HANF: Hyperparameter And Neural Architecture Search in Federated Learning

Jonas Seng¹, *  Pooja Prasad¹  Devendra Singh Dhami¹, ³  Kristian Kersting¹, ², ³

¹ Computer Science Department, TU Darmstadt
² Centre for Cognitive Science, TU Darmstadt
³ Hessian Center for AI (hessian.AI)

*correspondence: jonas.seng@tu-darmstadt.de

Abstract

Automated machine learning (AutoML) is an important step to make machine learning models being widely applied to solve real world problems. Despite numerous research advancement, machine learning methods are not fully utilized by industries mainly due to their data privacy and security regulations, high cost involved in storing and computing increasing amount of data at central location and most importantly lack of expertise. Hence, we introduce a novel framework, HANF - Hyperparameter And Neural architecture search in Federated learning as a step towards building an AutoML framework for data distributed across several data owner servers without any need for bringing the data to a central location. HANF jointly optimizes a neural architecture and non-architectural hyperparameters of a learning algorithm using gradient-based neural architecture search and \( n \)-armed bandit approach respectively in data distributed setting. We show that HANF efficiently finds the optimized neural architecture and also tunes the hyperparameters on data owner servers. Additionally, HANF can be applied in both, federated and non-federated settings. Empirically, we show that HANF converges towards well-suited architectures and non-architectural hyperparameter-sets using image-classification tasks.

1 Introduction

Federated Learning (FL) [McMahan et al. 2017] is a distributed machine learning method where a shared model learns on data distributed at different locations and share the locally computed model updates to a central server for aggregation to achieve the learning objective. Federated machine learning is a promising solution in several industries where its not feasible to share the data due to privacy and security regulations such as finance, defence and healthcare, to name a few. This can enable institutions to build a collective model while keeping their data at their own servers. Currently, most of the federated learning is limited to manually predefining the global model architecture that learns across several data owners [Konečný et al. 2016, Liang et al. 2020]. However, the choice of hyperparameters, both architectural (number of layers etc.) and non-architectural (learning rate etc.), are likely to be non-optimal since these values are selected based on human experiences only [Kairouz et al. 2021]. Basic approaches like Random Search or Successive Halving are often infeasible because training multiple models to convergence requires a lot of possibly limited resources, especially when it comes to FL.

In recent years, sophisticated methods to tune the hyperparameters of a neural network have been developed. Several approaches to perform (efficient) Neural Architecture Search (NAS) have been proposed [Pham et al. 2018, He et al. 2022, Liu et al. 2018, Zoph and Le 2016]. Similarly work

Preprint. Under review.
Figure 1: HANF jointly optimizing hyperparameters and neural architecture search using data distributed across $C$ number of data owning clients. The search stage optimizing hyperparameters and architecture search alternatively by computing the reward and evaluation stage then using the best architecture, finds the high-performing hyperparameters.

has been done to tune non-architectural hyperparameters for training neural networks [Khodak et al. 2021], [Agrawal et al. 2021a]. However, most of these techniques focus on just optimizing either the architecture or the non-architectural hyperparameters, thus omitting the underlying dependencies between them. As an example, for a deeper architecture it might be necessary to select a higher learning rate in order to increase the gradient signal arriving at early layers of the network when back-propagating. Additionally, the methods mentioned above focus only on ML-scenarios where data is stored on one machine. Since, in real-world scenarios more and more data are stored in a distributed fashion, techniques like neural architecture search and hyperparameters optimization have to be employed in federated settings to enable non-experts to efficiently build high-performing neural network even when data is not stored in a centralized server. Thus, in order to take an important step towards end-to-end AutoML solutions, dependencies between the choice of architectural and non-architectural hyperparameters have to be considered and respected.

To overcome the above mentioned limitations, we propose a novel framework, HANF - Hyperparameter And Neural architecture search in Federated learning. The overall architecture of HANF as shown in Figure 1 consists of two stages: search and evaluation. The search stage performs hyperparameter optimization and neural architecture search alternatively. First, a $n$-armed bandit game is played in which the fixed neural architecture and the model weights reflect the environment giving a reward and the non-architectural hyperparameters (hyperparameters from now on) of a discrete set of hyperparameters correspond to actions an agent can choose from. The reward is computed based on the validation loss of the network before and after a training step under some chosen hyperparameters. In a second phase the most promising hyperparameters are fixed for a certain number of training-steps and differentiable, cell-based NAS is performed as proposed in the DARTS framework [Liu et al. 2018]. The differentiability of DARTS is exploited to allow for "averaging" architectures of several clients in a FL-setting using FedAvg. By "averaging" the architectures, a new global architecture is obtained which is then used for further training. In our work, different clients collaboratively search for a global architecture by exchanging gradients in each iteration, where the convergence is naturally guaranteed. The evaluation stage then uses the best cell-architectures found to build up a bigger network using these cells. This network is trained from scratch using the same data. The $n$-armed bandit approach is used to find promising hyperparameters during training. We make our code publicly available at: [https://anonymous.4open.science/r/HANF-BD92/](https://anonymous.4open.science/r/HANF-BD92/)
Table 1: Here we show methods trying to employ HO/NAS in centralized and/or FL-settings. HANF is the first method that can do both HO and NAS in a federated learning setting. Here, Fed means Federated and Cent means Centralized.

