoted by \([[(l^i_k)^{-1}]_{jj}]\). Finally, the update (6) becomes

\[
B^i_{k+1} = B^i_k + (\tilde{Y}^i_k - B^i_k S_k)(l^i_k)^{-1}(U^i_k)^T (\tilde{Y}^i_k - B^i_k S_k)^T.
\tag{7}
\]

Hessian update option 2. Direct update (Algorithm 2). Setting \( B_0 = 0 \) in (6) gives

\[
\tilde{B}^i_k = \tilde{Y}^i_k (M^i_k)^{-1} (\tilde{Y}^i_k)^T,
\tag{8}
\]

which we refer to as the Direct update. Then, we utilize the following learning technique

\[
B^i_{k+1} = (1 - \beta_k) B^i_k + \beta_k \tilde{B}^i_k,
\tag{9}
\]

where \( 0 < \beta_k \leq 1 \) is the learning rate. Thus, for the Direct update, the new Hessian approximation is a convex combination of the previous update and the matrix in (8).

Note, that for both Option 1 and Option 2, the Hessian approximation \( B^i_{k+1} \) is symmetric, since \( M^i_k = S^T_k \nabla f_i(w_k) S_k, B^i_k, \) and \( M^i_k - S^T_k B^i_k S_k \) are symmetric for \( i = 1, \ldots, n \).
well defined, which is crucial to avoid checking condition on the curvature pairs $Y_k^i$. Second, it ensures that after truncation, any eigenvalue (in absolute value) that is smaller (resp. greater) than a user defined threshold $\omega$ is replaced by $\omega$. This truncation is crucial for several reasons. Firstly, it leads to stable updates. Secondly, it ensures that after truncation, the resulting matrix will be full-rank and positive definite. Therefore, it guarantees that the following search direction is a decent direction

$$p_k = (|B_{k+1}|_\Omega^{-1}) \nabla F(w_k).$$

**Search direction option 2. FedSONIA (Algorithm 4).** FedSONIA is a modification of SONIA [22] that is adapted to the Federated Learning setting. The key idea of SONIA is to utilize curvature information in one subspace and perform a gradient step in orthogonal complement. The intuition is that, by incorporating some curvature information the resulting search direction should be ‘better’ than a pure gradient direction, but the user has control over the dimension of each subspace, and therefore they have control over the cost of obtaining the curvature information. Truncation (Definition 1) and modified SR1 update (eq. (6)) ensure that the Hessian-approximation is full-rank, positive and well defined, which is crucial to avoid checking condition on the curvature pairs $Y_k^i S_k (\tilde{Y}_k^i)^T S_k$ for

**Algorithm 2 Direct update.**

**Require:** $\tilde{Y}_k^i \in \mathbb{R}^{d \times m}$, $M_k^i \in \mathbb{R}^{m \times m}$, $B_k^i \in \mathbb{R}^{d \times d}$ \forall i

1: On the server: $0 < \beta_k \leq 1$ – learning rate.
2: for $i = 1, \ldots, n$ do
3: compute $\tilde{B}_k^i = \tilde{Y}_k^i (M_k^i)^T$;
4: select learning rate $\beta_k$
5: compute $B_{k+1}^i = (1 - \beta_k)B_k^i + \beta_k \tilde{B}_k^i$
6: end for

**Algorithm 3 Truncated inverse Hessian approximation**

**Require:** $\nabla F(w_k) \in \mathbb{R}^d$, $\tilde{Y}_k^i \in \mathbb{R}^{d \times m}$, $M_k^i \in \mathbb{R}^{m \times m}$, $B_{k+1}^i \in \mathbb{R}^{d \times d}$, $\Omega > \omega > 0$ – truncation constants.

1: On the server:
2: truncate $B_{k+1}^i = V_k \Lambda_k V_k^T$;
3: compute search direction $p_k$ via (11);
4: return $p_k$.

3.4 Iterate update

Note that, at the start of the iterate update step, all the Hessian approximations $B_{k+1}^i$, and all variables from (4), are available on the main node. Then, the server forms

$$\nabla F(w_k) = \frac{1}{n} \sum_{i=1}^n f_i(w_k), \quad \tilde{Y}_k := \frac{1}{n} \sum \tilde{Y}_k^i = \frac{1}{n} \sum_{i=1}^n (C_k^i + H_k^i S_k),$$

$$M_k := \frac{1}{n} \sum_{i=1}^n M_k^i = S_k \nabla^2 F(w_k) S_k^T, \quad B_{k+1} := \frac{1}{n} \sum_{i=1}^n B_{k+1}^i.$$

Now the goal is to compute a search direction $p_k$, of the general form:

$$p_k = -A_k \nabla F(w_k),$$

for some matrix $A_k$. Two specific options for $A_k$ are described now.

