On the Advancements of the Forward-Forward Algorithm

Mauricio Ortiz Torres, Markus Lange, Arne P. Raulf

German Aerospace Center (DLR), Institut for AI-Safety and Security
Sankt Augustin and Ulm, Germany

Abstract— **The Forward-Forward algorithm has evolved in machine learning research, tackling more complex tasks that mimic real-life applications. In the last years, it has been improved by several techniques to perform better than its original version, handling a challenging dataset like CIFAR10 without losing its flexibility and low memory usage. We have shown in our results that improvements are achieved through a combination of convolutional channel grouping, learning rate schedules, and independent block structures during training that lead to a 20% decrease in test error percentage. Additionally, to approach further implementations on low-capacity hardware projects we have presented a series of lighter models that achieve low test error percentages within (21±6)% and number of trainable parameters between 164,706 and 754,386. This serving also as a basis for our future study on complete verification and validation of these kinds of neural networks.**

I. Introduction

Given the interest in low-power hardware in aerospace projects, the Forward-Forward (FF) Algorithm [1], as introduced by its author Geoffrey Hinton in 2022, offers great potential for the realization of classification tasks of lower memory consumption. The FF algorithm employs two principles: First, the layers conforming to the network are trained independently by measuring their gradients with a locally defined "goodness" loss function, which measures the sum of the squares of the activity vectors in each layer. Second, the ground truth labels are overlaid on the training dataset, to generate a subset of positive and negative datasets. The network is trained to maximize its goodness, favored by positive data and disfavored by negative data. The advantage of working with two forward passes comes during its training phase, because, in difference to the traditional backpropagation, there is no need for storing neural activities or stopping to propagate error derivatives. This approach translates in lower memory requirement during training and in the inference phase. The algorithm has great flexibility for a better hypertuning of its parameters and also for the realization of faster and more efficient inferences. Consequently making it suitable for its use in low-power hardware and possible parallelizations during training [2]. In this paper, we will present a summary of the latest research regarding this algorithm and its performance. Moreover, we will discuss the further improvements that have been made to the general structure of the algorithm as well as some of our studies on lightweight FF models.

II. The Forward-Forward (FF) algorithm

A. Original approach

The architecture of the Forward-Forward algorithm is structured as a traditional neural network. However its training procedure differs significantly and it has two input spaces. These input spaces are given by positive and negative samples which are created from the considered dataset of interest. Their creation for example for MNIST data is as follows: Let $Y = \{y_1, \ldots, y_{10}\}$ be the set of labels, i.e., the different numbers in the MNIST dataset. Take an arbitrary image and its respective label vector y_i. In order to construct positive data samples we one-hot encode y_i and replace the first pixels of the image with the generated one-hot vector. To obtain a negative data sample from the chosen image we randomly pick a label that is not y_i, e.g. $y_j \in Y$, $i \neq j$ and one-hot encode it. We then replace the first pixels of the image with the generated one-hot vector of the wrong label y_j. Thus positive samples are overlaid with the correct label and negative samples with an incorrect one. In contrast to the usual backpropagation training, the FF algorithm operates with two forward passes of the network. One *positive* pass to adjust the weights of the hidden layer, a^l [1], such that the goodness of each layer, $g^l(x) = \sum_j (a^l(x))_j^2$, increases for positive samples above a defined threshold value θ and one *negative* pass to decrease the goodness of the negative samples below the value of θ. During a pass through the network the output of each hidden layer is normalized to only transmit the relative information and not the explicit "goodness"-value from layer to layer.

The overall loss is

$$C_{\mathrm{FF}} = C^l_{pos} + C^l_{neg} =$$

$$\ln[1 + \exp(\theta - g^l(x_{pos}))] + \ln[1 + \exp(g^l(x_{neg}) - \theta)] \quad (1)$$

After the entire model has been trained under the "goodness" loss (1), the inference phase can be initiated by two different methods, referred to as one-pass inference and multi-pass inference. The "one-pass" inference works by training a unique SoftMax/Sigmoid layer with a general multi-classification loss function, such as the cross-entropy loss, and with an input constructed by the activity vectors of each of the layers conforming the original network (Fig. 1).

