Forward-Forward: Is it time to bid adieu to backprop?

Sai Anuroop Kesanapalli, Shashank Rangarajan, Avtaran Jain, Anukaran Jain
Department of Computer Science
University of Southern California
Los Angeles, CA, USA
{kesanapa, sr87317, anukaran, avtaranj}@usc.edu

Abstract

The aim of this work is to delve deeper into the Forward-Forward (FF) algorithm (Hinton, 2022) by characterizing and analyzing its performance in comparison to the traditional back-propagation (backprop) approach. The FF algorithm achieves accuracies on par with backprop on benchmark datasets but showcases deteriorating performance as the complexity of the dataset increases. Furthermore, FF is as efficient as backprop in terms of system performance. However, popular backprop architectures such as CNN and Self-Attention do not work well with FF due to their inherent structural biases. Despite this, FF requires lesser number of samples to achieve similar accuracy as that of backprop and has a more stable performance than the latter, showing promise for further investigation into this novel approach.

1 Introduction

Back-propagation (Hinton et al., 1986) has been at the forefront of various machine learning innovations, allowing for the creation of semi-intelligent tools such as ChatGPT, Alexa, DALL-E, and many more. Despite its success, the algorithm comes with its own set of limitations (Hinton, 2022). In general, backprop training is highly memory intensive as it relies on the activations computed during the forward pass that are retained in memory to calcluate the gradients in the backward pass. This is where FF might offer an advantage by not requiring storage of activations and iterations of backward pass during training.

In our work, we delve deeper into understanding the following question - does FF offer any advantages over backprop when working with varying datasets and architectures? In particular, the above question is answered through the following steps:

1. Comparing performance of baseline FF and backprop models on datasets of increasing complexity
2. Integrating FF with other popular architectures such as Convolutional Neural Networks (CNN) (Krizhevsky et al., 2017) and Self Attention (Vaswani et al., 2017)
3. Exploring whether FF acts as a good initializer for backprop and vice-versa
4. Analyzing the system performance of FF and backprop
5. Analyzing sample complexity of FF vs backprop

We have discovered that although basic FF algorithm performs on-par with backprop on simpler datasets, more complex architectures such as CNN and Self-Attention do not perform well with FF due to their weight-sharing nature (Hinton, 2022). Additionally, we have discovered that FF and backprop are fundamentally different, making it less appealing to integrate one with the other. However, our experiments have shown that both approaches have similar system performance based on the metrics we have gathered during model training. Moreover, when using varying dataset subsamples, FF has demonstrated more stable performance than backprop.

Our paper is organized as follows: We first explain the relevant background (Section 2). Next, we glance at the works that have explored FF, as well as the ones that have influenced the algorithm (Section 3). We then describe our methodology (Section 4), and detail our experiments with their nuances, discussing the results (Section 5). Finally, we conclude (Section 6) and outline future works (Section 7).

2 Background

2.1 Back-propagation

Back-propagation (backprop), is a widely-used algorithm in training neural networks. Its breakthrough lies in its ability to enable the optimization of models with many layers, by computing the gra-

dients of each parameter with respect to the loss function. This is done by propagating the errors from the output layer back to the input layer, hence the name *back-propagation*. The key idea is to apply the chain-rule in calculus to compute the gradients at each layer, by multiplying the gradients of the layer ahead with the local gradients of the current layer. These gradients are then used to update the model parameters using gradient descent. Backprop is computationally efficient because it stores the gradients and activations, and only needs to compute the gradients once, which can then be reused during the optimization process.

2.2 Forward-Forward

The core idea behind the FF algorithm is to use two forward passes, one with positive (real) data and the other with negative (generated) data. Instead of computing the gradients with respect to the global loss, every layer has its own objective function called the *goodness* metric. This allows for the training of every layer in a neural network to be done separately. During training, the weights of each layer are adjusted to increase the goodness value above a specified threshold in the positive pass, while the opposite is done in the negative pass. In our experiments throughout, the sum of squared activities is used as the *goodness* function.

2.3 Goodness metric

Suppose that the *goodness* function for a layer is the sum of the squares of the activities of the rectified linear neurons in that layer. *Goodness* should be well above some threshold value θ for real data and well below that value for negative data. The aim is to correctly classify input vectors as positive data or negative data based on the probability derived by the following *logistic* function (Hinton, 2022):

$$p(positive) = \sigma\Big(\sum_j y_j^2 - \theta\Big)$$

$$p(negative) = \sigma\Big(\theta - \sum_j y_j^2\Big)$$

where y_j is the activity of hidden unit j before layer normalization. This threshold θ becomes a hyper-parameter that can be tuned for better performance.

