Optimising Neural Networks with Fine-Grained Forward-Forward Algorithm: A Novel Backpropagation-Free Training Algorithm

James Gong, Bruce Li, and Waleed Abdulla
University of Auckland, Auckland, New Zealand
{hgon777,tli389}@aucklanduni.ac.nz
w.abdulla@auckland.ac.nz

***Abstract*—This paper introduces the Fine-Grained Forward-Forward Algorithm (FGFF), a novel backpropagation-free network training methodology that builds upon the foundational Forward-Forward algorithm. FGFF takes full advantage of being entirely backpropagation-free and enhances neural network performance through the implementation of a fine-grained significance matrix. This matrix quantitatively assesses the contribution of each neuron's activity towards the overall goodness function, thereby facilitating more explorative and precise weight adjustments. The efficacy of the algorithm is demonstrated by comparing its accuracy with the vanilla Forward-Forward algorithm and the standard backpropagation methods. Our evaluations employ three benchmark datasets: MNIST, Fashion-MNIST, and CIFAR-10. The results indicate that the FGFF achieves superior performance over the Forward-Forward algorithm, specifically showing a testing accuracy improvement of up to 25.54% with no obvious cost in the computation time, underscoring its enhanced capability in handling complex pattern recognition tasks.**

***Index Terms*—Backpropagation-free, Forward-Forward, Supervised Learning, Mnist Classification**

I. Backpropagation limitations

The backpropagation algorithm has been fundamental to the advancement of deep learning, yet it is encumbered by significant limitations that challenge both its practical application and biological plausibility. Firstly, the backpropagation algorithm is biologically implausible, as there is scant evidence to suggest that the human cortex propagates error derivatives backward or retains neural activities for subsequent processing [1]. Additionally, backpropagation necessitates a complete understanding of the forward pass to accurately calculate error derivatives, rendering it ineffective when the forward mechanism is treated as a black-box [2]. These limitations not only necessitate stringent prerequisites for deploying deep learning models but also challenge the prevailing notion that emulating the biological brain processes is a viable pathway toward achieving Artificial General Intelligence (AGI) [3].

II. The Forward-Forward Algorithm

Recent research has explored various backpropagation-free techniques for training neural networks, aiming to overcome the limitations of traditional backpropagation, such as its biological implausibility and computational inefficiency. Notable approaches include Feedback Alignment [4]–[7], and Hebbian Learning [8]–[10]. These methods offer various advantages but sometimes involve partial backpropagation and complex mechanisms. Additionally, a significant amount of work has been done to enhance the Forward-Forward algorithm, adding substantial value to its core methodology [11], [12].

The Forward-Forward algorithm [13] is an entirely backpropagation-free, greedy, and layerwise approach to training neural networks by replacing the forward and backward passes with two distinct forward passes, as indicated in Fig. 1. In the first pass, positive data are used, which are generated through the correct one-hot encoding of input data, and each instance in the dataset is accurately encoded with its true label, ensuring the network learns from correctly encoded examples. In contrast, the second pass processes negative data generated by intentionally encoding the input data with the incorrect label. Fig. 2 illustrates examples of positive data and negative data from the MNIST [14] dataset, where the indices of the first 10 pixels are employed to represent the labels for the data. Within this framework, the algorithm computes two specific measures of activity: positive activity and negative activity. Positive activity is derived by squaring the output of each neuron during the positive pass, which emphasises the activation levels when the data is correctly labelled. Conversely, negative activity is calculated identically during the negative pass, reflecting the neuron's response to incorrectly labelled data. Then, in each layer, the algorithm aims to train the mean of the positive activities far above a threshold value and the mean of the negative activities far below that.

Therefore, the loss function can be written as:

$$L_j = \frac{1}{2}\max\Big(0, \delta - (\mathrm{g_{pos;i}} - \text{threshold})\Big) + \max\Big(0, \delta - (\text{threshold} - \mathrm{g_{neg;i}})\Big) \quad (1)$$

$$g_{\text{pos, neg;i}} = \frac{1}{n}\sum_{j=1}^{n} x_{ij}^2 \quad (2)$$

where a hinge loss is employed to push the positive goodness and the negative goodness in the current layer to be at least $2 \times \delta$ apart centred around the threshold value. $g_{pos,neg;i}$

stands for the positive and negative goodness value calculated in the current layer i. Following the calculation of the loss, appropriate adjustments are made to the respective weights and biases using gradient descent. To ensure that large activation values from the current layer do not overly influence the activity values in the subsequent layer, L2 normalisation is applied to all output values before they enter the next layer.

During prediction, the data are initially labelled with all possible categories. After that, these data are passed into the network, and the category of data generating the highest activity sum also referred to as the goodness value, is interpreted as the outcome of the network.