**Search direction option 1. Truncated inverse Hessian approximation (Algorithm 3).** Here, $A_k$ is taken to be the truncated inverse Hessian approximation $(|B_{k+1}|_\Omega^{-1})$. Consider the following:

**Definition 1.** Let $B_k, V_k, \Lambda_k$ be matrices such that $B_k = V_k \Lambda_k V_k^T$, and let $0 < \omega \leq \Omega$. The truncated inverse Hessian approximation of $B_k$ is $(|B_k|_\Omega^{-1}) := V_k (|\Lambda_k|_\Omega^{-1}) V_k^T$, where $(|\Lambda_k|_\Omega^{-1})_i = \min \{|\Lambda|_i, \omega\}, \Omega \}.$

Definition 1 was proposed in [30] and was used to provide a convergence guarantee for their Nonconvex Newton method (to a local minimum). Firstly, an eigen-decomposition of $B_k$ is computed, but with every eigenvalue replaced by its absolute value. Secondly, a thresholding step is applied, so that any eigenvalue (in absolute value) that is smaller (resp. greater) than a user defined threshold $\omega$ (resp. $\Omega$) is replaced by $\omega$ (resp. $\Omega$). This truncation is crucial for several reasons. Firstly, it ensures $B_k$ is well-defined, which leads to stable updates. Secondly, it ensures that after truncation, the resulting matrix will be full-rank and positive definite. Therefore, it guarantees that the following search direction is a decent direction

$$p_k = (|B_{k+1}|_\Omega^{-1}) \nabla F(w_k).$$

**Search direction option 2. FedSONIA (Algorithm 4).** FedSONIA is a modification of SONIA [22] that is adapted to the Federated Learning setting. The key idea of SONIA is to utilize curvature information in one subspace and perform a gradient step in orthogonal complement. The intuition is that, by incorporating some curvature information the resulting search direction should be ‘better’ than a pure gradient direction, but the user has control over the dimension of each subspace, and therefore they have control over the cost of obtaining the curvature information. Truncation (Definition 1) and modified SR1 update (eq. (6)) ensure that the Hessian-approximation is full-rank, positive and well defined, which is crucial to avoid checking condition on the curvature pairs $Y_k^i S_k (\tilde{Y}_k^i)^T S_k$ for
Algorithm 4 FedSONIA

Require: $\nabla F(w_k) \in \mathbb{R}^d$, $\bar{Y}_k \in \mathbb{R}^{d \times m}$, $M_k \in \mathbb{R}^{m \times m}$, $\Omega > \omega > 0$ – truncation constants.
1: On the server;
2: compute $B_{S}^{\text{SONIA}} := \bar{Y}_k (M_k)^{\dagger} \bar{Y}_k^T$;
3: compute $QR$ factorization of $\bar{Y}_k (= Q_k R_k)$;
4: compute spectral decomposition of $R_k (M_k)^{\dagger} R_k^{T} (= V_k \Lambda_k V_k^T)$;
5: construct $\tilde{V}_k := Q_k V_k$;
6: truncate $\Lambda_k$ to form $|\Lambda_k|_\omega^{(1)}$ via Definition [1];
7: Set $\rho_k$ and decompose gradient via (13);
8: Compute search direction $p_k$ via (14);
9: return $p_k$.

FedSONIA). To construct the approximation of the Hessian we use (8). Note, that SONIA does not need the Hessian approximation $B_k$ itself to commit a step. Instead, it utilizes only Hessian-matrix product $Y_k$ and sketching matrix $S_k$.

Define $B_{S_k+1}^{\text{SONIA}} = Y_k (M_k)^{\dagger} Y_k^T$. Following the strategy in [22], apply an economy QR factorization of $Y_k = Q_k R_k$, so that $Q_k \in \mathbb{R}^{d \times m}$ has orthonormal columns and $R_k \in \mathbb{R}^{m \times m}$ is upper triangular. Thus, $B_{S_k+1}^{\text{SONIA}} = Q_k R_k (M_k)^{\dagger} R_k^{T} Q_k^{T}$. Applying the spectral decomposition, we obtain $R_k (M_k)^{\dagger} R_k^{T} = V_k \Lambda_k V_k^T$, where the columns of $V_k \in \mathbb{R}^{m \times m}$ form an orthonormal basis (of eigenvectors), and $\Lambda_k \in \mathbb{R}^{m \times m}$ is a diagonal matrix containing the corresponding eigenvalues. Therefore,

$$B_{S_k+1}^{\text{SONIA}} = Q_k V_k \Lambda_k V_k^T Q_k^{T} \tilde{V}_k := Q_k V_k \tilde{V}_k \Lambda_k \tilde{V}_k^T,$$

where $\tilde{V}_k \in \mathbb{R}^{d \times m}$ has orthonormal columns since $Q_k$ has orthonormal columns and $V_k$ is an orthogonal matrix. Then, we apply Definition [1] to form the truncated approximation

$$\left( |B_{S_k+1}^{\text{SONIA}}|_\omega \right)^{-1} = \tilde{V}_k |\Lambda_k|_\omega^{(1)}^{-1} \tilde{V}_k^T.$$  

Next, we perform subspace decomposition on the current gradient direction:

$$\nabla F(w_k) = g_k + g_k^T,$$

where $g_k = \tilde{V}_k \tilde{V}_k^T \nabla F(w_k)$ and $g_k^T = \left( I - \tilde{V}_k \tilde{V}_k^T \right) \nabla F(w_k)$. Putting this all together, the FedSONIA search direction (similar to [22]) is:

$$p_k := - \left( |B_{S_k+1}^{\text{SONIA}}|_\omega \right)^{-1} g_k - \rho_k g_k^T \tilde{V}_k \tilde{V}_k^T = 1 \left( |B_{S_k+1}^{\text{SONIA}}|_\omega \right)^{-1} + \rho_k (I - \tilde{V}_k \tilde{V}_k^T) \nabla F(w_k),$$

where $\rho_k \in [1/3, 1/(\max(|\Lambda_k|_\omega^{(1)}^{-1})))$. One has control over the cost of the curvature information via (14) through the memory size $m$. Note that this will also affect the communication complexity.