2.4 Method to produce positive and negative samples

In FF, a positive datum belongs to the training dataset whereas a negative datum is adversarially

```python
def overlay_y_on_x(x, y, num_labels):
    """Replace the first num_labels pixels of data [x] with one-hot-encoded label [y]
    """
    x_ = x.clone()
    x_[:, :num_labels] *= 0.0
    x_[range(x.shape[0]), y] = x.max()
    return x_
```

Figure 1: `overlay` method snippet

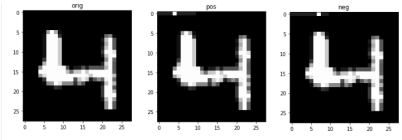


Figure 2: Original, positive, and negative samples

generated to ensure that the model learns contrastively. An easy way to generate the negative data is to mislabel the training samples. We do this using the `overlay` method in Figure 1, and a sample visualization for an MNIST image is shown in Figure 2.

3 Related Works

In 3.1, 3.2, 3.3, 3.4 we present an overview of recent line of works related to FF which help us understand the present status of FF and get an idea about the open challenges that FF poses. In 3.5 we present a few system performance and ML benchmarking works that we draw inspiration from, to aid in our analysis of FF from a practical perspective.

3.1 Forward-Forward

FF is a novel idea and not a lot of work has been done with regards to exploring the potential benefits and drawbacks of using this algorithm. Moreover, there has been a growing interest in the search of alternatives to backprop, especially in scenarios related to low system analog devices. (Hinton, 2022) explores the feasibility of the FF approach when compared to backprop and shows that FF performs nearly as well as backprop when dealing with simple multi layer neural networks. However, the paper does not dive deeper into the pros and cons of this approach, when looking at varying models and architectures. We attempt to reproduce the results demonstrated in the paper, as well as gauge the effectiveness of the proposed approach when dealing with hybrid models and its compatibility with popular backprop-based architectures such as CNN and Self-Attention. In addition to these, we delve deeper into the system performance comparison of FF vs backprop in this work.

3.2 Activation Learning

(Zhou, 2022) presents the activation learning paradigm as an alternate framework to backprop, which though is very similar to the FF algorithm proposed by (Hinton, 2022), differs in the way weights are updated. There is an abundance of comparison between activation learning and backprop present in this work. We referred to these comparisons and formulated our own juxtapositions between FF and backprop algorithms, analyzing the results for further insights.

3.3 Local Activity Contrastive algorithm

(Zhu et al., 2022) propose Local Activity Contrastive (LAC) algorithm to learn auto-encoders. The idea behind the paper is to use loss functions in every layer of the neural network to replicate locality. Particularly, when learning the difference between activations of two inputs, an original image and an reconstructed image each are minimized. The above is done using two forward passes. LAC is also shown to be beneficial as a pre-training method for Convlutional Neural Networks (CNN), and is what inspired us to delve deeper into a hybrid learning model that uses both backprop and FF.

3.4 Extensions to FF

Lastly, other studies such as those of (Ororbia and Mali, 2023) utilize FF algorithm as a tool to propose new learning paradigms as generalizations. However, these papers do not perform a direct and detailed analysis between FF and backprop.

3.5 System Performance Metrics and Benchmarking

Benchmarking of Deep Learning (DL) models along with analyzing the system behaviour during the training and inference phases is a crucial step in profiling these models. In many cases, these models run on edge-accelerators which have limited power budget (Khochare et al., 2022). Further, these devices are constrained by CPU and GPU computational power, hence it is crucial to see how extant and new DL models behave in light of these considerations.

Typically, E2E runtime, CPU and GPU utilization and memory consumption (Liu et al., 2019), power and energy consumption (Holly et al., 2020) are logged by several works. (Beutel et al., 2020) report training times, energy, CPU running time with respect to the federated learning framework they propose.

(SK et al., 2022) do a detailed system profiling of Nvidia Jetson edge accelerators during the training phase of various deep learning models. In particular, they log E2E time, stall time, GPU compute time, CPU and GPU frequencies, energy consumption, average socket power load as part of their study. However, their work is Jetson-specific and the results may not be universally applicable.

