FFCL: Forward-Forward Net with Cortical Loops, Training and Inference on Edge Without Backpropogation

Ali Karkehabadi, Houman Homayoun, Avesta Sasan
University of California, Davis, CA, USA.
{akarkehabadi,hhomayoun,asasan}@ucdavis.edu

ABSTRACT

The Forward-Forward Learning (FFL) algorithm is a recently proposed solution for training neural networks without needing memory-intensive backpropagation. During training, labels accompany input data, classifying them as positive or negative inputs. Each layer learns its response to these inputs independently. In this study, we enhance the FFL with the following contributions: 1) We optimize label processing by segregating label and feature forwarding between layers, enhancing learning performance. 2) By revising label integration, we enhance the inference process, reduce computational complexity, and improve performance. 3) We introduce feedback loops akin to cortical loops in the brain, where information cycles through and returns to earlier neurons, enabling layers to combine complex features from previous layers with lower-level features, enhancing learning efficiency.

CCS CONCEPTS

• **Computing methodologies** → **Learning to rank**.

KEYWORDS

Logic Locking, De-obfuscation, Formal Verification

ACM Reference Format:
Ali Karkehabadi, Houman Homayoun, and Avesta Sasan. 2024. Forward-Forward Net with Cortical Loops: A Backpropagation-Free Learning Solution for Training and Inference on Edge. In *GLSVLSI '24: Great Lakes Symposium on VLSI, June 12–15, 2024, Tampa, Florida, USA. ACM, New York, NY, USA, 6 pages.*

1 INTRODUCTION

Deep learning has revolutionized problem-solving methodologies, with backpropagation serving as the cornerstone technology enabling the learning process. The backpropagation algorithm aims to fine-tune network parameters to minimize the discrepancy between the network's predictions and the actual ground truth[10]. This process leverages gradient descent, utilizing the chain rule of differentiation to compute the loss function's gradient, allowing for the backward propagation of error. This mechanism enables updates to network parameters in a direction opposite to the gradient, optimizing network efficacy.

GLSVLSI '24, June 12–14, 2024, Clearwater, FL, USA

ACM ISBN 979-8-4007-0605-9/24/06
https://doi.org/10.1145/3649476.3660391

Over the past decade, backpropagation has undergone significant evolution, incorporating new features to address challenges encountered when training deep learning models, such as vanishing or exploding gradients. While these advancements have improved performance, backpropagation remains resource-intensive, demanding massive memory and computational power. Unlike the inference phase, where intermediate activations are consumed soon after they are generated, backpropagation necessitates the retention of all intermediate activations in memory for weight adjustment. This memory-intensive nature presents a considerable challenge, especially when deploying backpropagation on edge devices with limited resources. In such settings, the constraints on memory and compute capabilities exacerbate the difficulty of implementing backpropagation effectively, hindering its practical application in resource-constrained environments. On the other hand, Deep Learning (DL) systems, though inspired by the intricate structure and functionality of the human brain, diverge considerably in their training mechanisms. Notably, there is a lack of concrete evidence to suggest that the brain employs a learning methodology akin to the backpropagation algorithm used in training DL models [9]; The BP algorithm adjusts the weights of connections between artificial neurons based on the gradient of the error function, a process not directly observed in biological neural networks. The brain's ability to efficiently learn and generalize from a limited set of examples far surpasses current DL models [12]. This efficiency hints at a learning paradigm in the brain that is fundamentally different from the iterative error reduction in machine learning. Neuroscientific research indicates the existence of feedback mechanisms within the brain, characterized by complex neural loops rather than the straightforward, error-correcting backpropagation paths found in artificial neural networks [4]. These loops suggest a more dynamic and possibly more efficient information processing system, where forward and backward signals may contribute to learning in a manner not fully replicated by current DL training methodologies.

Recognizing the substantial memory demands of conventional training methods and the limitations of implementing such methods on edge devices, alongside the brain's capability to learn without relying on backpropagation, Geoffrey Hinton introduced the concept of FFL [5]. This innovative approach aims to markedly diminish the memory requirements for training at the edge, bringing them down to levels comparable to those needed for inference with no need for storing the activations. This methodology not only offers a more memory-efficient solution for edge computing but also aligns more closely with our current understanding of neural learning processes in the brain. However, it introduces a complexity in the inference phase, necessitating the separate evaluation of each potential outcome by concurrently inputting the data and respective labels. In our research, we delve into the practicality and efficacy

of FFL, contributing several enhancements that support this emerging field and enhance the FFL solution. Our findings are intended to stimulate broader interest and participation in this promising area of study. Specifically, we have developed refinements to the FFL algorithm that 1) enhance its learning capabilities, and 2) substantially decrease the computational burden during the inference stage. Through sharing our insights and improvements, we aim to encourage wider adoption and further exploration of this novel approach within the research community.