| Method           | Neural Architecture Search | Hyperparameter Optimization | Data Setting |
|------------------|---------------------------|-----------------------------|--------------|
| DARTS Liu et al. [2018] | ✓                         | x                           | Cent         |
| DP-FNAS Singh et al. [2020] | ✓                         | x                           | Fed          |
| FedNAS He et al. [2022]     | ✓                         | x                           | Fed          |
| DP-FTS-DE Dai et al. [2021] | x                         | ✓                           | Fed          |
| FedEx Khodak et al. [2021]  | x                         | ✓                           | Fed          |
| **HANF**          | ✓                         | ✓                           | Fed+Cent     |

Overall, we make the following important contributions:

C1. **We propose a novel framework, HANF, that jointly optimizes the architectural and non-architectural hyperparameters of a neural network in data distributed setting and can be used in classical centralized data setting as well (See Tab. 1).**

C2. **HANF enables multiple data owners to collaboratively tune hyperparameters and search for a high performing architecture with data residing at each participating data owners side.**

C3. **We empirically show that HANF converges towards well-suited architectural and non-architectural hyperparameters using image-classification tasks.**

2 Related Work

**Hyperparameter Optimization.** Several works address hyperparameter optimization in federated learning Koskela and Honkela [2018], Mostafa [2019]. DP-FTS-DE Dai et al. [2021] integrated differential privacy into federated Thompson sampling with distributed exploration to preserve privacy and used it for federated hyperparameter tuning. Genetic CFL Agrawal et al. [2021b] clustered edge devices based on the training hyperparameters and genetically modified the parameters clusterwise. FLoRA Zhou et al. [2021] used a single-shot task by querying each client for several configurations once and select the best configuration based on what the clients returned w.r.t. learning progress. In contrast, HANF follows a few-shot policy as we adjust the hyperparameters several times during training. FedEx Khodak et al. [2021] used weight-sharing method of neural architecture search for hyperparameter optimization in the federated setting. All these works focus on non-architectural parameters optimization.

**Neural Architecture Search.** NAS is a well-researched technique for automatically searched optimal neural architecture based on reinforcement learning (RL) Zoph and Le [2016], Zoph et al. [2016, 2018], evolutionary algorithms (EAs) Xie and Yuille [2017], Galván and Mooney [2021], Darwish et al. [2020], and gradient based (GD) methods Liu et al. [2018], Dong and Yang [2019], Xie et al. [2018], Li et al. [2020]. Gradient based methods have been found to be more robust compared to the former Zhu et al. [2021] and thus we adapt gradient-based neural architecture search in HANF. Since this approach is differentiable it can be easily used in federated setting. In recent years, several research can be seen in federated NAS. Fed-NAS He et al. [2022] uses the gradient-based NAS method MiLeNAS He et al. [2020] for personalized federated learning. DP-FNAS Singh et al. [2020] also adopted differentiable search strategy Liu et al. [2018] combined with differential privacy.

As per our knowledge, our proposed framework HANF is the first work that has addressed neural architecture search and hyperparameter optimization in a single algorithm using gradient based architecture search and n-armed bandit approach in federated learning.

3 HANF

The main contribution of HANF framework is that it allows to efficiently tune for neural architecture and non-architectural hyperparameters jointly. Additionally, HANF is made applicable for FL-settings by employing differentiable NAS and FedAvg. We first define our problem setup in detail before presenting our proposed solution.
3.1 Problem Definition