MLPerf (Mattson et al., 2020) is a community-driven effort which provides a unified platform for measuring the performance of ML hardware and systems. It is a benchmark suite with one of its major objectives being that it enables a fair comparison between candidate systems. To this end, they assorted a representative set of tasks from several major ML areas, spanning from vision, language, recommendation, to reinforcement learning. Further, for each benchmark, they chose quality metrics close to the state of the art for the corresponding model and data set. While this is a very detailed work, it does not measure low-level system metrics. However, this work sets the tone for creating a similar platform for fair comparison of FF vs backprop based methods.

4 Methodology

(Hinton, 2022) explores the feasibility of the FF approach when compared to backprop and shows that FF performs nearly as well as backprop when dealing with simple feed-forward multi-layer neural networks. In our first experiment, we reproduced the results demonstrated in the paper (Table 1). Additionally, we experimented with various datasets (Section 4.1) and gauged the system metrics (Section 4.2) for both FF and backprop. In our second experiment we explored the efficacy of a hybrid FF-backprop framework. Our intuition here was to verify if FF could act as a good weight initializer for backprop and vice-versa. Finally, in our third experiment, we attempted to implement more complex architectures like CNN and Self-Attention to see if, similar to backprop, FF also benefited from these architectural inductive biases. The model architectures for these experiments are discussed in Section 4.3.

4.1 Datasets

For our experiments, the following 5 vision datasets were used – MNIST (LeCun et al., 2010), Fash-

ionMNIST (Xiao et al., 2017), CIFAR10, CIFAR100 (Krizhevsky, 2009) and SVHN (Netzer et al., 2011).

4.2 Metrics Logged

The following types of system performance metrics were analyzed:

Time based metrics: `E2E_time` measures end-to-end time for training the model. `epoch_time` measures the time taken for training per epoch. `GPU_compute_time` measures the time during which training occurs on GPU. For time logging purposes, we used `torch.cuda.Event` in conjunction with `torch.cuda.synchronize()`.

Utilization based metrics: `GPU_utilization` measures percent of time over the past sample period during which one or more kernels was executing on the GPU, `memory_usage` measures the ratio of the used to total available GPU memory, `memory_utilization` logs percent of time over the past sample period during which device memory was being read or written, `power_drawn` measures the last measured power drawn for the entire board, in watts (Nvidia, 2016).

4.3 Model Architectures

`baseline_ff_model`: We used the code on GitHub (Pezeshki, 2023) as our foundation, and made modifications to build the baseline models described in (Hinton, 2022). This model contains 4 fully connected layers with 2000 ReLUs. We also added relevant `Dataloader` for different datasets 4.1. Furthermore, for CIFAR100, we modified the `overlay` function to include a one-hot encoded label representation in the first 100 pixels of the image.

`baseline_backprop_model`: Since FF and backprop are two distinct algorithms , we decided to create architectures that catered to each algorithm differently. As a result, the backprop implementation consisted of a three layer neural net [`input` $\rightarrow$ `75` $\rightarrow$ `50` $\rightarrow$ `num_classes`] for SVHN, and [`input` $\rightarrow$ `250` $\rightarrow$ `110` $\rightarrow$ `num_classes`] for remaining datasets.

`hybrid_model`: The architecture of this model was dependent upon what algorithm was used for fine-tuning. If we used backprop as a weight initalizer, the architecture was similar to that of `baseline_ff_model` (4.3). On the other hand, if we used FF as a weight initalizer, the architecture was similar to that of `baseline_backprop_model` (4.3).

`baseline_CNN_model`: We adapted the same code (Pezeshki, 2023) to create custom classes like `Convolutional_layer`, `MaxPool_layer`, and `Flatten_layer`. These layers were similar to their traditional implementation with the addition of a goodness objective in the convolutional layer. Together, these classes were used to create the forward forward CNN implementation. The custom CNN model consisted of four layers. The first two layers were convolutional layers, each followed by max pooling and flattening. The last two layers were linear layers similar to the `baseline_ff_model` (4.3).

`baseline_attention_model`: We created an `Attention_Layer` by first chunking the input images into patches, feeding the patch embeddings into a `nn.MultiHeadedAttention` layer and finally passing them to a `nn.Linear` layer. The outputs of the attention layer was fed into the later layers of the `baseline_ff_model` (4.3)

5 Experiments and Results

Most of these experiments were carried out on Google Colab environment with standard GPU allocation and a few were replicated on USC CARC v100 compute nodes. We report the experiment results based on Google Colab execution only.