Finally, a new search direction $p_k$ is computed, which depends on the chosen update strategy (Algorithm 3 or 4) and to compute the new iterate $w_{k+1}$ via (13). Recall that for both these strategies the resulting search direction $p_k$ has the form (10), where $A_k = \left( |B_{S_k+1}^{\text{SONIA}}|_\omega \right)^{-1}$ for the Truncated inverse Hessian approximation (11), and $A_k = \left( |B_{S_k+1}^{\text{SONIA}}|_\omega \right)^{-1} + \rho_k (I - \tilde{V}_k \tilde{V}_k^T)$ for FedSONIA (14). The process then restarts (from Section 2.2). Our framework is summarized in Algorithm 5.

Complexity. The per iteration complexity of FLECS depends on the chosen options for both the Hessian approximation and the iterate updates. The complexity of the Truncated L-SR1 update for one Hessian approximation consists of matrix products ($O(md^2)$) and a singular value decomposition of $m \times m$ matrix ($O(m^3)$). For the Direct update, the matrix products cost $O(md^2)$ and forming the pseudo-inverse of the $m \times m$ matrix costs $O(m^3)$. Therefore, the total complexity of either Hessian approximation update is $O(nmd^2)$. The complexity of computing search direction via Truncated inverse Hessian approximation is $O(d^3)$, since we utilize spectral decomposition and products of $d \times d$ matrices. FedSONIA update comprises 1) QR factorization ($O(dm^2)$); 2) pseudo-inverse of $m \times m$ matrix ($O(m^3)$); 3) spectral decomposition of $m \times m$ matrix ($O(m^3)$); which gives total complexity $O(dm^2)$. Finally, the complexity of FLECS on the server only depends on options for the search direction and equals $O(d^3 + nmd^2)$ for Truncated Inverse Hessian approximation and $O(nmd^2)$ for FedSONIA. The the worker step’s complexity consists of $m$ Hessian-vector products and matrix multiplication ($O(m^2)$).
We introduce assumptions and theoretical results for the strongly convex case. We introduce the assumptions for nonconvex case. We begin by listing the preliminary lemma.

Assumption 1. The function $F(w)$ is twice continuously differentiable for all $w \in \mathbb{R}^d$.

We begin by listing the preliminary lemma.

Lemma 1. Let Assumption 1 hold. Let $A_k$ be defined as in (10) depending on the choice of iterates update. Then, for any $k \geq 0$ there exist constants $0 < \mu_1 \leq \mu_2$ such that $\mu_1 I \preceq A_k \preceq \mu_2 I$.

We introduce assumptions and theoretical results for the strongly convex case.

Assumption 2. There exist positive constants $\mu$ and $L$ such that $\mu I \preceq \nabla^2 F(w) \preceq LI$, $\forall w \in \mathbb{R}^d$.

The following theorem establishes global linear convergence of FLECS under strong convexity.

Theorem 1. Let Assumptions 1 and 2 hold. Then for iterates $\{w_k\}$ generated by Algorithm 5 with $0 < \alpha_k = \alpha \leq \frac{\mu}{\mu_2 L}$, we have $F(w_k) - F^* \leq (1 - \alpha \mu \Omega)^k[F(w_0) - F^*]$ for all $k \geq 0$.

We are now ready to present a convergence guarantee which shows local linear convergence for FLECS under strong convexity. Note that, while Theorem 1 shows that FLECS converges linearly under strong convexity, the rate depends on parameters related to the Hessian approximations. The following local rate is independent of problem conditioning.

Theorem 2. Let Assumptions 1 and 2 hold, and $\|w_0 - w^*\| \leq \frac{\mu^2}{2}\Omega$, $\frac{1}{n} \sum_{i=1}^{n} \|B_{k+1}^i - \nabla^2 f_i(w^*)\|_F \leq \frac{\mu^2}{\Omega}$.

Then for iterates $\{w_k\}$ generated by Algorithm 5 with Truncated inverse Hessian approximation step (Algorithm 3) with parameters $\alpha_k = \alpha = 1$, $0 \leq \omega \leq \mu$, $\Omega \geq L$ we have $\|w_k - w^*\|^2 \leq \frac{1}{2\mu} \|w_0 - w^*\|^2$ for all $k \geq 0$.

We introduce the assumptions for nonconvex case.

Assumption 3. The function $F$ is bounded below by a scalar $\bar{F}$.

Assumption 4. The gradients of $F$ are $L$-Lipschitz continuous for all $w \in \mathbb{R}^n$.

The following result shows that FLECS is convergent when problem (1) is nonconvex.

Algorithm 5 FLECS

Require: $w_0$ – starting point, $m$ – memory size, $B^i_0$ – initial Hessian approximations for each worker $i = 1 \ldots n$ on the server; $0 < \omega < \Omega$ – truncation constants.