We consider a federated learning setting with a set of clients \( C \) of size \( C \), each holding a dataset \( D_1, \ldots, D_C \). The data of each client \( c \) is split into training \( (X_{\text{train}}^{(c)}, y_{\text{train}}^{(c)}) \) and validation data \( (X_{\text{val}}^{(c)}, y_{\text{val}}^{(c)}) \) and is used to solve a supervised learning task. The goal is to obtain a global neural network by choosing an appropriate architecture \( a \) from a space of architectures \( A \) and appropriate non-architectural hyperparameters \( h \) from a discrete set of non-architectural hyperparameters \( H \) which minimize the network’s validation loss over all clients once the network is trained, thus solving the supervised learning task. That is, we aim to solve the following bi-level optimization problem:

\[
\min_{a, h} \sum_{c \in C} v_c \cdot L_{a, h}(w^*, X_{\text{val}}^{(c)}, y_{\text{val}}^{(c)})
\]

where

\[
w^* = \arg\min_w \sum_{c \in C} v_c \cdot L_{a, h}(w, X_{\text{train}}^{(c)}, y_{\text{train}}^{(c)})
\]

Note that since we are in a FL setting, the global validation loss is a weighted sum of the client’s local validation losses. Each client’s weight \( v_c \) is defined as the fraction of validation data a client \( c \) holds:

\[
v_c := \frac{|X_{\text{val}}^{(c)}|}{\sum_{c \in C} |X_{\text{val}}^{(c)}|}
\]

We will refer to the global validation loss as \( L_{a, h}(w, X_{\text{val}}, y_{\text{val}}) \). Also, we will use the terms non-architectural hyperparameters and hyperparameter-configuration interchangeably from now on.

3.2 HANF Architecture

The main observation leading to HANF is that one can exploit differentiable architecture search methods to employ a federated version of architecture search via parameter averaging. Additionally, the use of parameter sharing can be exploited to perform informed updates of non-architectural hyperparameters during the learning process. Overall, HANF is composed of two stages:

1. Search stage: aims to identify a high-performing cell-based architecture using which a model is constructed that will be employed in the evaluation stage. The search stage is split into two phases, an HO and a NAS phase. HANF starts by initializing model parameters \( w \) and architectural parameters \( a \). \( w \) can be initialized using any arbitrary initialization method while \( a \) is set to 0 in order to give each architecture equal weight at the beginning of training. Additionally it initializes a discrete non-architectural search space \( H \) as well as list of reward-estimates \( r \) of size \( |H| \). The initialization takes place on server-side. After initialization HANF performs HO-phases \( e^{(1)}, \ldots, e^{(n)} \) and NAS-phases \( i^{(1)}, \ldots, i^{(n)} \) in an alternating fashion, i.e. the search stage. Each HO- and NAS-phase takes place for a certain number of communication rounds, denoted \( |e^{(j)}| \) and \( |i^{(j)}| \) respectively. The goal of the HO-phase is to identify a promising hyperparameter-configuration \( h \in H \) for the subsequent NAS-phase. During an HO-phase the weight(s) of the model and its architecture is fixed. The goal of the NAS-phase is to learn a promising architecture which minimizes the averaged validation loss.

2. Evaluation stage: Here, the cell-architecture found in the previous stage is used to build an evaluation model. While training this model, we employ HO in order to identify well-suited non-architectural hyperparameters optimizing the global validation loss.

We now describe the HO and NAS phase in detail.

**Hyperparameter Optimization (HO).** The HO-phase tries to identify a good set of non-architectural hyperparameters for the subsequent NAS-phase. This is tackled by using weight sharing and by considering selection of a good set of hyperparameters as a \( n \)-armed bandit problem. Optimizing for hyperparameter-configuration \( h \) in general can be stated as:

\[
h^* = \arg\min_h L_{a, h}(w^*, X, y)
\]

Here, \( w^* \) are the model- and architecture-weights minimizing the loss under hyperparameters \( h \in H \) and a fixed architecture \( a \in A \). \( X \) denotes the data that should be used to train to predict labels \( y \).
The space of network-architectures $\mathcal{A}$ is represented as a supernet. In our case, a supernet is a DAG describing multiple cell-architectures at once using weighted edges. Cells will be defined in the next paragraph. Since training a supernet to convergence for each configuration of hyperparameters is infeasible even for small hyperparameter-spaces, we use one communication round of training as an approximation for $w^*$ similar to DARTS [Liu et al. 2018]. This is done by randomly sampling $m$ hyperparameter-configurations from a distribution over the discrete set of hyperparameters $\mathcal{H}$. Each sampled configuration is then used to perform one communication round of training using the same weights $w$ and architecture $a$ of the supernet. Each client computes its local validation loss before and after performing one training step in each communication round, denoted as $r_{a,w}^c$ and $r_{a',w'}^c$ respectively. These are used to compute a reward-signal $r_{h}^e$ for using hyperparameter-configuration $h$ in HO-phase $e$ to perform one training-step:

$$r_{h}^e = \sum_{c \in C} v_c \cdot (r_{a,w}^c - r_{a',w'}^c)$$

This is done for each hyperparameter-configuration $h$ sampled in one exploration round, i.e. we obtain a vector $r^{(e)}$. Each entry in this vector corresponds to one hyperparameter-configuration in $\mathcal{H}$, thus in $r^{(e)}$ all entries that correspond to a sampled hyperparameter-configuration contain the reward, all other entries are set to zero. The rewards obtained in round $e$ are then used to update the reward-estimates $r$ by, similar to action-value methods known from Reinforcement Learning [Sutton and Barto 2018]:

$$r = r + (i \circ \alpha \cdot (r^{(e)} - r)) + ((1 - i) \circ \alpha \cdot r)$$