[1]Note that the output of a hidden layer is sometimes referred to as the activity vector of that layer.

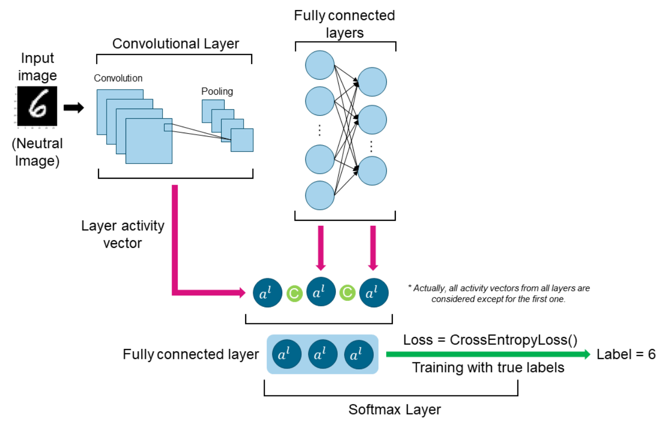

Fig. 1. Flow diagram for the one-pass inference step.

Hence the activity vectors from the second[2] to the last layer of the network are concatenated into a new vector that is later on used as input for the SoftMax layer. The “multi-pass” inference works with the structure of the trained network but with multiple generated overlaid images as input. We employ the same procedure from the generation of positive and negative data to create N possible overlaid images for each one of the N available labels. The overlaid images are then used one at a time in the forward pass, and the associated goodness values for each layer are collected and added up. By sampling all the possible overlaid images we can collect the goodness for each layer and with the use of an argmax function determine the final label of the image (Fig. 1).

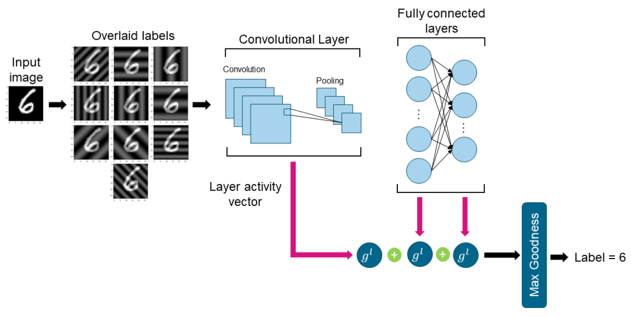

Fig. 2. Flow diagram for the multi-pass inference step.

B. Variations to the algorithm

Along with the several improvements to the model topology while using the original FF algorithm there are several other approaches that try to deal with some of its weaknesses. We classify them according to the components that built the algorithm, starting from the creation of better correlated input data, improved training loss functions, more effective training routines, and faster inference schemes.

a) Creation of input data: According to Hinton the creation of negative input data must be generated such that they have different long-range correlations but very similar short-range correlations[3]. When working with convolutional neural networks (CNN) this is more challenging, since the label information must be present over the entire image, ensuring that the multiple filters applied in each layer can capture features of the original image as well as the information on the classification space. One way of creating the data is given by Scodellaro [3], where a spatially extended labeling technique is used. The technique introduces a superposition of the training dataset image with a second image of 2D Fourier modes of the same size as the training image. The mapping between labels and the specific wavelength, amplitude, and orientation of the Fourier modes is chosen freely. Thus generating as many different waves as labels used in the classification problem. In other studies the authors employ simple learnable embeddings for the same purpose [2], [4]. However, further details on the correlations from the generated images with such techniques are still lacking from the original papers. Another approach is presented by Xing Chen [5]. The creation of positive and negative samples is based on the principles of Noise Contrastive Estimation, where a pairing of positive+positive and positive+negative samples are used as inputs to the network.