1: for $k = 0, 1, \ldots$ do
2: On $i$-th machine:
3: sample $S_k \in \mathbb{R}^{d \times m}$; 
4: collect $H_k^i S_k$, $w_k$ from the server;
5: compute $Y_k^i := \nabla^2 f_i(w_k) S_k$, $U_k^i := S_k^T Y_k^i$;
6: send $\nabla f_i(w_k)$, $M_k^i$, $C_k^i = \mathcal{C}(Y_k^i - H_k^i S_k)$ to the server.
7: On the server:
8: sample $S_k$;
9: collect $C_k^i$, $M_k^i$, $i = 1 \ldots n$ from workers;
10: compute $Y_k^i = C_k^i + H_k^i S_k$;
11: compute $B_{k+1}^i$ via Algorithm 1 or select learning rate $\beta_k$ and compute $B_{k+1}^i$ via Algorithm 2;
12: form $\nabla F(w_k), \bar{Y}_k, M_k, B_{k+1}$ as average over workers of $\nabla f_i(w_k), \bar{Y}_k^i, M_k^i, B_{k+1}^i$, $i = 1, \ldots, n$;
13: compute search direction $p_k$ via Algorithm 3 or 4;
14: select stepsize $\alpha_k$ and set $w_{k+1} = w_k + \alpha_k p_k$;
15: sample $S_{k+1} \in \mathbb{R}^{d \times m}$;
16: send $w_{k+1}, B_{k+1} S_{k+1}$ to all workers.
17: end for

4 FLECS: Convergence Theory

In this section, convergence theory for our FLECS framework is presented. We provide global convergence results in both the strongly convex, and nonconvex cases. A local convergence result is also presented, which shows local linear convergence of the iterates under strong convexity. We define $w_0$ to be the initial point, $w^* = \arg \min_{w \in \mathbb{R}^d} F(w)$ and $F^* = F(w^*)$.

Assumption 1. The function $F(w)$ is twice continuously differentiable for all $w \in \mathbb{R}^d$.

Assumption 2. There exist positive constants $\mu$ and $L$ such that $\mu I \preceq \nabla^2 F(w) \preceq LI$, $\forall w \in \mathbb{R}^d$.

The following theorem establishes global linear convergence of FLECS under strong convexity. We introduce the assumptions for nonconvex case.

Assumption 3. The function $F$ is bounded below by a scalar $\bar{F}$.

Assumption 4. The gradients of $F$ are $L$-Lipschitz continuous for all $w \in \mathbb{R}^n$.

The following result shows that FLECS is convergent when problem (1) is nonconvex.
Theorem 3. Let Assumptions 2 and 3 hold and \( w_0 \) be the starting point. Then after \( T > 0 \) iterations of Algorithm 1 with \( 0 < \alpha_k = \alpha \leq \frac{4\mu}{\|\nabla^2 F\|} \), we have
\[
\frac{1}{T} \sum_{k=0}^{T-1} \|\nabla F(w_k)\|^2 \leq \frac{2(F(w_0) - \hat{F})}{\alpha \mu_1 T} \rightarrow_{T \rightarrow \infty} 0.
\]

5 Numerical Experiments

We present numerical experiments for FLECS on a regularized logistic regression problem:
\[
\min \left\{ \frac{1}{n} \sum_{i=1}^{n} \frac{1}{r} \sum_{j=1}^{r} \log(1 + \exp(-b_{ij} a_{ij}^T w)) + \mu \|w\|^2 \right\},
\]
where \( \{a_{ij}, b_{ij}\} j \in [m] \) are data points on \( i \)-th device. We use three datasets from the LIBSVM library: a9a (123 features), gisette-scale (5000 features) and real-sim (20958 features). We use Top-K with \( k = 4d \) and random dithering applied to each matrix column with \( s = 128 \) levels and \( \infty \)-norm as a matrix compressors. The performance of FLECS is compared with FedNL.

In our experiments we set regularization parameter \( \mu = 10^{-5} \), the memory size to \( m = 16 \) and use \( B_k^i = 0 \) as the initial Hessian approximation for FLECS and FedNL for random dithering and \( B_k^i = \nabla f_i(w_0) \) for Top-K. Truncation parameters were set to \( \omega = 10^{-3} \), \( \Omega = 10^8 \), and for the FedSONIA update we set \( \rho = \frac{1}{16} \) and \( \alpha = 1 \). For FLECS we use a Truncated L-SR1 Update (Algorithm 1) and a Direct update (Algorithm 2) with learning rate \( \beta = 1 \) as the Hessian learning techniques. For FedNL we use the same \( \beta \).

The following three rules were used to compute the search direction \( p_k \): (1) a Truncated Inverse Hessian Approximation (Algorithm 3), (2) FedSONIA (Algorithm 4) with \( \rho_k = \frac{1}{16} \), and (3) a regularized update where \( B_{k+1} = B_k + \frac{1}{2} \sum_{i=1}^{n} \|C_{ik}\| I \), where \( C_{ik} \) are defined in (4). The last update is motivated by 36. Note, that in the case of FLECS this update does not guarantee positive definiteness of Hessian approximation \( B_k \) if \( m < d \), but it can still perform well if \( m \) is large enough.

For FedNL after random dithering the matrix \( B_{k+1} \) might not be symmetric, so we use \( \frac{1}{2} \) \( (B_{k+1} + B_{k+1}^T) \) as Hessian approximation. In this case, we compare algorithms on the number of bits sent to the server per iteration. For Top-K compression, the communication complexity is roughly the same, so we estimate the performance based on the number of Hessian-vector products.

For the iterates update we set the step-size \( \alpha = 1 \) for all algorithms and datasets except FedSONIA on gisette and real-sim, where we set \( \alpha = 0.1 \), as it gives the best performance. Further details specifying algorithm parameters and implementation, as well as additional numerical experiments, can be found in Appendix B.