5.1 Comparison of Baseline FF with backprop

The goal of this experiment was to establish a baseline comparison between FF and backprop in their best modes of performance, and to analyze the system performance.

5.1.1 Setup

For the best performance mode in FF, we trained the `baseline_ff_model` (4.3) with 60 epochs for each layer. Here, an epoch means that the FF algorithm has *seen* the entire train dataset once. We used a batch size of 512 for both train and test. The training happens layer-wise – the first layer gets trained for 60 epochs followed by the second layer, and so on. Adam optimizer was used with a learning rate of 0.02 for each layer and the *sum of squared activities* as the goodness function. The threshold here was a hyperparameter which was finetuned for each dataset and we finally settled on a value of 15 for MNIST, 10 for FashionMNIST, SVHN, CIFAR10, and 1 for CIFAR100.

For backprop, we trained the `baseline_backprop_model` (4.3) for 20 epochs and used a batch size of 256 across the datasets for

both train and test. We utilized the *cross entropy* loss and the Adam optimizer with a learning rate of $6.7e^{-3}$. In this report, we present the best operating mode of these backprop models, as a result of which all but one of these models, corresponding to SVHN dataset, have the same architecture (Section 4.3).

5.1.2 Results

Accuracies: As seen from Table 1, FF performs at par with backprop for simple datasets such as MNIST and FashionMNIST. However, it lags behind backprop on more complex datasets such as CIFAR10, CIFAR100 and SVHN. Moreover, from the train accuracies (Table 1), it is evident that FF does not overfit the dataset as much as backprop does, i.e., it acts as an implicit regularizer.

E2E and GPU Compute times: From Figure 3, it can be observed that E2E times are similar for FF and backprop. For CIFAR100, the GPU Compute times for all but one layers of FF and backprop are quite similar, at around $\sim$ 1 sec. This is due to the higher number of parameters in the first layer, when compared to the rest. A similar observation has been made for CIFAR10 and SVHN datasets as well.

Memory usage and Power drawn: From Figure 4, it is observed that FF has higher memory usage than backprop. We attribute this to the way we have handled memory in our implementation of FF, which is not as efficient as the built-in implementation for backprop. Next, we observe that power drawn is similar for FF and backprop, except that the distribution for FF is not that tight, which can be attributed to the difference in the power consumption during per layer training and pre-processing phase.

5.2 Hybrid FF + backprop

Our goal here was to verify whether FF and backprop based approaches *sync-well* with each other, i.e. to test whether FF acts as a good initializer for backprop and vice-versa. Thus, we created a hybrid model and trained it in two directions.

5.2.1 Setup

First, we trained the `hybrid_model` (4.3) with FF for 60 epochs, and finetuned it with backprop for 20 epochs with a reduced learning rate of $1e^{-4}$. Next, we trained another `hybrid_model` with backprop for 20 epochs and finetuned it with FF for 60 epochs with a reduced learning rate of $1e^{-3}$.

5.2.2 Results

We observed a poor performance in both these experiments with accuracies ranging $\sim 10\%$. The model initialized with FF started with a very high training loss of the order $1e^3$ and could not be reduced significantly through the epochs. Similarly, the performance of the model, initialized by backprop and finetuned with FF, was just as bad. This could be attributed to the nature of objective functions of FF and backprop which do not gel well together.

5.3 Exploring FF with popular architectures

In this experiment, we wanted to explore the feasibility and efficacy of FF with popular architectures such as CNN and Self-Attention.

5.3.1 Setup

We trained `baseline_cnn_model` (4.3) on MNIST dataset using FF, for 1000 epochs, with a learning rate of 0.02. Furthermore, we trained the `baseline_attention_model` (4.3), for 60 epochs per layer, with learning rates $1e{-}3$, 0.02 for `nn.MultiHeadedAttention` and `nn.Linear` layers respectively.

5.3.2 Results

For the CNN model, we observed poor performance ($\sim 10\%$ accuracy), which confirms Hinton's conjecture that weight sharing is not feasible in FF, rendering CNNs ineffective.

For the attention based model, we saw a significant improvement over our CNN model ($\sim 60\%$ accuracy), however this was not comparable to baseline FF. We think this is mainly due to the way our overlay function is implemented. The overlay function passes the label information in the first few pixels of the image during training, and thus the baseline model learns to correlate this label with the rest of the image. However for the attention model, the label passed, gets split into patches and only one of the patches gets the label. Therefore, we think that sharing the label information with each patch could possibly boost the performance of the attention based model. We leave this exploration for future works (Section 7).