2 BACKGROUND

Predictive Coding (PC) [1] is a theory suggesting the brain predicts sensory input based on past experiences, focusing on discrepancies between predictions and actual input to efficiently process information. This framework explains how perceptions are formed, facilitates learning through error correction, and provides insights into various neurological and psychiatric conditions by highlighting the brain's active role in shaping our sensory experiences and responses to the environment. The FFL algorithm, motivating this work, attempts to emulate the brain's learning processes and in this context adheres to predictive coding. In the PC theory, the brain is considered a predictive system where each layer strives to enhance the accuracy of its own inputs. In this context, as prior art to FFL, we overview the various forms of supervised predictive coding solutions investigated in DL. In a novel approach presented by Dellaferrera et al. [3], the backpropagation was replaced with a dual forward-pass solution. This second pass adjusts the input based on network errors, effectively mitigating the need for symmetric weights, removing the reliance on distant learning signals, and the cessation of neural activity during error backpropagation. Further expanding the landscape of neural network training, Kirsch et al. [6] introduced the Variable Shared Meta Learning (VSML) framework that replaces backpropagation by only forward operations. This algorithm harmonizes various meta-learning methodologies, showcasing that through weight-sharing and benefiting from network sparsity sophisticated learning could be expressed. In another innovative stride, Baydin et al. [2] proposed a technique, denoted as forward gradient, for computing gradients using directional derivatives that are obtained in forward-mode differentiation. The final work, presented in this paper as background, is the solution proposed by Geoffery Hinton, denoted as Forward-Forward learning. Given this paper extends the FFL algorithm, in this section we dive deeper into this concept and cover greater details:

Introduced by Geoffrey Hinton, the Forward-Forward algorithm presents a novel paradigm in neural network training by substituting the conventional backpropagation technique with an additional forward pass. Within this framework, the initial pass is tasked with generating predictions, while the subsequent forward pass is dedicated to refining the model in light of errors detected during the initial prediction phase. This approach endeavors to surmount several of backpropagation's constraints, including its dependency on symmetric weights and the backward error transmission, by confining all modifications to forward operations, thus aligning more closely with mechanisms of biological learning.

In this algorithm, the positive or initial pass processes genuine data, adjusting weights to enhance a defined 'goodness' measure across each layer. In contrast, the negative pass utilizes "negative data" to reduce this 'goodness' measure. Investigating two particular 'goodness' metrics—the sum of squared neural activations and their reciprocal—highlights the potential for adopting diverse metrics in this context.

The learning mechanism, as described in Equation 1, is designed to ensure the 'goodness' measure for authentic data substantially surpasses a predetermined threshold, while remaining markedly below this threshold for negative data. The algorithm aims to accurately classify input vectors as either positive or negative, estimating the likelihood of input being positive by applying the logistic function, σ, to the 'goodness' metric, offset by a threshold, θ. This innovative approach signifies a leap towards enhancing the efficiency and biological fidelity of neural network training, opening new pathways for advancements in machine learning research and its practical deployments.

$$G(\mathrm{X}) = \sigma((\sum_j (x_j^2) - \theta)) \tag{1}$$

This approach underscores a strategic pivot towards enhancing the efficiency and applicability of neural network training, eschewing conventional learning paradigms.

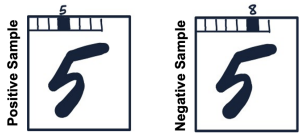

Figure 1: [5] embeds labels in MNIST's black border, altering the first 10 pixels for class representation: '1' for the true class in positive samples and '1' in a random other class for negative samples, with the rest as '0'

Algorithm 1 captures the FFL procedure. The algorithm iteratively adjusts the model's weights over a predefined number of epochs and iterations within each epoch. In each iteration, the algorithm identifies positive samples, where the class is correctly labeled, and negative samples, which are randomly chosen from non-matching classes. For both sets of samples, it computes the activations at each layer of the model. Fig. 1 illustrates the technique for generating negative and positive data in the forward-forward algorithm. The algorithm computes the gradients of the loss function, which is designed to penalize the model for incorrect classifications. Specifically, for positive samples (g_{pos}), the loss increases when the model's confidence in the correct classification is low. For negative samples (g_{neg}), the loss increases when the model incorrectly classifies them as positive. In each epoch of the Forward-Forward Algorithm, both positive and negative data are utilized to calculate the loss function. This loss is computed as the sum of the differences between a pre-defined threshold θ and the 'goodness' measure G, for both positive x_{pos} and negative x_{neg} inputs. The 'goodness' measure $G(x)$ for any input x is obtained by applying the ReLU activation function to the matrix multiplication of x with the transposed weight matrix plus a bias term:

$$G(x) = \mathrm{ReLU}(X \times W^T + b) \tag{2}$$

The loss function can be defined as:

$$\text{Loss} = (\theta - G(x_{\text{pos}})) + (G(x_{\text{neg}}) - \theta) = -G(x_{\text{pos}}) + G(x_{\text{neg}}) \tag{3}$$