6 Conclusion

This work presents a new second order framework for Federated Learning problems. The framework employs sketching and compression to ensure communication costs are low, and Hessian approximations are stored on the central server so that memory requirements for the workers is low. Due to the inclusion of partial approximate second order information, the algorithm enjoys favourable convergence guarantees, and numerical experiments show the practical benefits of our approach.
References

[1] A. Agafonov, P. Dvurechensky, G. Scutari, A. Gasnikov, D. Kamzolov, A. Lukashevich, and A. Daneshmand. An accelerated second-order method for distributed stochastic optimization. In 2021 60th IEEE Conference on Decision and Control (CDC), pages 2407–2413, 2021.

[2] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. Advances in Neural Information Processing Systems, 30, 2017.

[3] A. S. Berahas, M. Jahani, P. Richtárik, and M. Takáč. Quasi-newton methods for machine learning: forget the past, just sample. Optimization Methods and Software, pages 1–37, 2021.

[4] J. Brust, J. B. Erway, and R. F. Marcia. On solving l-sr1 trust-region subproblems. Computational Optimization and Applications, 66(2):245–266, 2017.

[5] B. Bullins, K. Patel, O. Shamir, N. Srebro, and B. E. Woodworth. A stochastic newton algorithm for distributed convex optimization. Advances in Neural Information Processing Systems, 34, 2021.

[6] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-newton matrices and their use in limited memory methods. Mathematical Programming, 63(1):129–156, 1994.

[7] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. ACM transactions on intelligent systems and technology (TIST), 2(3):1–27, 2011.

[8] D. Chen, C. S. Hong, Y. Zha, Y. Zhang, X. Liu, and Z. Han. Fedsvrg based communication efficient scheme for federated learning in mec networks. IEEE Transactions on Vehicular Technology, 70(7):7300–7304, 2021.

[9] Y. Chen, R. S. Blum, M. Takac, and B. M. Sadler. Distributed learning with sparsified gradient differences. IEEE Journal of Selected Topics in Signal Processing, 2022.

[10] J. K. cný, H. B. McMahan, D. Ramage, and P. Richtarik. Federated optimization: Distributed machine learning for on-device intelligence. arXiv preprint arXiv:1610.02527, 2016.

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

[12] A. Daneshmand, G. Scutari, P. Dvurechensky, and A. Gasnikov. Newton method over networks is fast up to the statistical precision, 2021.

[13] J. E. Dennis, Jr and J. J. Moré. Quasi-newton methods, motivation and theory. SIAM review, 19(1):46–89, 1977.

[14] P. Dvurechenskii, D. Dvinskikh, A. Gasnikov, C. Uribe, and A. Nedich. Decentralize and randomize: Faster algorithm for wasserstein barycenters. In Advances in Neural Information Processing Systems, pages 10760–10770, 2018.

[15] P. Dvurechensky, D. Kamzolov, A. Lukashevich, S. Lee, E. Ordentlich, C. A. Uribe, and A. Gasnikov. Hyperfast second-order local solvers for efficient statistically preconditioned distributed optimization. arXiv preprint arXiv:2102.08246, 2021.

[16] P. K. et al. Advances and open problems in federated learning, 2019.

[17] R. Fletcher. Practical methods of optimization. John Wiley & Sons, 2013.

[18] H. He, X. Yu, J. Zhang, S. Song, and K. B. Letaief. Cell-free massive mimo for 6g wireless communication networks. Journal of Communications and Information Networks, 6(4):321–335, 2021.

[19] S. Horváth, C.-Y. Ho, L. Horvath, A. N. Sahu, M. Canini, and P. Richtárik. Natural compression for distributed deep learning. arXiv preprint arXiv:1905.10988, 2019.

[20] S. Horváth, D. Kovalev, K. Mishchenko, S. Stich, and P. Richtárik. Stochastic distributed learning with gradient quantization and variance reduction, 2019.

[21] R. Islamov, X. Qian, and P. Richtárik. Distributed second order methods with fast rates and compressed communication. In International Conference on Machine Learning, pages 4617–4628. PMLR, 2021.
[22] M. Jahani, M. Nazari, R. Tappenden, A. Berahas, and M. Takáč. Sonia: A symmetric blockwise truncated optimization algorithm. In International Conference on Artificial Intelligence and Statistics, pages 487–495. PMLR, 2021.

[23] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In CIDR, volume 1, pages 2–1, 2013.

[24] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 583–598, 2014.

[25] X. Lu. A study of the limited memory SR1 method in practice. University of Colorado at Boulder, 1996.

[26] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), 2017.

[27] K. Mishchenko, E. Gorbunov, M. Takáč, and P. Richtárik. Distributed learning with compressed gradient differences, 2019.

[28] A. Nedić, A. Olshevsky, and C. A. Uribe. Fast convergence rates for distributed non-bayesian learning. IEEE Transactions on Automatic Control, 62(11):5538–5553, 2017.

[29] J. Nocedal and S. J. Wright. Numerical optimization. Springer, 1999.

[30] S. Paternain, A. Mokhtari, and A. Ribeiro. A newton-based method for nonconvex optimization with fast evasion of saddle points. SIAM Journal on Optimization, 29(1):343–368, 2019.

[31] M. Pilanci and M. J. Wainwright. Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence. SIAM Journal on Optimization, 27(1):205–245, 2017.

[32] X. Qian, R. Islamov, M. Safaryan, and P. Richtárik. Basis matters: Better communication-efficient second order methods for federated learning. arXiv preprint arXiv:2111.01847, 2021.

[33] M. Rabbat and R. Nowak. Decentralized source localization and tracking wireless sensor networks. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, volume 3, pages 921–924, 2004.