One final approach consist on the substitution of a positive and negative samples with convolutional group channels. Here the output of each convolutional layer used in the network $Y \in \mathbb{R}^{N\times C\times H\times W}$[4] is subdivided into group input channels $\hat{Y}_j \in \mathbb{R}^{N\times S\times H\times W}$, where $S = C/J$ and J corresponds to the numbers of classes present in the classification problem. In this construction an holistic goodness factor for each layer is defined as

$$G_{n,j} = \frac{1}{S\times H\times W}\sum_{s=1}^{S}\sum_{h=1}^{H}\sum_{w=1}^{W}\hat{Y}^2_{n,j,s,h,w} \qquad (2)$$

and subsequently an associated positive and negative goodness can be calculated for each convolutional layer, as $\mathbf{g}_{pos} = \mathbf{G}\cdot\mathbf{Z}^T$ and $\mathbf{g}_{neg} = \mathbf{G}\cdot(\mathbf{1}-\mathbf{Z}^T)$, having thus $\mathbf{Z} \in \{0,1\}^{N\times J}$ as the one-hot encoded vector of the true labels [6].

b) Loss functions: Variations of the loss function (1) are considered under the scope of distance metric learning, where the construction of task specific distance spaces gives better control on the distances from the data samples and their respective classes. New loss functions directly measure the discrepancy between projections of input patterns and labels. One such measure is given by Thomas Doom [4] where the Symba loss function has the disappearance of a threshold variable, and the direct difference in between positive and negative goodness values reinforces the discrepancies between highly correlated positive and negative samples. Furthermore, Yujie Wu proposes a loss function that introduces a mixture between absolute and relative distances from the goodness measures [2]. In the case of convolutional

[2]It was shown that using the first hidden layer as part of input in the linear classifier leads to worse performance in the predictions [1].

[3]In the case of classical multi-layer perceptons (MLP) this can be easily achieve with the procedure mentioned in Section II-A.

[4]We work with the NCHW format, where N : Number of data samples in the batch, C : Image channels, H : Image height, W : Image width.

channel-wise competitive learning the team from Andreas Papachristodoulou introduces a channel-wise loss function, defined as,

$$C^l = -\frac{1}{N}\sum_{n=1}^{N}\log\left(\frac{\exp(g_n^{pos})}{\sum_{l=1}^{L}\exp(G_{n,j})}\right), \qquad (3)$$

that supports competitive learning though the channel dimension, where the probability of the target class over the total goodness score for each sample is calculated [6].

c) Training routines: Based on research for parallel training in neural networks there have been explorations on architectures that benefit from allowing error signals to propagate through a neural network during training [7], [8]. By studying different training architectures on layer-wise trained models it has been shown that these architectures can lead to an asynchronous layer-wise parallelism with low memory footprint. The training architectures can be split into four types: 1) the traditional backpropagation, 2) the greedy local update[5], 3) the overlapping local update, and 4) the chunked local update. In these architectures, each group of layers is trained independently and learned through an auxiliary loss function which updates the weights within the group. On the overlapping local update, the network is trained under a series of overlapped n-layers from the original network and conformed by n-subsequent layers, in which every new group takes the last layer from the previous group as its initial layer. The chunked local update proceeds in building several chunks of n-layers sequentially ordered through the entire network without overlapping with other layers. To further enhance biological plausibility there are suggestions on integrating random direct feedback connections on the blocks to replace the back-propagation step and directly use the calculated loss functions to update all relevant weights [2].

d) Faster inference: One can infer that employing all activity vectors of a trained neural network could be inefficient, thus improved algorithms tackle this problem by directly employing the predictions estimated from each layer or a group of layers building the network. In the original paper, the model uses the activity vectors from the second to last layer, however, in more complex neural networks it has been found that initial layers either jeopardize or are superfluous to the final prediction. In newer models, the best results are found by considering the final layer or the last three layers of the model [4]. Other studies explore methods for achieving lightweight inference using a confidence variable in each layer to stop the inclusion of further activity vectors within the model, and consequently achieving faster inference times [9].