This loss aims to optimize the 'goodness' measure by maximizing it for positive inputs and minimizing for negative inputs relative to the threshold θ. So The loss function is defined as:

$$loss = (\log(1 + e^{-g_{pos}}) + \log(1 + e^{g_{neg}}))/2 \quad (4)$$

This loss function combines the penalties for both types of errors in a manner that encourages the model to correctly classify both positive and negative samples with high confidence. Through single-layer BP (and not full BP), the algorithm updates the model's weights according to gradients, progressively reducing the classification error over time. To generate positive and negative samples for the Forward-Forward Algorithm, the algorithm modifies the first N (the number of classes) pixels.

Algorithm 1 Forward-Forward Algorithm

for $l \in$ *model.layers* **do**
  for $e \in$ *MaxEpock* **do**
    # Prepare pos and neg samples
    $x_e, L_{e+} = get\_training\_sample(e)$
    $L_{pos}, L_{neg} = \{C\{0\}\}$; # concat C 0s
    $L_{e-} = random(0, C, L_{e+})$;
    $L_{pos}[L_e] = 1$; # change 0 in label position to 1
    $L_{neg}[L_{ne}] = 1$; # change 0 in a non label position to 1
    $x_{pos} = replace\_boarder(x_e, L_{pos})$
    $x_{neg} = replace\_boarder(x_e, L_{neg})$
    # Run pos and neg samples to target layer
    $g_{\text{pos}} = \text{RunLayers}(0, l, x_{pos})$
    $g_{\text{neg}} = \text{RunLayers}(0, l, x_{neg})$
    # Compute loss
    $\text{loss} = \frac{1}{2}(\log(1 + e^{-g_{\text{pos}}}) + \log(1 + e^{g_{\text{neg}}}))$
    # Update weights
    $W_{grad} = \text{one\_layer\_backpropagate}(loss)$;
    model.layer(l).weights_update(W_{grad});
  end
end

To construct positive samples, the algorithm marks the tensor index for the target class as 1, setting all others to 0, thus denoting the class's presence. Conversely, for negative samples, it chooses a random index, not of the actual class, to mark as 1, leaving the rest at 0, to indicate the class's absence. This method allows clear differentiation between classes by specifying class presence in positive samples and absence in negative ones. For example, for the MNIST dataset with 10 classes, this differentiation is achieved by altering the initial 10 pixels to represent class information.

During training, each layer is trained for a specified number of epochs before proceeding to the next layer. This differs from the traditional multilayer perceptron (MLP) approach, where all layers are typically trained simultaneously. In the Forward-Forward Algorithm, each layer learns independently and is only influenced by the outputs of the preceding layers, enhancing the specificity and efficiency of the learning process.

2.1 Motivation and Problem Statement

In the Forward-Forward Learning (FFL) approach, we encountered several challenges during training and inference. Initially, we found that input labels were directly provided only to the first layer, with subsequent layers receiving a blend of label and feature information. We theorized that this method of indirect labeling, which merges data and features, complicates the learning process for layers beyond the first. As a consequence of this labeling strategy, the inference phase's computational demand is increased by the number of classes, necessitating separate computations for each class by inputting its label into the first layer and processing through subsequent layers. For instance, in a 10-class scenario, this would require executing the network 10 separate times. While FFL mitigates memory constraints during training, it introduces heightened computational complexity during inference. Therefore, we explored alternative methods that could significantly reduce computational demands at inference. Moreover, although FFL draws inspiration from the hypothesis of the brain's Contrastive Excitatory (CE) learning process, it maintains a strictly forward flow of information within a Directed Acyclic Graph (DAG) structure. We hypothesize that incorporating feedback mechanisms, while retaining FFL's training efficiencies, could enhance the model's learning capabilities, albeit at the cost of increased training complexity since each node might undergo training during both the forward pass and potentially multiple rounds of feedback. Nonetheless, we anticipate that training incorporating feedback mechanisms, following the initial feedforward network training, could be executed more swiftly. Consequently, we propose a methodology designed to augment FFL's learning precision while simultaneously curbing computational burdens during inference.

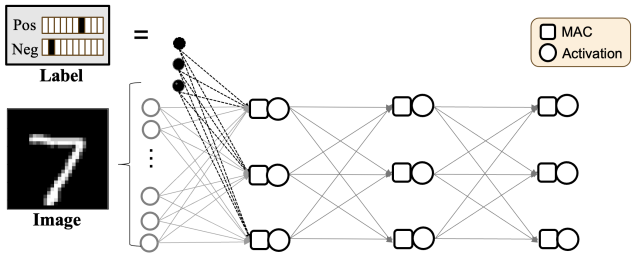

Figure 2: Forward-Forward Algorithm Setup: Transforms MNIST images to a 784-unit input layer. For class representation, the initial N pixels are adjusted: '1' for the correct class in positive samples and '1' in a non-class index for negative samples, particularly in the first 10 pixels for MNIST. Layers are trained sequentially.