[34] V. Rajaravivarma, Y. Yang, and T. Yang. An overview of wireless sensor network and applications. In Proceedings of the 35th Southeastern Symposium on System Theory, 2003., pages 432–436. IEEE, 2003.

[35] S. S. Ram, V. V. Veeravalli, and A. Nedic. Distributed non-autonomous power control through distributed convex optimization. In IEEE INFOCOM 2009, pages 3001–3005. IEEE, 2009.

[36] M. Safaryan, R. Islamov, X. Qian, and P. Richtárik. Fednl: Making newton-type methods applicable to federated learning. arXiv preprint arXiv:2106.02969, 2021.

[37] C. A. Uribe, D. Dvinskikh, P. Dvurechensky, A. Gasnikov, and A. Nedić. Distributed computation of Wasserstein barycenters over networks. In 2018 IEEE 57th Annual Conference on Decision and Control (CDC), 2018. Accepted, arXiv:1803.02933.

[38] D. P. Woodruff. Sketching as a tool for numerical linear algebra. arXiv preprint arXiv:1411.4357, 2014.

[39] L. Xiao and S. Boyd. Optimal scaling of a gradient method for distributed resource allocation. Journal of Optimization Theory and Applications, 129(3):469–488, 2006.

[40] C. Xie, O. Koyejo, I. Gupta, and H. Lin. Local adaalter: Communicationefficient stochastic gradient descent with adaptive learning rates, 2019.

[41] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. Computer networks, 52(12):2292–2330, 2008.

[42] Y. Zhang and L. Xiao. Communication-Efficient Distributed Optimization of Self-concordant Empirical Loss, pages 289–341. Springer International Publishing, Cham, 2018.

[43] X. Q. Zhize Li, Dmitry Kovalev and P. Richtárik. Acceleration for compressed gradient descent in distributed and federated optimization. International Conference on Machine Learning, 37, 2020.
A Proofs

A.1 Assumptions

Assumption 1. The function $F$ is twice continuously differentiable.

Assumption 2. There exist positive constants $\mu$ and $L$ such that $\mu I \preceq \nabla^2 F(w) \preceq LI$, $\forall w \in \mathbb{R}^d$.

Assumption 3. The function $F$ is bounded below by a scalar $\bar{F}$.

Assumption 4. The gradients of $F$ are $L$-Lipschitz continuous for all $w \in \mathbb{R}^n$.

A.2 Proof of Lemma 1

Lemma 1. Let Assumption 1 hold. Let $A_k$ be defined as in (10) depending on the choice of iterates update. Then, for any $k \geq 0$ there exist constants $0 < \mu_1 \leq \mu_2$ such that

$$\mu_1 I \preceq A_k \preceq \mu_2 I.$$

Proof. Truncated L-SR1 update. In this case $A_k = (|B_{k+1}|^\Omega_\omega)^{-1}$. By spectral decomposition $B_k = V_k A_k V_k^T$. By truncation (Definition 1)

$$\omega I \preceq |A_k|^\Omega_\omega \preceq \Omega I.$$  

Furthermore, $\frac{1}{\Omega} I \preceq A_k \preceq \frac{1}{\omega} I$. Defining $\mu_1 := \frac{1}{\Omega}$ and $\mu_2 := \frac{1}{\omega}$, and noting that $0 < \mu_1 \leq \mu_2$, gives the result.

FedSONIA update. In this case $A_k = (|B_{k+1}|^\Omega_\omega)^{-1} + \rho_k (I - V_k V_k^T)$. By Assumption 4

$$A_k = V_k (|A_k|^\Omega_\omega)^{-1} V_k^T = V_k (|A_k|^\Omega_\omega)^{-1} - \rho_k I V_k^T + \rho_k I \preceq \rho_k I.$$  

By truncation (Definition 1) $\omega I \preceq |A_k|^\Omega_\omega \preceq \Omega I$. Therefore, $A_k \preceq \frac{1}{\omega} I + \rho_k I \preceq \frac{2}{\omega} I$. Defining $\mu_1 := \frac{1}{\Omega}$ and $\mu_2 := \frac{2}{\omega}$, and noting that $0 < \mu_1 \leq \mu_2$, gives the result.

A.3 Proof of Theorem 1

Theorem 1. Let Assumptions 1 and 2 hold. Then for iterates $\{w_k\}$ generated by Algorithm 5 with $0 < \alpha_k = \alpha \leq \frac{\mu_1}{L^2}$ we have

$$F(w_k) - F^* \leq (1 - \alpha \mu_1)^k [F(w_0) - F^*]$$

for all $k \geq 0$.

Proof. We have

$$F(w_{k+1}) = F(w_k - \alpha A_k \nabla F(w_k))$$  

Assumption 4

$$\leq F(w_k) + \nabla F(w_k)^T (\alpha A_k \nabla F(w_k)) + \frac{L}{2} \|\alpha A_k \nabla F(w_k)\|^2$$  

$$\leq F(w_k) - \alpha \mu_1 \|\nabla F(w_k)\|^2 + \frac{\alpha^2 \mu_2^2 L}{2} \|\nabla F(w_k)\|^2$$  

$$= F(w_k) - \alpha \left( \mu_1 - \frac{\mu_2^2 L}{2} \right) \|\nabla F(w_k)\|^2 \leq F(w_k) - \alpha \frac{\mu_1}{2} \|\nabla F(w_k)\|^2,$$

(15)

where the last inequality is due to the choice of the step-size. By Assumption 3 we have $2\mu(F(w) - F^*) \leq \|\nabla F(w)\|^2$. Therefore,

$$F(w_{k+1}) \leq F(w_k) - \alpha \mu_1 (F(w_k) - F^*).$$

Subtracting $F^*$ from both sides,

$$F(w_{k+1}) - F^* \leq (1 - \alpha \mu_1) (F(w_k) - F^*).$$

Recursive application of the above inequality yields the desired result. \qed
A.4 Proof of Theorem 2

We begin with the technical lemma for FLECS with Truncated Hessian inverse Approximation step (Algorithm 3).