Even though the original purpose of the algorithm focuses on its use in classification problems, the range of applicability of the algorithm or its underlying principles is investigated for unsupervised learning as well. In one study the proposed models, referred to as Unsupervised learning Forward-Forward models (UFF), allow training with usual loss functions and without special inputs [10]. The models are built up with cells instead of layers and are based on different unsupervised deep learning models, ranging from an auto-encoder FF cell to a generative adversarial network FF cell, leading to a better performance and stability. The principles of the FF algorithm have been used to develop a novel variance-capturing FF auto-encoder that allows to efficiently update data-driven models in real-time [11]. The study finally, highlights the model's capacity to learn fault detection and isolation in process industries based on FF learning principles.

[5]Which in this case corresponds to the FF original training procedure.

III. Results & Analysis

We have worked with an improved algorithm that has the following characteristics:

1) Creation of convolutional group channels for positive/negative sampling (Figure 3).
2) Use of the channel-wise loss function (3).
3) Training through chunked local updates that alternate in between iterations.
4) Realization of inference based on the activity vectors from the last two layers of the model.

Given the original and our improved algorithm the different features explained in Section II-A and II-B are explored with a series of tests that give a particular focus on the plausible constructions of the network for the MNIST and CIFAR10 datasets. While working with the original algorithm we employed for the MNIST dataset a topology of $784 \times 500 \times 100$ for the MLP; and two convolutional layers of 32 and 64 filters plus a linear layer of 256×100 for the CNN. For CIFAR10 we considered a topology of $3072 \times 3072 \times 2000 \times 1000$ for the MLP, and four convolutional layers of 128, 264, 512, 1024 filters for the CNN. Our results in Table I show that the algorithm works correctly as a classifier and has similar results when employing any of the two kinds of inference options. One drawback of the one-pass inference is the additional softmax training time required to obtain the same order of error obtained from the multi-pass inference. Nevertheless, our tests have shown that the traditional one-pass inference results in less error percentage and faster inference times.

By adopting the improvements discussed in Section II-B we repeated the same tests and obtained the results shown in Table II. In the improvements we have employed a more elaborate network topology [4] which was trained using 300 epochs, where each layer consists of a batch normalization layer, followed by the convolution layer, a ReLU activation function, and an optional maxpool layer. Hence using six convolutional layers of 128, 264 to 512 filters, with three maxpool applications. This model will be denominated by the name *FF_deep*.

The results obtained show a considerable reduction of test error percentage by more than 20% with respect to the original algorithm. However, the use of a multi-step learning rate schedule was necessary for better stability and convergence of the results. In our models, we have applied them with a

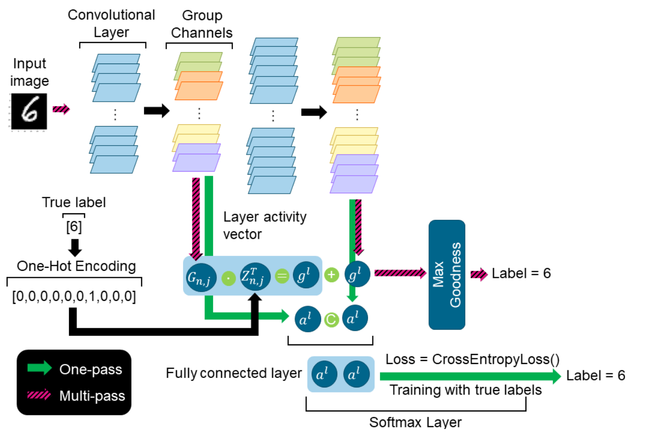

Fig. 3. Flow diagram for the inference step in our improved algorithm. The black arrows emphasises the creation of group channels and the use of the one-hot encoded matrix $\mathbf{Z}$ for the calculation of holistic goodness $\mathbf{G}$.

Dataset	Layers	Inference	Training Error [%]	Test Error [%]	Inference Time [s]
MNIST	MLP	One-pass	8.6	8.0	0.6
	MLP	Multi-pass	7.3	7.2	2.2
	CNN	One-pass	4.5	4.4	0.1
	CNN	Multi-pass	4.9	5.0	1.9
CIFAR10	MLP	One-pass	39.2	39.3	0.1
	MLP	Multi-pass	41.1	41.2	0.2
	CNN	One-pass	20.5	44.3	1.4
	CNN	Multi-pass	44.8	48.3	4.5

TABLE I

TEST RESULTS OF FF ALGORITHM FOR MNIST AND CIFAR10 DATASETS.