3 METHODOLOGY

We introduce a Feed Forward with Cortical Loop (FFCL) approach that builds on FFL, offering improvements through direct labeling of each hidden layer and incorporating feedback loops into the model architecture, along with a revised training methodology. The first enhancement boosts training accuracy, shortens training duration, and reduces inference complexity. The second adjustment further reduces the model's complexity with a marginal increase in model complexity. We detail our two-phase solution as follows:

3.1 Direct Label Feeding

Our methodology enhances supervised FFL training by directly inputting label information into each layer, diverging from FFL that blends and propagates label and feature data from the initial layer onwards. In the context of training, this approach involves appending the label to the input as additional data, effectively as padding, ensuring the integrity of the primary input remains intact.

This unique strategy allows for distinct computations corresponding to positive and negative labels, which are then integrated with the computations arising from the interaction of input features and layer weights, plus the bias, before being passed through the activation function. The clear separation of label influence in the computational process enhances the specificity of information each layer receives, making the learning process more efficient.

Once the first hidden layer is trained, the subsequent layers employ only the outputs derived from the combination of input features and layer weights from the previous layer, along with the bias, excluding the direct label influence from further computations. This selective utilization of output streamlines the training process for subsequent layers, as it circumvents the need for managing distinct datasets for positive and negative labels, thereby simplifying the computational framework.

This streamlined approach extends through all subsequent layers, maintaining the clarity and specificity of label information without the computational overhead of separating positive and negative label datasets. Such an architectural innovation substantially mitigates the challenges associated with diluting label clarity across layers and the inefficiencies tied to training with distinct datasets for different label outcomes. Moreover, this methodology significantly diminishes the computational burden during the inference phase. By obviating the need for extensive class-based recalculations, the model can swiftly execute inference tasks by applying the relevant class labels to the learned features from the preceding layer, thereby enhancing the model's performance and efficiency. Our approach, illustrated in Fig. 3, captures this advancement, demonstrating how each layer is directly labeled, and how features are directly propagated.

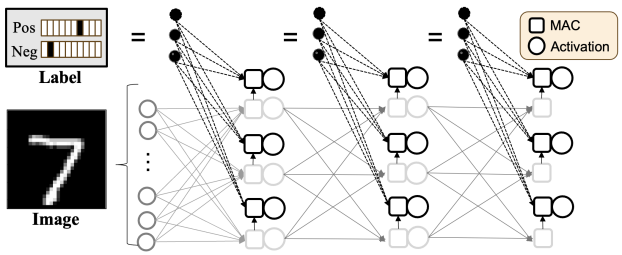

Figure 3: In our revised FFL we directly integrate class labels into the structure, maintaining original image integrity. Training links input and labels to initial layer neurons, enabling specialized computations. The following layers focus on weighted images and bias from the prior layer, optimizing processing.

In our solution, instead of replacing the boundary pixels of the image with the label pixels and using the label only in the computation of the first layer, we feed the label separately to each layer. The operation of each neuron, in Fig. 3 is divided into MAC operation followed by activation. As shown, in Fig. 3, the neurons we use for computing the estimation of goodness (in dark black) and neurons used for propagating features to the next layer are separated. The labels only impact the estimation neurons. The estimation neurons are fed by the propagation neurons as a bias (1 input), and N weights for N labels. or more specifically, as shown in eq. 5, for each layer first we generate the multiplication and accumulation sum s for each neuron using model weights connected to the input or previous layer activation w_{in} and bias, then using eq. 6 we compute the goodness, by using weights connected to labels w_{label} and using s as bias, and finally we prepare the activation of the neuron as input for the next layer in eq. 7. The complete detail of our algorithm is given in Algorithm 2.

$$s = x \times w_{in}^T + b \tag{5}$$

$$g_{pos/neg} = Relu(Label \times w_{label}^T + s) \tag{6}$$

$$a = Relu(s) \tag{7}$$

Algorithm 2 Forward-Forward with Direct Label Feed (FFDL)