Here, $\circ$ is the Hadamard product, $i$ is a binary vector indicating which hyperparameter-configurations were sampled in exploration round $e$ and $\alpha$ is a constant factor determining how aggressively the reward-history should be updated. If a hyperparameter-configuration was sampled at round $e$, its reward in $r$ will be corrected by the error of the current reward estimate. All configurations that were not sampled in $e$ are decreased by $\alpha$ since configurations that were good in an early stage of training might not be suitable in later stages anymore.

The use of the reward estimates is three-fold: Firstly, the hyperparameter-configuration with the highest reward achieved so far will be used to train the supernet in the next NAS-phase. Secondly, it is used to determine the number of communication-rounds in the HO-phase before the HO-phase is starting by computing:

$$\pi = \text{softmax}(r)$$

$$\epsilon = \sum_{h \in H} \ln(\pi(h)) \cdot \pi(h)$$

$$\kappa = \text{rnd}(\beta \epsilon)$$

Here, $\text{rnd}$ is the round-function, $\beta$ is a constant scaling factor and $\epsilon = |e^{(j)}|$ holds. In the beginning, all rewards are set to 0, thus leading to a uniform distribution which has the maximum entropy. Once the rewards indicate which hyperparameter-configurations lead to high rewards and which do not, $\pi$ will not be uniform anymore and thus the entropy will decrease over time, leading to less exploration and more exploitation in later training stages. The third use of the rewards is that the distribution $\pi$, which is computed based on the rewards, is used to sample the next hyperparameter-configurations tested in an exploration phase from the hyperparameter-space.

**Neural Architecture Search.** Once we have obtained a hyperparameter-configuration $h$ from the HO-phase, we will use $h$ to train our supernet for a predefined number of communication rounds using differentiable NAS. Performing training over multiple communication rounds is useful to speed up training. One could possibly set the number of communication rounds in the NAS-phase to 1, however this will slow down training significantly since we would then perform the costly search for hyperparameters most of the time instead of using a good configuration $h$ to train the supernet. The objective we want to minimize is:

$$\min_{a} \mathcal{L}_{a,h}(w^*, X_{val}, y_{val})$$

where $w^* = \arg \min_{w} \mathcal{L}_{a,h}(w, X_{train}, y_{train})$

This is another bi-level optimization problem which we aim to solve following the approach from [Liu et al. 2018]. That is, we define our search space to be a space over cells. A cell is a Directed Acyclic
Graph (DAG) in which each node is a feature representation and each edge is a mixed operation. The feature representation of some node \( z \) is computed using all its parent-nodes and the mixed operations defining the edges between \( z \) and its parent, i.e. for some node \( z_j \) the representation is computed as: \( z_j = \sum_{i < j} o^{(z_i, z_j)}(x^{z_i}) \) Here, \( o^{(z_i, z_j)} \) is a mixed operation and \( x^{z_i} \) is the feature representation of node \( z_i \). A mixed operation connecting nodes \( z_1 \) and \( z_2 \) is defined as a weighted sum over a set of operations \( O: o^{(z_1, z_2)} = \sum_{o \in O} \frac{\exp(a_o(z_1, z_2))}{\sum_{o' \in O} \exp(a_{o'}(z_1, z_2))} o(x) \) Here, \( a_o(z_1, z_2) \) are the architectural parameters to be learned. Since they fully describe the architecture, we will refer to them as the architecture \( a \) from now on.

Objective \( [10] \) is solved by an alternating optimization of the architecture and model parameters. First, the architecture is updated by following the gradient \( \nabla_w L_{a, h}(\hat{w}, X_{\text{train}}, Y_{\text{train}}) \) where \( \hat{w} = w - \eta \nabla_w L_{a, h}(w, X_{\text{train}}, Y_{\text{train}}) \). As a second step the model parameters are updated by following the gradient \( \nabla_w L_{a, h}(w, X_{\text{train}}, Y_{\text{train}}) \).