5.4 Comparison of FF with Linear and Attention Layers

Despite attention model not performing as well as the baseline model, we gauged the difference in system performance caused by the attention layer.

5.4.1 Setup

We trained `baseline_ff_model` (4.3) and `baseline_attention_model` (4.3) with FF on different datasets and logged the system metrics. Hyperparameters remain the same as in sections 5.1.1 and 5.3.1.

5.4.2 Results

GPU Utilization and Compute times: From Figure 5, for CIFAR100, we see that GPU Utilization for `baseline_ff_model` is unimodal (with one mode/lobe) whereas it is bimodal for `baseline_attention_model`. This can be attributed to larger GPU Compute times (hence utilization) for the attention layer. Moreover, we directly correlate these results with GPU Compute time plots to gain a more holistic view (5c, 5d). From the graphs, we observe that the lower lobe (5a, 5b) is concentrated around the same value ($\sim 35\%$) for both models, since the remaining layers are the same across these two models. Further, we observed similar results for all datasets.

Memory Utilization: Next, with regard to Memory Utilization, we observe a similar bimodal trend for `baseline_attention_model`, as can be seen for MNIST and CIFAR10 datasets in Figure 6. We explain this using a similar argument as done for GPU Utilization.

Power drawn: From Figure 7, for SVHN, we see that the power drawn is similar for both these models, but again note the prominent second lobe for `baseline_attention_model` (7b). A ready comparison of this with the time-series plot (7d) gives us the insight that this second lobe is due to the attention layer in `baseline_attention_model` which consumes higher power while training when compared to the linear layers. We conclude that even in architectures such as FF, training attention layer requires significantly more power than the linear ones. There is similar trend in results across datasets.

5.5 Comparison of Sample Complexity between FF and backprop

For this experiment we wished to explore how number of samples affects the performance of a given model trained with FF and backprop.

5.5.1 Setup

We varied dataset size by random subsampling, starting from the full train and test size for a given dataset, moving towards a minimum of 1000 train and test samples, in steps. This was done to explore the complexity across a spectrum of practical dataset sizes.

5.5.2 Results

From Tables 2 and 3 we observe that although FF performs slightly worse than backprop, it requires lesser number of samples to achieve similar accuracy as that of backprop and has a more stable performance when compared with backprop. By stability we imply that FF neither overfits nor underfits the subsampled datasets, while backprop overfits larger subsamples and underfits smaller subsamples. We proffer this as the major advantage that FF has over backprop.

6 Conclusion

In this work, we performed a principled performance comparison between FF and backprop in terms of accuracy and system metrics logged. We explored the feasibility of FF acting as a weight initializer for backprop and vice-versa. We then trained popular backprop architectures such as CNN and Self-Attention with FF, and examined the practicality of such implementations. Moreover, we compared the system performance of FF with Linear layers against its implementation with Self-Attention layers, and noted their differences. Finally, we compared the sample complexity between FF and backprop, and highlighted why FF has an edge over backprop.

7 Future works

We enlist the following avenues that can be explored as future work:

1. The layers of FF can be trained in a parallel manner rather than sequentially (Hinton, 2022), which could lead to better efficiency
2. Overlay information can be fed into each patch of the attention layer, which may improve performance
3. Different implementations of overlay can be tested that could improve performance of FF as a whole
4. The sensitivity of accuracy to threshold, and activation functions can be further investigated

	MNIST	FashionMNIST	SVHN	CIFAR10	CIFAR100
FF	97.31 | 97.14	87.63 | 85.95	82.32 | 76.28	54.85 | 49.24	19.93 | 13.63
backprop	99.94 | 98.04	94.67 | 89.43	86.38 | 80.52	81.48 | 53.88	52.21 | 23.75