for $l \in$ *model.layers* **do**
  for $e \in$ *MaxEpock* **do**
    # Get new training sample
    $x_e, L_{e+} = get\_training\_sample(e)$
    # use the input to run to target layer
    x_e = RunLayer$(0, l, x_e)$; #Activation output stored in x_e
    # Prepare pos and neg samples
    $L_{pos}, L_{neg} = \{C\{0\}\}$; # concat C 0s
    $L_{e-} = random(0, C, L_{e+})$;
    $L_{pos}[L_e] = 1$; # change 0 in label position to 1
    $L_{neg}[L_{ne}] = 1$; # change 0 in a non label position to 1
    $x_{pos} = \{x_e, L_{pos}\}$ # concat with label
    $x_{neg} = \{x_e, L_{neg}\}$ # concat with label
    # Run pos and neg samples to target layer
    g_{pos} = RunLayers$(0, l, x_{pos})$
    g_{neg} = RunLayers$(0, l, x_{neg})$
    # Compute loss
    loss = $\frac{1}{2}(\log(1 + e^{-g_{\text{pos}}}) + \log(1 + e^{g_{\text{neg}}}))$
    # Update weights
    W_{grad} = one_layer_backpropagate$(loss)$;
    model.layer(l).weights_update(W_{grad});
  end
end

As mentioned earlier, this direct label feeding approach not only improves accuracy but also reduces the inference computational cost. We will review the impact of direct label feed in the result section of this work, however, next we elaborate on how this approach reduces computational complexity. The original FFL is an MLP. Let's assume an MLP with M input neurons and N output neurons. The number of required FLOPS to compute each layer in the FFL is computed as $MN + 2N$, where the added $2N$ accounts for the cost of activation and also the addition of the bias term. However, note that in the original FFL, to perform the classification, each of the labels has to be fed to the first layer, and since the addition of the labels changes the activations in each layer, all layers need to be recomputed. Hence, for C classes, each layer needs to be computed C times, resulting in:

$$\text{FFL FLOPs/Layer} = MNC + 2NC \tag{8}$$

In our solution, however, the computation of each layer is done with label-independent activation from the previous layer and the application of new labels. For computing the label-independent neurons, we have $NM + N$ operations. For computing the goodness for one label we have $CN + N$ operation, where C is the number of labels, and N accounts for the addition of sum as bias. Finally, for the computation of all labels (C of them), we have $C^2N + CN$

operations. In the result, the total number of operations for each layer to measure the goodness across all classes is:

$$\text{FLOPs/Layer} = MN + NC^2 + NC + 2N \tag{9}$$

Removing N from both equations, the advantage comes when the following inequality holds:

$$M + C^2 + 2 < MC + C \tag{10}$$

which could, with some estimation simplified as advantagous when $M > C + 1$. With MNIST, C is 10, and on average M is 500 in the model used in FFL.

3.2 Forward-Forward Net with Cortical Loops

In our second key contribution, we incorporate feedback mechanisms analogous to those found in the brain, enabling each layer to provide feedback to the preceding one. Given the complexities involved in training models with feedback, we employ a technique that involves unrolling the network multiple times and sharing weights across these unrolled instances. The extent of unrolling determines how far back the information can propagate through the network. For instance, duplicating the network twice, as depicted in Fig. 4, allows information from each layer to influence up to two preceding layers. In this setup, a weight-sharing strategy is adopted to ensure consistency across all duplicates of the network, meaning that updating the feedforward weights in the first layer concurrently updates those in the subsequent duplicates.

This unrolling approach transforms the challenge of training a network with feedback into training an expanded feedforward network, which is readily manageable with existing learning frameworks. In our model, the same input is fed into each unrolled instance, simulating the effect of feedback on a static image. This technique could also accommodate variations of the input image, such as augmented or rotated versions, potentially enhancing model robustness. Furthermore, this architecture could facilitate the processing of time-sequenced images from the same scene, offering rich temporal information that might enhance learning. While these applications remain unexplored in the current work, they underscore the potential of our feedback-based architecture, which we aim to investigate in future studies.

The proposed unrolling approach also introduces considerations regarding training schedules. Training can occur for the feedforward path as soon as the previous layer is ready, and for the feedback path when the subsequent layer is trained. Various training schedules are conceivable, ranging from multiple forward passes capturing information from subsequent layers as it becomes available, to Just-In-Time training that initiates feedback loop training as soon as it's feasible. Although exploring the impact of different training schedules is beyond the scope of this paper and reserved for future research, we present a single training methodology without asserting its superiority. It's important to note that the model's complexity is only marginally increased by the added feedback mechanism, thanks to weight sharing, thus preventing significant growth in model size. For inference, we evaluate the accuracy of the layers in the final unrolled instance, which contains the bulk of the feedback information. Accuracy metrics are reported both individually for each layer and collectively for the entire model, with the latter being the aggregation of positive and negative votes for each class across all layers in the final instance. Unlike a simple average, this ensemble approach combines votes, offering a nuanced measure of accuracy. The training algorithm for our unrolled model is detailed in Algorithm 3.