The above is performed on client-side in each communication round and yields new architectural and model parameters \( a'_c \) and \( w'_c \) for each client \( c \) respectively. Since both, the architectural and model parameters, are parameters of a non-convex optimization problem with a differentiable loss-function which is optimized using gradient descent methods, we can use FedAvg to perform optimization in a federated learning setting.

We use two types of cells: Normal cells and reduction cells. Normal cells keep the dimensions of data such that a fraction of clients holds two times more samples with some label which is optimized using gradient descent methods, we can use FedAvg to perform optimization in a federated learning setting.

Discretizing the Architecture. Since differentiable NAS requires a continuous relaxation of the architecture space \( A \), we have to discretize the architecture learned by HANF after training. In this work we make the cell-architecture discrete by selecting the top \( k \) operations with the highest associated architectural weight over all cells. An additional restriction ensures that no operation connects the same two nodes.

4 Experiments and Results

In our experiments we applied HANF to solve an image classification task on FashionMNIST and CIFAR-10. Both datasets were partitioned randomly on a set of clients such that each client holds the same number of samples with a certain tolerance. Since in federated learning it is common to have data unequally distributed across clients, we also conducted experiments in which we introduced a label skew in the data (referred to as non-i.i.d. subsequently) Kairouz et al. [2021]. In this case we distributed data such that a fraction of clients holds two times more samples with some label \( y \) than all the other clients. We employed two stages in the experiments. First, we aim to train a supernet defining a search space over CNN-architectures and to find proper hyperparameters to train this supernet. In a second step we use the best normal cell and reduction cell obtained in the search-stage to build up a larger CNN. For discretizing the architecture \( k = 2 \) was chosen in order to be comparable to other cell-based NAS-approaches. We then apply HO again on the bigger network with a fixed architecture found in Search stage. The results of the bigger network were then reported. Although in our experiments each client participated on training in each communication round, however, HANF is not restricted to such scenarios.

4.1 Search Space

Search Stage. Our search space is divided into an architecture search space \( A \) and a non-architectural hyperparameter-search space \( \mathcal{H} \). \( A \) is defined as the space of cells consisting of 7 nodes. The nodes are connected by a mixed operation consisting of the following primitives: Separable and dilated separable convolutions of size \( 3 \times 3 \) and \( 5 \times 5 \), max- and average pooling of size \( 3 \times 3 \), an identity operation as well as a zero-operation. We used a stride of one if applicable and used padding. Our convolutional operations are defined using the ReLU-Conv-BN order and we apply separable convolutions twice as in Liu et al. [2018].
Algorithm 1: HANF Framework server side

Require: set of clients $C$, client weight $v_c \forall c \in C$

Require: hyperparameter-space $H$

initialize parameters $w$ and architecture $a$

initialize reward estimates $r \leftarrow 0$

$P \leftarrow \text{softmax}(r)$

for $p$ in phases do
    if $p == \text{‘ho’}$ then:
        sample $n$ hyperparameters $h$ from $P$
        $r_p \leftarrow 0$
        for $h$ in $h$ do
            $l_1, l_2, w^*, a^* \leftarrow \text{client_steps}(h, w, a)$
            $r_p[h] \leftarrow \sum_{c \in C} v_c \cdot (l_1[c] - l_{c}(c))$
        end for
        $r \leftarrow \text{update_rewards}(r, r_p)$
        $P \leftarrow \text{softmax}(r)$
    end if

    if $p == \text{‘nas’}$ then:
        $h^* \leftarrow H[\arg \max_h r[h]]$
        for $j$ in nas_steps do
            $w, a \leftarrow \text{client_steps}(h^*, w, a)$
        end for
    end if
end for

We stack 8 cells s.t. the input of each cell is the output of its last two predecessors. There are normal cells as well as reduction cells. The output of a normal cell keeps the dimensions of the input whereas reduction cells reduce the input’s dimensions. The very first cell receives the input twice. The output of each cell is defined as the depth-wise concatenation of the representations of its nodes. Reduction cells are placed after $1/3$ and $2/3$ of the total network depth.

The hyperparameter-space $H$ consists of candidates for the learning rate used for model- and architecture parameter-updates sampled from a log-uniform distribution from $10^\exp(U(−4,0))$ and $10^\exp(U(−5,−1))$ respectively. Further we included candidates for weight decay used in updates of both parameter-types sampled from $10^\exp(U(−5,−1))$ and we included candidates for the momentum used in SGD-updates of model parameters with momentum candidates sampled from $U(0, 1)$. We used 120 i.i.d. samples from these distributions to define our discrete hyperparameter search space.