Lemma 2. Let Assumptions 1 and 2 hold and 0 ≤ ω ≤ μ, Ω ≥ L. Then for Hessian approximations $B_{k+1}$ we have

$$||B_{k+1}|^\Omega - \nabla^2 f(w^*)||_F \leq ||B_{k+1} - \nabla^2 f(w^*)||_F$$

for all $k \geq 0$.

Proof. By Definition 1,

$$|B_{k+1}|^\Omega := V_{k+1}|A_k|^\Omega V_{k+1}^T,$$

where $B_{k+1} = V_{k+1}A_kV_{k+1}^T$ is spectral decomposition of $B_{k+1}$. Then,

$$|||B_{k+1}|^\Omega - \nabla^2 f(w^*)||_F = ||V_{k+1}|A_k|^\Omega V_{k+1}^T - \nabla^2 f(w^*)||_F$$

$$= ||V_{k+1}^T|A_k|^\Omega V_{k+1} - V_{k+1}^T \nabla^2 f(w^*)V_{k+1}||_F$$

Analogically,

$$||B_{k+1} - \nabla^2 f(w^*)||_F = ||\lambda_k - \lambda_k V_{k+1}^T \nabla^2 f(w^*) V_{k+1}||_F.$$

Now, let us denote $\lambda_i, i = 1, \ldots, d$ as eigenvalues of $B_{k+1}$ (diagonal elements of $\Lambda_{k+1}$). Let $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$ be corresponding diagonal elements of $|\Lambda_{k+1}|^\Omega$.

Noting, that

$$0 \leq \omega I \leq \nabla^2 F(w^*) \leq LI \leq \Omega I,$$

we have $|\tilde{\lambda}_i - h_i| \leq |\lambda_i - h_i|$ for all $i = 1, \ldots, n$, where $h_i = [V_{k+1}^T \nabla^2 F(w^*) V_{k+1}]_{ii}$.

Therefore,

$$|||B_{k+1}|^\Omega - \nabla^2 f(w^*)||_F \leq ||B_{k+1} - \nabla^2 f(w^*)||_F$$

by definition of Frobenius norm and the fact that matrices $|B_{k+1}|^\Omega - \nabla^2 f(w^*)$ and $\Lambda_k - \lambda_k V_{k+1}^T \nabla^2 f(w^*) V_{k+1}$ differ only in diagonal elements.

Theorem 2. Let Assumptions 1 and 2 hold, and $\|w^0 - w^*\| \leq \frac{\mu^2}{2} \sum_{i=1}^{n} \|B_{k+1} - \nabla^2 f_i(w^*)\|_F \leq \frac{\mu^2}{2}.$

Then for iterates $\{w_k\}$ generated by Algorithm 3 with Truncated inverse Hessian approximation step (Algorithm 3) with parameters $\alpha_k = \alpha = 1, 0 \leq \omega \leq \mu, \Omega \geq L$ we have

$$\|w_k - w^*\|^2 \leq \frac{1}{\Omega^2} \|w_0 - w^*\|^2$$

for all $k \geq 0$.

Proof.

$$\|w_{k+1} - w^*\|^2 = \|w_k - w^* - A_k \nabla F(w_k)\|^2 \leq \|A_k\|^2 \|A_k^{-1}(w_k - w^*) - \nabla F(w_k)\|^2$$

$$\leq \mu^2 \|A_k^{-1}(w_k - w^*) - \nabla F(w_k)\|^2$$

$$\leq 2\mu^2 \left( \left( A_k^{-1} - \nabla^2 F(w^*) \right) (w_k - w^*) \right) \leq \nabla^2 F(w^*)(w_k - w^*) - \nabla F(w_k) + \nabla F(w^*) \right)$$

Assumption 3.

$$\leq 2\mu^2 \left( \left( A_k^{-1} - \nabla^2 F(w^*) \right) \|w_k - w^*\|^2 + \frac{L^2}{4} \|w_k - w^*\|^2 \right)$$

$$\leq 2\mu^2 \left( \left| A_k^{-1} - \nabla^2 F(w^*) \right| \|w_k - w^*\|^2 + \frac{L^2}{4} \|w_k - w^*\|^2 \right)$$

$$\leq 2\mu^2 \left( \left| B_{k+1} \right|^\Omega - \nabla^2 F(w^*) \right) \|w_k - w^*\|^2 + \frac{L^2}{4} \|w_k - w^*\|^2$$

Lem. 3.

$$\leq 2\mu^2 \left( \left| B_{k+1} - \nabla^2 F(w^*) \right| \|w_k - w^*\|^2 + \frac{L^2}{4} \|w_k - w^*\|^2 \right)$$

Lem. 3.
Therefore, by assumptions of the theorem
\[ \|w_{k+1} - w^*\|^2 \leq \frac{1}{\mu^2} \|w_k - w^*\|^2 \left( \|B_{k+1} - \nabla^2 F(w^*)\|^2 + \frac{L^2}{4} \|w_k - w^*\|^2 \right) \leq \frac{1}{2} \|w_k - w^*\|^2 \]

\[ \Box \]