Algorithm 3 Forward-Forward Net with Cortical Loops (FFCL)

for n & $l \in$ *model.networks.layers* **do**
  for $e \in$ *MaxEpock* **do**
    #In the sequence from 0 up to the $n + l$, *priority* : $n \leq l$.
    # Get new training sample
    $x_e, L_{e+} = get\_training\_sample(e)$
    # use the input to run to target layer
    x_e = RunLayer$(0, n, l, x_e)$; #Activation output stored in x_e
    # Prepare pos and neg samples
    $L_{pos}, L_{neg} = \{C\{0\}\}$; # concat C 0s
    $L_{e-} = random(0, C, L_{e+})$;
    $L_{pos}[L_e] = 1$; # change 0 in label position to 1
    $L_{neg}[L_{ne}] = 1$; # change 0 in a non label position to 1
    $x_{pos} = \{x_e, L_{pos}\}$ # concat with label
    $x_{neg} = \{x_e, L_{neg}\}$ # concat with label
    # Run pos and neg samples to target layer
    g_{pos} = RunLayers$(0, n, l, x_{pos})$
    g_{neg} = RunLayers$(0, n, l, x_{neg})$
    if *network != 0* **then**
      # Run pos and neg samples to target layer
      $g_{\text{pos}} = g_{\text{pos}}$ + RunLayers$(0, n-1, l+1, x_{pos})$
      $g_{\text{neg}} = g_{\text{neg}}$ + RunLayers$(0, n-1, l+1, x_{neg})$
    end
    # Compute loss
    loss = $\frac{1}{2}(\log(1 + e^{-g_{\text{pos}}}) + \log(1 + e^{g_{\text{neg}}}))$
    # Update weights
    W_{grad} = one_layer_backpropagate$(loss)$;
    model.layer(l).weights_update(W_{grad});
  end
end

In Algorithm 3, we introduce Forward-Forward Net with Cortical Loops that employs a Parallel Direct Label Feeding mechanism, defined within an MLP (Multilayer Perceptron) framework. Here, 'network' denotes the number of parallel instances of the Direct Label Feeding model, while 'layer' refers to the number of layers within each network instance.

For networks identified by an index of 1, indicating the first network, there is an absence of backward input connections. Hence, the computation of g_{pos} and g_{neg} is exclusively based on the outputs from the last layer of this network. In contrast, for networks with indices greater than 1, the model incorporates inputs from two sources for the calculation of g_{pos} and g_{neg}: outputs from the preceding layer and backward inputs from an adjacent layer. This approach is critical for the accurate calculation of the loss.

It is crucial to note the sequence of training for the layers in this model. The training process prioritizes the aggregate of the number of networks and the number of layers, under the stipulation that the network index is less than or equal to that of the layer index. This prioritization scheme facilitates a systematic and effective training regimen for the Parallel Direct Label Feeding model.

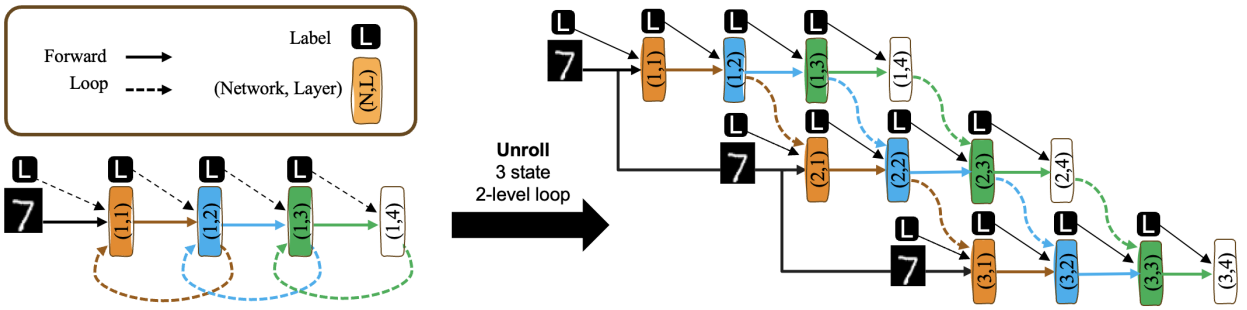

Figure 4: Architecture of Forward-Forward Net with Cortical Loops. To train the model effectively using existing training engines, the model is unrolled. The number of times, the model is unrolled decides the degree to which feedback information propagates in the system. For example, with a 3 (N) layer network, each layer feedback could reach 2 (N-1) previous layers.

In our proposed architecture, feedback is currently limited to the immediately preceding layer and is applied whenever feasible, utilizing distinct weights from those in the feedforward path. This implementation represents just one approach to integrating feedback mechanisms. The potential exists for feedback to extend beyond the adjacent layer, creating additional loops within the network. Moreover, it's conceivable that not all feedback connections are necessary; a more sparse feedback architecture might be sufficient for our objectives, a topic of future study.