Evaluation Stage. Here, our search space consists of non-architectural hyperparameters only. We chose to tune for the learning rate, weight decay, momentum and path-dropout. We chose the first three hyperparameters since these directly influence the behavior of the optimizer, thus of the learning progress. Path-dropout was chosen to avoid over-fitting and to demonstrate that HANF is able to select proper configurations to do so. We again sample 120 i.i.d. samples as described above, for path-dropout we chose to sample from $U(0,0.3)$.

4.2 Results

We show that HANF, while performing both HO and NAS, achieves near-state of the art results compared to DARTS in FL settings and FedEx on FashionMNIST and CIFAR-10. Even in label-skew scenario, HANF performs reasonably well and introducing label skews does not seem to adversely affect its performance (See Table 2). HANF finds high-performing cell-architectures and hyperparameter-configurations in FL-settings without adding significant overhead compared to DARTS (See Table 3).

It should further be noted in case of CIFAR-10, the accuracy-scores are negatively correlated with the number of clients the data is distributed on. This is not surprising since we used the same search space in all our experiments. We hypothesize that the performance-drop is only observed for CIFAR-10 because it contains colored images while FashionMNIST contains only black & white images. Thus,
We observe that HANF chooses more “cautious” hyperparameters than engineers usually do. For Table 3: This table shows the runtime in GPU-days (d) or GPU-hours (h) of each algorithm. For DARTS and HANF we report the GPU-days needed to complete one search-stage (120 communication rounds) and one evaluation-stage (1500 communication rounds). All experiments were run on Nvidia A100-GPUs. It can be seen that HANF adds little overhead on top of DARTS. FedEx is the fastest algorithm, however note that FedEx does not perform architecture search.

Table 2: This table compares the performance of the models found by DARTS, DARTS federated, FedEx and HANF in terms of classification-accuracy. Each method was performed 3 times per dataset and number of clients. We reported the mean accuracy and standard deviation. We see that HANF is able to achieve near-state of the art results for FashionMNIST and CIFAR-10. The best results are shown in bold, centralized results are not considered in selecting the best method.

The same model (having the same capacity) has to learn a higher amount of information, potentially leading to a drop in performance.

Figure 3 shows the hyperparameters selected by HANF over time for a single run on CIFAR-10. We observe that HANF chooses more “cautious” hyperparameters than engineers usually do. For example, in DARTS it’s common to start with a learning rate of 0.025, HANF however chooses much lower learning rates most of the time. Additionally, the choice of the momentum seems to not have high influence on the learning behavior since the choices are distributed nearly uniform over possible configurations of the momentum. Another interesting observation is that HANF is self-correcting as it recovers once it has chosen bad hyperparameter-configurations for a certain number of training steps. Figure 2 illustrates this effect using the performance over time of the model trained by HANF.

5 Discussion

Societal Impact. HANF is a step towards building AutoML solutions and using it on real world datasets to solve real time problems in industries like finance, defence, healthcare etc. can be exciting. However, we acknowledge that automating the critical systems could be risky. While our research is important for automating the task of hyperparameter optimization and efficiently searching for high-performing neural architecture, the framework would still require human in the loop to verify the final outcome.

Limitations. While searching for a neural architecture and hyperparameters at the same time, HANF still has limitations. Regarding the architecture search, we currently are restricted to cell-based architectures. Thus users still have to define a basic structure of the network (i.e. the supernet). In terms of non-architectural hyperparameters we currently require a discrete search space, thus not reflecting the continuous nature of some hyperparameters like learning rates.
Figure 3: This plot shows configurations of the hyperparameters on the vertical axes and each line corresponds to one choice of hyperparameters at a communication round. The color-scheme of which line corresponds to early or late rounds is depicted on the right. It can be seen that HANF prefers small learning-rates and weight-decay for both, model and architectural parameters. However, the choice of momentum does not seem to be important for HANF since the lines are distributed rather uniformly over possible momentum-configurations. The experiment was done on CIFAR-10.

Conclusion. We have shown that it is possible to efficiently optimize both architectural and non-architectural hyperparameters at the same time in federated learning which are crucial under the framework of AutoML [Ripley 1993], [King et al. 1995], [Kohavi and John 1995]. Our method can also be applied in classical settings where data is stored on one machine. By employing a differentiable NAS algorithm we are able to use FedAvg for efficient architecture search while a $n$-bandit approach was used to obtain estimations of the effect of certain hyperparameter-configurations in order to pick promising configurations. The $n$-bandit approach was used during the architecture search phase and the architecture validation phase. Our results show that HANF is able to compete with existing NAS-methods like DARTS while optimizing for additional non-architectural hyperparameters.