### A.5 Proof of Theorem 3

**Theorem 3.** Let Assumptions 1, 2, 4 hold and \( w_0 \) be the starting point. Then after \( T > 0 \) iterations of Algorithm 5 with \( 0 < \alpha_k = \alpha \leq \frac{\mu}{\mu^2} \), we have
\[ \frac{1}{T} \sum_{k=0}^{T-1} \|\nabla F(w_k)\|^2 \leq \frac{2(F(w_0) - F)}{\alpha \mu_1} \xrightarrow{T \to \infty} 0. \]

**Proof.** By (15)
\[ F(w_{k+1}) \leq F(w_k) - \frac{\mu_1}{2} \|\nabla F(w_k)\|^2. \]

Summing both sides of the above inequality from \( k = 0 \) to \( T - 1 \), we obtain
\[ \hat{F} - F(w_0) \leq F(w_T) - F(w_0) = \sum_{k=0}^{T-1} (F(w_{k+1}) - F(w_k)) \leq \sum_{k=0}^{T-1} \frac{\mu_1}{2} \|\nabla F(w_k)\|^2. \]

Therefore, we have
\[ \sum_{k=0}^{T-1} \|\nabla F(w_k)\|^2 \leq \frac{2(F(w_0) - \hat{F})}{\alpha \mu_1}. \]

(16)

Dividing (16) by \( T \) we conclude
\[ \frac{1}{T} \sum_{k=0}^{T-1} \|\nabla F(w_k)\|^2 \leq \frac{2(F(w_0) - \hat{F})}{\alpha \mu_1 T}. \]

\[ \Box \]

### B Additional Numerical Experiments

#### B.1 Top-K Compressor

In this experiment we used Top-K compression with \( k = 4d \) on a9a dataset with regularization parameter \( \mu = 10^{-5} \). We compare FedNL [36] and FLECS in the number of hessian-vector products, since compression cost is roughly the same. For FLECS we set memory size \( m = 16 \) and step-size \( \alpha = 1 \). For both algorithms we set \( B_0^n = \nabla F(w_0) \) and Hessian approximation rate \( \beta = 1 \). In this experiment we omit initialization cost (computation of the full Hessian at starting point) for both algorithms.

![Figure 2: Comparison of objective function \( F(w_k) \) and the squared norm of gradient \( \|\nabla F(w_k)\|^2 \) between FLECS and FedNL.](image-url)
B.2 Comparison with DIANA and ADIANA.

In order to compare our framework with DIANA and ADIANA we use random dithering with number of levels \( s = \sqrt{\beta} \) and \( p = \infty \). For FLECS we set hyperparameters as: \( m = 1, \beta = 1, \alpha = 1 \) and initialize Hessian approximations with 0.

For DIANA and ADIANA we used theoretical parameters \( L = 1/4 \) and strong convexity constant \( \mu = 10^{-5} \). For these methods we used an original code provided by authors [27, 43].

![Graph](image1)

Figure 3: Comparison of objective function \( F(w_k) \) and the squared norm of gradient \( ||\nabla F(w_k)||^2 \) between DIANA, ADIANA and FLECS.

B.3 Top-K vs Random Dithering

In this subsection we want to compare Top-K compressor with random dithering for FLECS. We set hyperparameters as: \( m = 16, \beta = 1, \alpha = 1 \) and initialize Hessian approximations with \( \nabla^2 f(w_0) \).

We use Top-K with \( k = 4d \) and random dithering with \( s = 128 \) and \( p = \infty \), so communication cost per iteration is roughly the same. In this experiment we omit initialization cost (computation of the full Hessian at starting point) for both algorithms.

![Graph](image2)

Figure 4: Comparison of objective function \( F(w_k) \) and the squared norm of gradient \( ||\nabla F(w_k)||^2 \).

B.4 Dependence on the Memory Size

Here we compare different memory sizes for FLECS on a9a dataset. We use random dithering with parameters \( s = 128 \) and \( p = \infty \). Hyperparameters set as \( \beta = 1, \alpha = 1 \) and initialize Hessian approximations initialized with 0.
C Compressors

FLECS allows several different compression operators. In this section we give definitions and provide examples of possible compressors for our framework.

C.1 Unbiased Compressors.

**Definition 2.** A random operator $C : \mathbb{R}^{d \times m} \to \mathbb{R}^{d \times m}$ satisfying
\[
E_Q[C(X)] = X, \quad E_C[\|C(X)\|_F] \leq (\omega + 1)\|X\|_F^2, \tag{17}
\]
for all $X \in \mathbb{R}^{d \times m}$ is a unbiased compressor operator.

Random dithering for vectors ($\omega$-quantization) [2], applied to each column of the matrix, is an example of unbiased compressors that we used in our experiments.

C.2 Biased compressors.

**Definition 3.** A random operator $C : \mathbb{R}^{d \times m} \to \mathbb{R}^{d \times m}$ satisfying
\[
E_C[\|C(X)\|_F] \leq \|X\|_F, \quad E_C[\|C(X)\|_F^2] \leq (1 - \delta)\|X\|_F^2, \tag{18}
\]
for all $X \in \mathbb{R}^{d \times m}$ is a contractive compressor operator.

In our experiments we use Top-$K$ as a contractive compressor.