$\gamma = 0.1$ at the epochs 150 and 200.[6] Additionally, the usage of convolutional channel-wise competitive learning enables faster inference times for the multi-pass scheme, however, it is still subject to greater times depending on the number of used activation vectors. This characteristic is nevertheless avoided by the one-pass inference step because it's only dependent on the final trained model that initially considered the number of activity vectors before training. Given that the improved algorithm still conserves an independent block structure during training, the benefits from efficient hyperparameter-tuning of the models are still present in this case.

The algorithm presents some specific characteristics that make it very suitable for low-capacity hardware projects. Firstly, the avoidance of backpropagation allows for fewer gradients to be computed during training, and all activation vectors do not need to be kept in memory for the backward pass. This has the advantage that the model can learn while pipelining sequential data through the network without resorting to saving the activity vectors of intermediate layers or propagating error derivatives. The FF algorithm simplifies the computational process by making it less intensive and more easily structured with the two forward passes. The independent training of each layer or block presents better

[6]For the MultiStepLR class defined in pytorch, γ is associated with the multiplicative factor of learning rate decay.

Dataset	Layers	Inference	Training Error [%]	Test Error [%]	Inference Time [s]
MNIST	CNN	One-pass	0.2	0.8	1.7
	CNN	Multi-pass	0.2	0.7	1.4
CIFAR10	CNN	One-pass	1.1	18.8	2.1
	CNN	Multi-pass	1.9	18.2	1.8

TABLE II

TEST RESULTS OF IMPROVED FF ALGORITHM FOR MNIST AND CIFAR10 DATASETS.

regulation on the hyperparameters of the model and thus results generally in better convergence of its gradients. This also prevents further propagation of vanishing or exploding gradients to subsequent layers. The algorithm also has the potential to be parallelizable due to the training independence of each layers, leading to possible accelerations on parallel computing architectures.

To tackle further implementations on low-capacity hardware projects we have explored the possibilities of lighter FF models based only on CNN layers with the mentioned configuration of techniques. The goal was to achieve a noticeable reduction on parameters, because the *FF_deep* model is based on an architecture in [4] where their shallow network has trained 4,100,000 parameters for the last two activation vectors in the inference phase. In Table III we show a list of the studied light FF models along with the *FF_deep* model and their respective architectures. Similarly, in Table IV the results on all the light FF models are provided. The models present a test error of $(21{\pm}6)\%$ and consider a different range of trainable parameters. Our smallest network *FF_tiny* has a reduction in parameters of 96% in respect to *FF_deep*, but is subject to the highest test error of 24.1%. In contrast, *FF_optimal* conserves a test error below 20% and a parameter reduction of 81% in comparison to our largest model.

Model name	Channels; Kernel Size	Trainable Parameters
FF_tiny	[3,50,50,50,50]; [3,3,3,4]	164,706
FF_small	[3,50,50,70,70]; [3,3,3,4]	239,266
FF_medium	[3,50,50,100,150]; [3,3,3,4]	484,506
FF_optimal	[3,50,50,100,160,160]; [3,3,3,4,3]	754,386
FF_deep	[3,130,130,260,260,260,510]; [3,3,3,5,3,3]	4,131,346

TABLE III

ARCHITECTURES FROM THE LIGHT FF MODELS TRAINED WITH THE IMPROVED FF ALGORITHM.

During training the models are trained using 50-70 epochs. Such small value shows that the training methods employed lead to quick convergences of the gradients. Nevertheless, a careful use of the MultiStep learning schedule with 3-5 milestones in between the epochs and $\gamma \in \{0.1, 0.2\}$, pushes the models to the optimal 18% test error. The process of training was susceptible to unstable gradients that propagate though the whole network given our training technique because of our choice to obtain shallow networks. This set of light FF models offers broad flexibility to realize further analysis on low-power hardware, without giving up accuracy on the models. In particular, we are employing this flexibility in an upcoming work on verification and validation of FF models.