Future Work. Integrating privacy preserving techniques like differential privacy (DP) or secure multi-party computation (SMPC) with HANF could be a potential further development. Extending and improving HANF can also be considered in future work. For example, the non-architectural hyperparameter-space can be extended by a continuous component. Currently, it is required to discretize continuous search spaces which is susceptible for losing optimal configurations during discretization of the search space. Additionally, HANF can be applied and adapted on tasks other than image classification, for example language modelling.

Acknowledgements

This work was supported by the ICT-48 Network of AI Research Excellence Center "TAILOR" (EU Horizon 2020, GA No 952215), the Collaboration Lab "AI in Conctruction" (AICO), the safeFBDC project and the project National High-Performance Computing for Computational Engineering Sciences (NHR4CES).

References

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Artificial intelligence and statistics, pages 1273–1282. PMLR, 2017.

Jakub Koneˇcn`y, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. arXiv preprint arXiv:1610.02527, 2016.

Paul Pu Liang, Terrance Liu, Liu Ziyin, Nicholas B Allen, Randy P Auerbach, David Brent, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think locally, act globally: Federated learning with local and global representations. arXiv preprint arXiv:2001.01523, 2020.
Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018. URL http://arxiv.org/abs/1802.03268.

Chaoyang He, Erum Mushtaq, Jie Ding, and Salman Avestimehr. FedNAS: Federated deep learning via neural architecture search, 2022. URL https://openreview.net/forum?id=10HZX4YDqHT.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* 2018.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* 2016.

Mikhail Khodak, Renbo Tu, Tian Li, Liam Li, Maria-Florina F Balcan, Virginia Smith, and Ameet Talwalkar. Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. *Advances in Neural Information Processing Systems*, 34, 2021.

Shaashwat Agrawal, Sagnik Sarkar, Mamoun Alazab, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu, and Quoc-Viet Pham. Genetic CFL: optimization of hyper-parameters in clustered federated learning. *CoRR*, abs/2107.07233, 2021a. URL https://arxiv.org/abs/2107.07233.

Ishika Singh, Haoyi Zhou, Kunlin Yang, Meng Ding, Bill Lin, and Pengtao Xie. Differentially-private federated neural architecture search. *arXiv preprint arXiv:2006.10559* 2020.

Zhongxiang Dai, Bryan Kian Hsiang Low, and Patrick Jaillot. Differentially private federated bayesian optimization with distributed exploration. *Advances in Neural Information Processing Systems*, 34, 2021.

Antti Koskela and Antti Honkela. Learning rate adaptation for federated and differentially private learning. *arXiv preprint arXiv:1809.03832* 2018.

Hesham Mostafa. Robust federated learning through representation matching and adaptive hyper-parameters. *arXiv preprint arXiv:1912.13075* 2019.

Shaashwat Agrawal, Sagnik Sarkar, Mamoun Alazab, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu, and Quoc-Viet Pham. Genetic cfl: Hyperparameter optimization in clustered federated learning. *Computational Intelligence and Neuroscience*, 2021, 2021b.

Yi Zhou, Parikshit Ram, Theodoros Salonidis, Nathalie Baracaldo, Horst Samulowitz, and Heiko Ludwig. Flora: Single-shot hyper-parameter optimization for federated learning. *arXiv preprint arXiv:2112.08524* 2021.

Barret Zoph, Deniz Yuret, Jonathan May, and Kevin Knight. Transfer learning for low-resource neural machine translation. *arXiv preprint arXiv:1604.02201* 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

Lingxi Xie and Alan Yuille. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1379–1388, 2017.

Edgar Galván and Peter Mooney. Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Transactions on Artificial Intelligence*, 2(6):476–493, 2021.

Ashraf Darwish, Aboul Ella Hassanien, and Swagatam Das. A survey of swarm and evolutionary computing approaches for deep learning. *Artificial Intelligence Review*, 53(3):1767–1812, 2020.
Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 1761–1770, 2019.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. arXiv preprint arXiv:1812.09926, 2018.

Guohao Li, Guocheng Qian, Itzel C Delgadillo, Matthias Muller, Ali Thabet, and Bernard Ghanem. Sgas: Sequential greedy architecture search. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 1620–1630, 2020.

Hangyu Zhu, Haoyu Zhang, and Yaochu Jin. From federated learning to federated neural architecture search: a survey. Complex & Intelligent Systems, 7(2):639–657, 2021.

Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 11993–12002, 2020.

Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Brian D Ripley. Statistical aspects of neural networks. Networks and chaos—statistical and probabilistic aspects, 50:40–123, 1993.

Ross D. King, Cao Feng, and Alistair Sutherland. Statlog: comparison of classification algorithms on large real-world problems. Applied Artificial Intelligence an International Journal, 9(3):289–333, 1995.