4 RESULTS

Our evaluation of the enhanced FFL solution, which incorporates direct label input and cortical loops, was conducted on three benchmark datasets: MNIST [8], Fashion-MNIST [11], and CIFAR-10 [7]. Both MNIST and Fashion-MNIST datasets consist of images with dimensions of 28×28 pixels. These images were converted into a flattened format to form an input layer with 784 units. For the CIFAR-10 dataset, which contains images of size $3 \times 32 \times 32$, we processed the images in a similar fashion, resulting in an input layer of 3072 units. All computational experiments were performed using a NVIDIA T4 GPU, equipped with 2,560 CUDA cores, 320 Tensor cores, and 16 GB of GDDR6 memory. The models were trained and tested using FP16 precision. In each of the three experiments, a 4-layer MLP architecture was employed. The specific model architecture used in these experiments is detailed in Table 1.

Model Size	MNIST	Fashion-MNIST	CIFAR10
input	784	784	3072
1st Layer	500	500	2048
2nd Layer	500	500	1024
3rd Layer	500	500	512
4th Layer	500	500	512

Table 1: The MLP model architecture used for training the FFL, FFDL, and FFCL for each selected dataset.

Each model underwent training 50 times, with each training session spanning 5,000 epochs using the Adam optimizer. We documented the accuracy for each layer individually and then calculated the overall model accuracy. The accuracies reported are averages from the 50 training iterations. Please note that the overall model accuracy is not simply an average of the layer accuracies. In line with the methodology established by FFL [5], model accuracy is determined by aggregating the 'goodness' votes from all layers before making a final decision. Thus, the model accuracy reflects the collective decision derived from the sum of votes, rather than an average of individual model layer decisions. Table 2 shows the accuracy comparisons of FFL, FFDL, and FFCL when trained on the MNIST dataset. The table clearly shows that each layer's accuracy in FFDL improves over FFL, and similarly, each layer's accuracy in FFCL improves over FFL. It is also noteworthy that while there is a significant drop in layer accuracies within FFL, the direct label feeding minimizes this drop in both FFDL and FFCL. Finally, the table highlights that the final model accuracy has significantly improved in both FFDL and FFCL, with FFCL achieving an accuracy of 97.23% compared to FFL's 94.86%.

Accuracy	FFL[5]	FFDL (Label Feed)	FFCL (Cortical Loop)
1st Layer	94.77%	96.10%	96.72%
2nd Layer	94.43%	95.54%	96.64%
3rd Layer	94.39%	94.93%	95.76%
4th Layer	91.83%	94.77%	95.46%
Model	94.86%	96.57%	97.23%

Table 2: Comparative results of the Forward-Forward method and the New Ideas for MNIST dataset

Table 3 presents the accuracy results of models trained on the FashionMNIST dataset. Due to the greater complexity of the FashionMNIST compared to MNIST, the accuracies reported are somewhat lower. Nonetheless, the enhancements from utilizing FFDL and FFCL are more distinct. The model accuracy of FFL was recorded at 86.04%, whereas FFDL achieved a model accuracy of 87.6% and FFCL reached 89.7% accuracy, marking an improvement of nearly 4% over FFL.

Accuracy	FFL[5]	FFDL (Label Feed)	FFCL (Cortical Loop)
1st Layer	85.83%	86.11%	86.19%
2nd Layer	83.93%	84.21%	84.64%
3rd Layer	82.81%	83.93%	84.12%
4th Layer	81.19%	82.49%	83.98%
Model	86.04%	87.61%	89.71%

Table 3: Comparative results of the Forward-Forward method and the New Ideas for FashionMNIST dataset

Table 3 documents the training accuracy of three models on the 4 dataset, which is the most complex dataset in our experiment. A very similar pattern is noted here. In FFL, the model accuracy significantly decreases in deeper layers, whereas in FFDL and FFCL, this drop is much less pronounced. Additionally, FFCL exhibits higher accuracy compared to FFDL, and FFDL shows higher accuracy compared to FFL across all layers and in overall model accuracy.

As demonstrated, both the direct feeding of labels and the introduction of cortical loops effectively enhance model accuracy in forward-forward training.

Accuracy	FFL[5]	FFDL (Label Feed)	FFCL (Cortical Loop)
1st Layer	46.36%	47.29%	47.32%
2nd Layer	46.56%	44.12%	46.12%
3rd Layer	37.34%	43.69%	45.78%
4th Layer	36.23%	42.86%	44.22%
Model	47.78%	48.71%	49.93%

Table 4: Comparative results of the Forward-Forward method and the New Ideas for CIFAR10 dataset

Table 5 details the computational complexity for a 4-layer model trained on the MNIST and Fashion MNIST datasets, as shown in Table 1. Table 6 provides the inference complexity for the CIFAR 10 model. The FLOP counts are derived from equations 8 and 9.