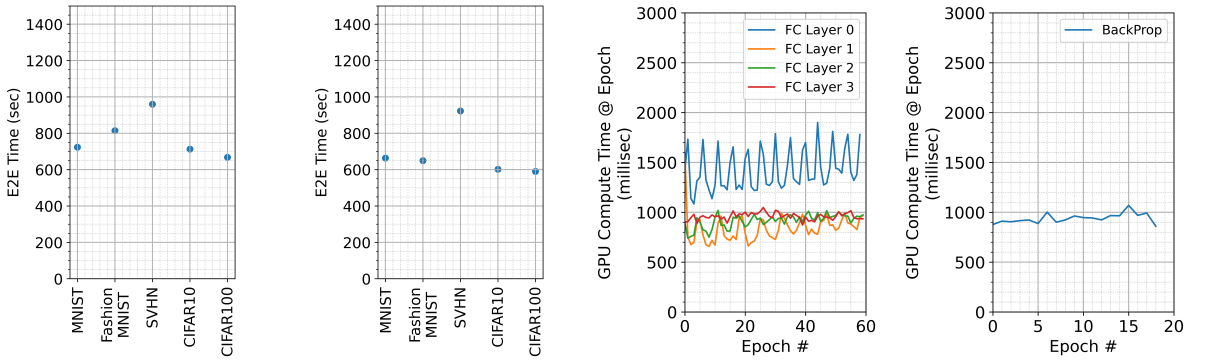
Table 1: Train | Test Accuracies (%) of FF and backprop

Train | Test Samples	**60k | 10k**	**50k | 10k**	**40k | 10k**	**30k | 10k**	**20k | 10k**	**10k | 10k**	**5k | 5k**	**1k | 1k**
MNIST	97.23 | 96.79	96.62 | 96.41	95.62 | 95.16	94.05 | 94.02	90.48 | 90.68	74.88 | 74.70	37.78 | 38.08	9.10 | 11.20
FashionMNIST	86.98 | 85.56	86.01 | 84.22	84.62 | 83.06	83.37 | 81.35	79.69 | 78.32	69.22 | 68.32	44.44 | 44.70	9.10 | 8.50
CIFAR10	-	48.77 | 45.52	45.27 | 42.49	40.73 | 38.73	34.83 | 34.02	27.69 | 27.82	17.42 | 16.46	12.90 | 12.50
CIFAR100	-	2.40 | 2.05	-	-	-	-	-	-
Train | Test Samples	**73k | 26k**	**63k | 26k**	**53k | 26k**	**43k | 26k**	**26k | 26k**	**13k | 13k**	**3k | 3k**	**1k | 1k**
SVHN	68.15 | 64.15	62.37 | 58.17	48.45 | 46.27	45.73 | 42.30	25.97 | 25.60	10.89 | 11.82	9.21 | 9.39	7.63 | 7.87

Table 2: Train | Test Accuracies (%) of FF for various number of samples

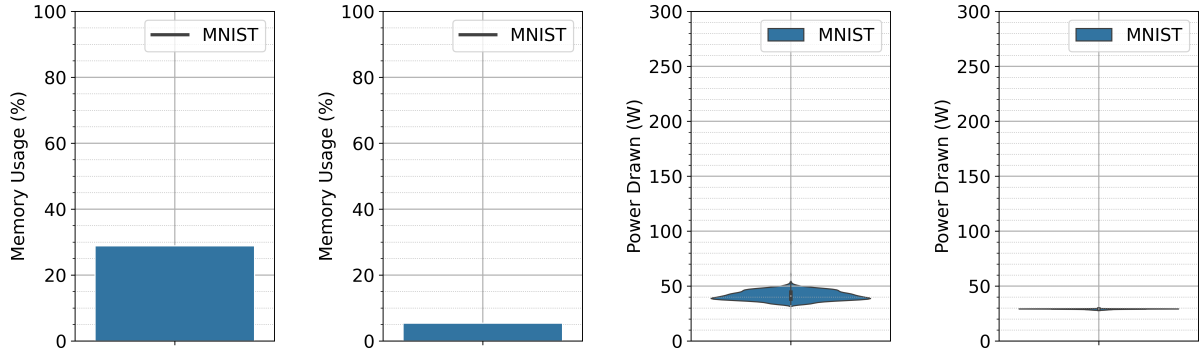
Train | Test Samples	**60k | 10k**	**50k | 10k**	**40k | 10k**	**30k | 10k**	**20k | 10k**	**10k | 10k**	**5k | 5k**	**1k | 1k**
MNIST	99.83 | 97.66	82.71 | 97.54	66.66 | 97.89	49.99 | 97.56	33.32 | 97.19	16.64 | 95.69	8.31 | 46.83	1.65 | 8.89
FashionMNIST	95.32 | 89.21	79.56 | 88.95	63.47 | 88.21	46.90 | 86.92	30.96 | 85.92	15.56 | 85.88	7.79 | 42.09	1.55 | 8.00
CIFAR10	-	83.81 | 52.98	67.81 | 52.67	51.93 | 50.00	35.47 | 47.61	17.49 | 44.44	9.50 | 20.62	1.89 | 3.60
CIFAR100	-	55.87 | 23.91	47.21 | 22.79	37.20 | 21.37	27.57 | 18.93	14.44 | 16.41	8.50 | 7.13	1.86 |0.84
Train | Test Samples	**73k | 26k**	**63k | 26k**	**53k | 26k**	**43k | 26k**	**26k | 26k**	**13k | 13k**	**3k | 3k**	**1k | 1k**
SVHN	87.35 | 81.00	75.61 | 80.66	61.65 | 77.18	50.91 | 78.19	30.54 | 76.31	15.42 | 37.11	3.64 | 7.48	1.36 | 2.35