Ron Kohavi and George H John. Automatic parameter selection by minimizing estimated error. In Machine Learning Proceedings 1995, pages 304–312. Elsevier, 1995.
A Algorithms

Algorithm 2 describes the computation-steps performed by each selected client in a communication round during the Search-stage of HANF. The function $L$ corresponds to the loss-function and the functions $\text{wl}(h)$ and $\text{al}(h)$ retrieve the model-weight-learning rate and the architecture-learning rate from a hyperparameter-configuration $h$ respectively.

Algorithm 2 HANF Framework Client-side Search stage
Require: Network parameters $w$ and architecture $a$
Require: Hyperparameter configuration $h$
Require: Data $X_{train}, X_{val}, y_{train}, y_{val}$

\begin{align*}
l_1 &\leftarrow L_{a,h}(w, X_{val}, y_{val}) \\
w^* &\leftarrow w - \text{wl}(h)\nabla_w L_{a,h}(w, X_{train}, y_{train}) \\
a &\leftarrow a - \text{al}(h)\nabla_a L_{a,h}(w^*, X_{val}, y_{val}) \\
w &\leftarrow w - \text{wl}(h)\nabla_w L_{a,h}(w, X_{train}, y_{train}) \\
l_2 &\leftarrow L_{a,h}(w, X_{val}, y_{val}) \\
\end{align*}

return $l_1, l_2, w, a$

Algorithm 3 HANF Framework Client-side Evaluation stage
Require: Network parameters $w$ and architecture $a$
Require: Hyperparameter configuration $h$
Require: Data $X_{train}, X_{val}, y_{train}, y_{val}$

\begin{align*}
l_1 &\leftarrow L_{a,h}(w, X_{val}, y_{val}) \\
w &\leftarrow w - \text{wl}(h)\nabla_w L_{a,h}(w, X_{train}, y_{train}) \\
l_2 &\leftarrow L_{a,h}(w, X_{val}, y_{val}) \\
\end{align*}

return $l_1, l_2, w, a$

During the Evaluation stage HANF just performs regular network-updates using SGD (or SGD-like variants) under a passed hyperparameter-configuration $h$. This is shown in Algorithm 3.

B Training Details

The search spaces of both the Search stage and Evaluation stage are described in Section 4.1. We trained the supernet in the Search stage for $120$ communication rounds on Fashion-MNIST and $240$ communication rounds on CIFAR-10. After each HO-phase we used $10$ communication rounds to perform NAS, the communication rounds per HO were computed based on the entropy of the softmaxed reward-estimates as described in Section 3.2. We chose to set $\beta = 4$ in Eq. 3.2 to allow a maximum of $16$ communication rounds for HO, assuming a uniform distribution over the $120$ hyperparameter-configurations we use as a search space. In each communication round, all selected clients perform one epoch of training. Data was shuffled before distributed on each client using a fixed seed. During training we used gradient clipping with a value of $5$ and chose a batch size of $64$. After the Search stage we selected the architecture leading to the highest accuracy score obtained in the $120/240$ communication rounds for Fashion-MNIST/CIFAR-10. This architecture was then used to build and train an evaluation network from scratch as described in Section 3.2. In the Evaluation stage we performed HO again and allowed the training to take place for $1500$ communication rounds where selected clients perform one epoch of training in each communication round. We again computed the number of HO-rounds as in Eq. 3.2 with $\beta = 4$ and set the number of rounds training under a certain hyperparameter-configuration $h$ to $10$. During training the evaluation network we set gradient clipping to $5$ and chose a batch size of $96$ to be comparable to DARTS.

C Hyperparameter Selection on Fashion-MNIST

Figure 4 shows the hyperparameters selected by HANF over time for a single run on Fashion-MNIST. We observe that HANF chooses more "cautious" hyperparameters than engineers usually do. Additionally, the choice of the momentum seems to not have high influence on the learning behavior since the choices are distributed nearly uniform over possible configurations of the momentum.
Figure 4: This plot shows configurations of the hyperparameters of Fashion-MNIST dataset on the vertical axes and each line corresponds to one choice of hyperparameters at a communication round. The color-scheme of which line corresponds to early or late rounds is depicted on the right.

D Technical Details

We make our code publicly available at: [https://anonymous.4open.science/r/HANF-BD92/](https://anonymous.4open.science/r/HANF-BD92/). We have used FLOWER framework [https://flower.dev/](https://flower.dev/) to implement federated learning setting in HANF. All experiments were run on Nvidia A100-GPUs with 40 GB RAM. One GPU was used for the server and clients were distributed over two GPUs while still being isolated from each other during the experiments.