Number of FLOPS	FFL[5]	FFDL	FFCL
1st Layer	420,000	420,000	1,260,000
2/3/4th Layers	2,510,000	306,000	918,000
4 Layers Model	7,950,000	1,338,000	4,014,000

Table 5: Comparative Results of FLOPS for the Forward-Forward Method vs. Label-Enhanced Hidden Layers Model on the MNIST and FashionMNIST Dataset, Using Models with 500 Neurons per Hidden Layer

By directly feeding labels, we can reuse neuron outputs to generate label-specific outputs in each layer without restarting the model for each label. This method significantly reduces the inference complexity in the 2nd, 3rd, and 4th layers, while maintaining the same complexity in the first layer. For instance, in models for MNIST and Fashion MNIST, total model complexity drops from 7.95M FLOPS to 1.34M FLOPS, a 5.9X reduction. While FFCL has a higher computational demand than FFDL due to executing the unrolled model, its complexity is still only half of FFL for MNIST and FashionMNIST, and almost a quarter for CIFAR 10. This shows that FFCL and FFDL not only improve accuracy over FFL but also greatly reduce inference complexity.

Number of FLOPS	FFL[5]	FFDL	FFCL
1st Layer	62,955,520	6,520,832	19,562,496
2nd Layer	20,992,000	2,211,840	6,635,520
3rd Layer	5,253,120	581,632	1,744,896
4th Layer	2,631,680	319,488	958,464
4 Layers Model	91,832,320	9,633,792	28,901,376

Table 6: Comparative Results of FLOPS for the Forward-Forward Method vs. Label-Enhanced Hidden Layers Model on the CIFAR10 Dataset, Using Models with 500 Neurons per Hidden Layer

5 DISCUSSION ON FUTURE WORK

The FFL introduces a fresh approach to on-device learning by eliminating the conventional need for backpropagation. The FFDL and FFCL variants proposed in this paper further improve on FFL by increasing accuracy and reducing computational overhead. However, forward-forward learning is still in its early stages with many unanswered questions and many unexplored possibilities. In this section, we like to highlight some of the possibilities for extending the FFDL and FFCL. The FFCL uses the input to each unrolled copy of network. Introducing variations such as altered or time-series versions of the initial image could improve the model's robustness by enhancing the flow of information to each layer, potentially boosting resilience and performance. Another area for optimization is the FFCL's use of dense, inter-layer cortical feedback loops. Modifying these loops to extend across multiple layers or using them more sparingly could substantially simplify the current setup. Furthermore, changing the backpropagation requirements in FFL, FFCL, and FFDL reduces the need for backpropagation to a single layer, easing the storage burden of intermediate activations—a significant advantage for resource-limited edge devices. However, these models still require some backpropagation that typically depends on floating-point hardware. Researching ways to perform single-layer backpropagation on fixed-point hardware could further lessen memory requirements and eliminate the reliance on expensive floating-point computations.

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under Award #2203399.

6 CONCLUSION

In conclusion, this paper presented two novel variants of Forward-Forward Learning (FFL), each enhancing the framework in distinct ways to improve efficiency and accuracy. The first variant, Forward-Forward Direct Labeling (FFDL), incorporates direct label feeding into each voting layer, which significantly boosts model accuracy and reduces computational costs during inference by eliminating the need to rerun the entire network for different label votes. The second variant, Forward-Forward Cortical Loops (FFCL), builds on the direct label feeding strategy by integrating cortical loops, which allow for bidirectional information flow throughout the learning network, thereby further enhancing model accuracy.

REFERENCES

[1] Laurence Aitchison and Máté Lengyel. 2017. With or without you: predictive coding and Bayesian inference in the brain. *Current opinion in neurobiology* 46 (2017), 219–227.

[2] Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr. 2022. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587* (2022).

[3] Giorgia Dellaferrera and Gabriel Kreiman. 2022. Error-driven input modulation: Solving the credit assignment problem without a backward pass. In *International Conference on Machine Learning*. PMLR, 4937–4955.

[4] David A Forsyth, Joseph L Mundy, Vito di Gesú, Roberto Cipolla, Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. 1999. Object recognition with gradient-based learning. *Shape, contour and grouping in computer vision* (1999), 319–345.

[5] Geoffrey Hinton. 2022. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345* (2022).

[6] Louis Kirsch and Jürgen Schmidhuber. 2021. Meta learning backpropagation and improving it. *Advances in Neural Information Processing Systems* 34 (2021), 14122–14134.

[7] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[8] Yann LeCun, Corinna Cortes, Chris Burges, et al. 2010. MNIST handwritten digit database.

[9] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. 2020. Backpropagation and the brain. *Nature Reviews Neuroscience* 21, 6 (2020), 335–346.

[10] Wan-Duo Kurt Ma, JP Lewis, and W Bastiaan Kleijn. 2020. The HSIC bottleneck: Deep learning without back-propagation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 5085–5092.

[11] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[12] Anthony M Zador. 2019. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications* 10 (2019), 3770.