Table 3: Train | Test Accuracies (%) of backprop for various number of samples



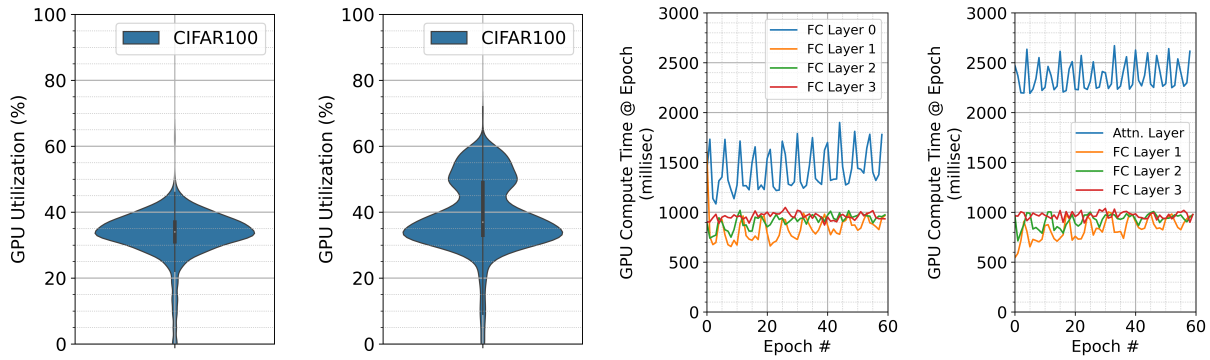
(a) FF | E2E time (b) backprop | E2E time (c) FF | GPU Compute time (d) backprop | GPU Compute time