Model name	Inference	Training Error [%]	Test Error [%]
FF_tiny	One-pass	15.3	24.4
	Multi-pass	17.9	24.1
FF_small	One-pass	12.3	23.4
	Multi-pass	14.8	23.1
FF_medium	One-pass	8.0	21.5
	Multi-pass	10.1	20.7
FF_optimal	One-pass	2.0	19.6
	Multi-pass	2.4	19.3
FF_deep	One-pass	1.2	18.8
	Multi-pass	1.8	18.2

TABLE IV

TEST RESULTS OF LIGHT FF MODELS WITH THE IMPROVED FF ALGORITHM USING THE CIFAR10 DATASET.

IV. CONCLUSION

The research on the Forward-Forward algorithm keeps evolving into further areas of machine learning and pursues more demanding tasks that are close to real-life applications. The algorithm continues to show its great potential for usage in low-power hardware given its capabilities of layer- or block-wise training, and its simplified computational process. In our results, we have seen that the improved algorithm performs better than the original one. Moreover, it can handle more challenging datasets without losing its flexibility. The improvements in the generation of input data and training routines open the possibility for different learning configurations that allow a faster and more efficient flow of information between the layers. However, the group architectures must be limited since otherwise they could fall back into traditional backpropagation and lose the benefits from the FF architecture. The presented lighter FF models show how this configuration of techniques leads to a broad range of smaller models that still achieve low test error percentages within $(21\pm6)\%$ and a number of number of trainable parameters between 164,706 and 754,386. Hence, these models have a high performance comparable to those analyzed in state-of-the-art FF papers while keeping a considerably smaller size. As a next step, we will investigate the benefits provided by these kinds of models to achieve complete verification and validation of the neural network.

REFERENCES

[1] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations, 2022.

[2] Yujie Wu, Siyuan Xu, Jibin Wu, Lei Deng, Mingkun Xu, Qinghao Wen, and Guoqi Li. Distance-forward learning: Enhancing the forward-forward algorithm towards high-performance on-chip learning, 2024.

[3] Riccardo Scodellaro, Ajinkya Kulkarni, Frauke Alves, and Matthias Schröter. Training convolutional neural networks with the forward-forward algorithm, 2024.

[4] Thomas Dooms, Ing Jyh Tsang, and Jose Oramas. The trifecta: Three simple techniques for training deeper forward-forward networks, 2023.

[5] Xing Chen, Dongshu Liu, Jeremie Laydevant, and Julie Grollier. Self-contrastive forward-forward algorithm, 2024.

[6] Andreas Papachristodoulou, Christos Kyrkou, Stelios Timotheou, and Theocharis Theocharides. Convolutional channel-wise competitive learning for the forward-forward algorithm. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, pages 14536–14544, 2024.

[7] Michael Laskin, Luke Metz, Seth Nabarro, Mark Saroufim, Badreddine Noune, Carlo Luschi, Jascha Sohl-Dickstein, and Pieter Abbeel. Parallel training of deep networks with local updates, 2021.

[8] Yuwen Xiong, Mengye Ren, and Raquel Urtasun. Loco: Local contrastive representation learning. Advances in neural information processing systems, 33:11142–11153, 2020.

[9] Amin Aminifar, Baichuan Huang, Azra Abtahi, and Amir Aminifar. Lightff: Lightweight inference for forward-forward algorithm, 2024.

[10] Taewook Hwang, Hyein Seo, and Sangkeun Jung. Employing layer-wised unsupervised learning to lessen data and loss requirements in forward-forward algorithms. arXiv preprint arXiv:2404.14664, 2024.

[11] Deepak Kumar, Umang Goswami, Hariprasad Kodamana, Manojkumar Ramteke, and Prakash Kumar Tamboli. Variance-capturing forward-forward autoencoder (vffae): A forward learning neural network for fault detection and isolation of process data. Process Safety and Environmental Protection, 178:176–194, 2023.