Figure 3: FF vs backprop



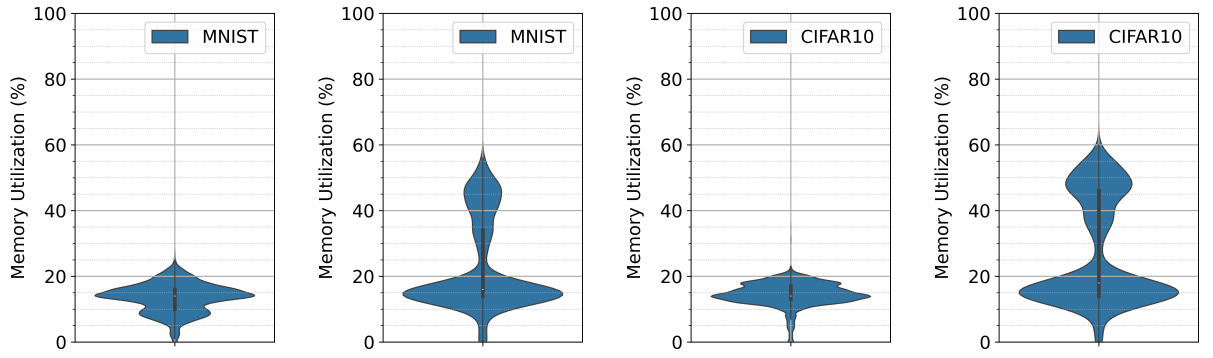
(a) FF (b) backprop (c) FF (d) backprop

Figure 4: Memory usage



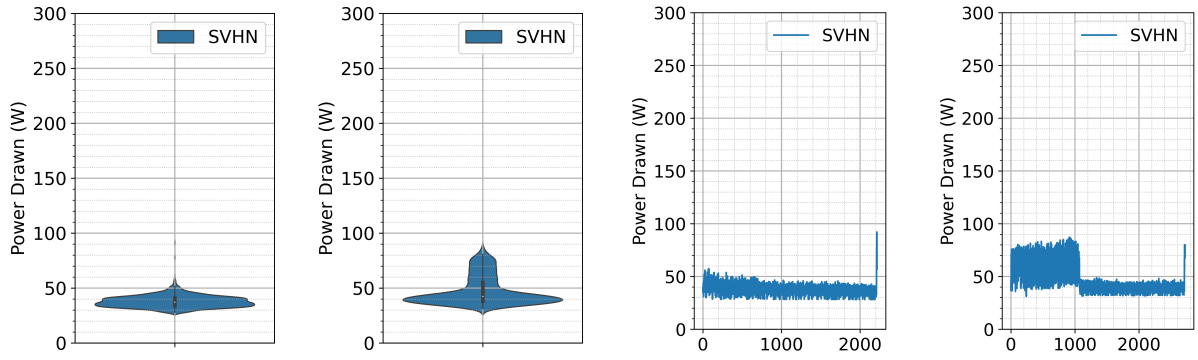
(a) Linear | GPU Util. (%) (b) Attention | GPU Util. (%) (c) Linear | GPU Comp. time (d) Attention | GPU Comp. time

Figure 5: FF with Linear vs Attention Layers | GPU Utilization and Compute time



(a) Linear | Mem. Util. (%) (b) Attention | Mem. Util. (%) (c) Linear | Mem. Util. (%) (d) Attention | Mem. Util. (%)

Figure 6: FF with Linear vs Attention Layers | Memory Utilization



(a) Linear | Power (W) (b) Attention | Power (W) (c) Linear | Power (W) (d) Attention | Power (W)

Figure 7: FF with Linear vs Attention Layers | Power Drawn

References

Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. 2020. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*.

Geoffrey Hinton. 2022. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*.

Geoffrey E Hinton, Terrence J Sejnowski, et al. 1986. Learning and relearning in boltzmann machines. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1(282-317):2.

Stephan Holly, Alexander Wendt, and Martin Lechner. 2020. Profiling energy consumption of deep neural networks on nvidia jetson nano. In *2020 11th International Green and Sustainable Computing Workshops (IGSC)*, pages 1–6.

Aakash Khochare, Sai Anuroop Kesanapalli, Rahul Bhope, Yogesh Simmhan, et al. 2022. Don't miss the train: A case for systems research into training on the edge. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 985–986. IEEE.

Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. Technical report.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Yann LeCun, Corinna Cortes, and CJ Burges. 2010. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

Jie Liu, Jiawen Liu, Wan Du, and Dong Li. 2019. Performance analysis and characterization of training deep learning models on mobile device. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 506–515.

Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. 2020. Mlperf training benchmark. *Proceedings of Machine Learning and Systems*, 2:336–349.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning.

Nvidia. 2016. nvidia-smi.txt.

Alexander Ororbia and Ankur Mali. 2023. The predictive forward-forward algorithm. *arXiv preprint arXiv:2301.01452*.

Mohammad Pezeshki. 2023. Mohammadpz/pytorch_forward_forward: Implementation of hinton's forward-forward (ff) algorithm - an alternative to back-propagation.

Prashanthi SK, Sai Anuroop Kesanapalli, and Yogesh Simmhan. 2022. Characterizing the performance of accelerated jetson edge devices for training deep learning models. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(3):1–26.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Hongchao Zhou. 2022. Activation learning by local competitions. *arXiv preprint arXiv:2209.13400*.

He Zhu, Yang Chen, Guyue Hu, and Shan Yu. 2022. Contrastive learning via local activity. *Electronics*, 12(1):147.

8 Acknowledgements

We thank Prof. Jesse Thomason, and the CSCI 566 staff, in particular, Deqing Fu, for their constant guidance and support throughout the duration of this work.

9 Code

Notebooks for various experiments performed in this work can be found at `https://github.com/ksanu1998/ff`.

Individual Contributions

Final report: Equal contributions
Code:

- SR, SAK - Implementing baseline FF
- KJ, AJ - Implementing backprop
- SAK, KJ - Implementing logging harness, plotting
- SAK, SR, AJ, KJ - Experimenting with datasets
- SR, AJ - Implementation of hybrid model
- AJ, KJ - Implementing baseline CNN
- SR, SAK - Implementing Attention model and comparision experiments
- SAK, SR - Sample complexity

Code, sections and parts of the report not listed above were equally contributed by all the members.

Name Legend:
SAK - Sai Anuroop Kesanapalli,
SR - Shashank Rangarajan,
KJ - Anukaran Jain,
AJ - Avtaran Jain