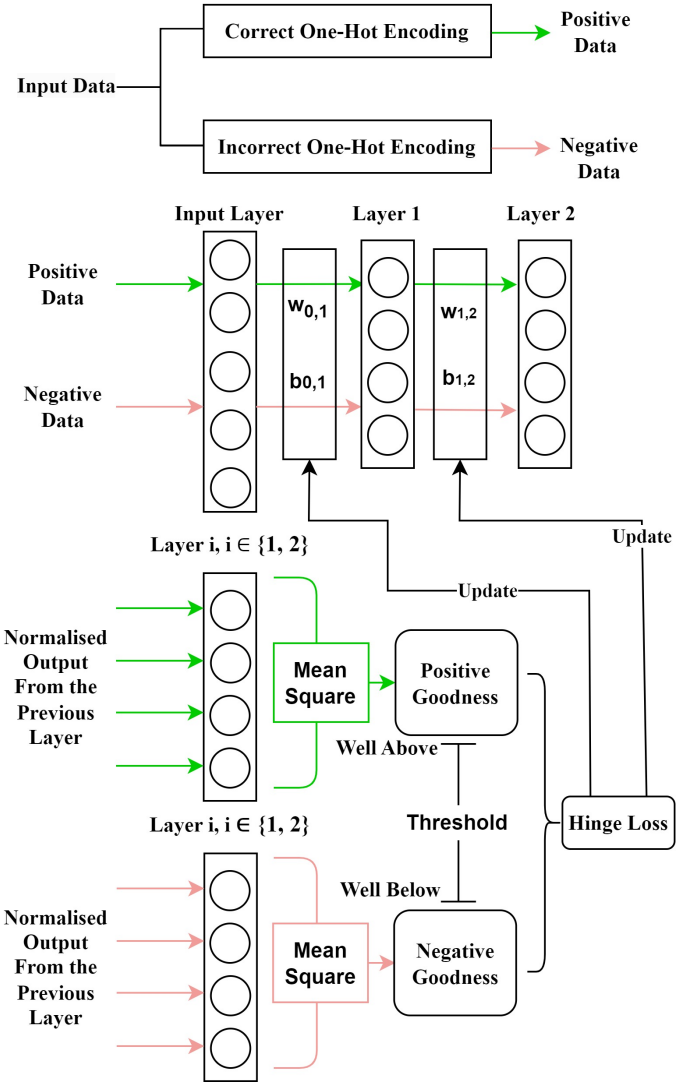
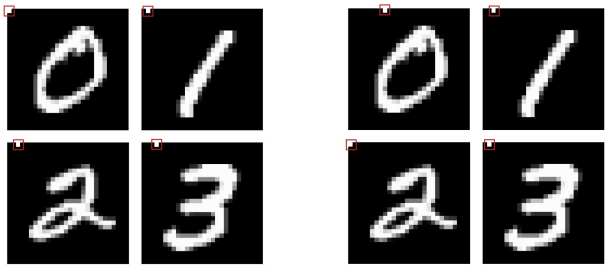


Fig. 1: Two distinct forward passes in the Forward-Forward algorithm. The green arrows demonstrate the positive forward pass with positive data input. The red arrows demonstrate the negative pass with negative data input. Layer i represents any hidden layer in the network, where i can be either 1 or 2.

From a biological perspective, the brain tends to react more vigorously to the patterns previously learned and less rigorously to the patterns they are not familiar with. Hence, the rigorousness of the neural activity is taken to compute and classify the output when a combination of patterns is presented.

However, an inherent assumption of the Forward-Forward algorithm is that each neuron and each layer contributes identically to the overall goodness function. In fact, earlier layers tend to capture more abstract features compared to the more concrete features learnt by the later layers [17]. Furthermore, within the same layer, the significance of each neuron's contribution can vary considerably. This raises critical questions about the weighting of individual neuron activities: How significant is each neuron, and to what extent should the activity value generated by each neuron influence the overall goodness function? These concerns are central to the design of FGFF, which are addressed in Section III.



(a) Positive Data Sample (b) Negative Data sample

Fig. 2: Positive and negative data visualization for the Forward-Forward algorithm. The first 10 pixels (top left corner) in the images are used for one-hot encoding. For positive data samples, the correct index corresponding to the label is encoded as white; for example, if the label is 0, the first pixel (index 0) is encoded as white. In negative data samples, the indices are randomly encoded while excluding the correct index. The encoded index has been highlighted.

III. The Fine-Grained Forward-Forward Algorithm

The Fine-Grained Forward-Forward algorithm (FGFF) builds upon the Forward-Forward algorithm by introducing a key novel element: **neuron activity scaling factors**, also referred to as the fine weights or the significance matrix in Fig. 3. These scaling factors are pivotal in modulating the activity values of neurons within each layer. By appropriately assigning these scaling factors, the concerns regarding the variable significance of individual neurons and their influence on the overall goodness function are directly addressed. The pseudo-code for FGFF is presented in Algorithm 1.

A. Introducing novel calculation of layer-wise goodness function

The neuron activity scaling factors parameter is structured as a 2D tensor within the neural network architecture. It is defined such that the number of rows corresponds to the number of

hidden layers in the network, and the number of columns in each row corresponds to the number of neurons in each respective layer. For instance, in a network with two hidden layers, each containing 500 neurons, the tensor is initialised as a shape of 2 by 500 filled with 1[1]. This 2D tensor arrangement allows for independent scaling of neuron activities across different layers during training. As the network learns from repeated exposure to stimuli, these values dynamically adjust, optimising the responsiveness and adaptability of the network over time.

Similar to the Forward-Forward algorithm, FGFF employs one loss function across each layer with the core objective remaining unchanged: to train the network such that the mean of the positive activities significantly exceeds a specified threshold, while the mean of the negative activities falls well below it. However, in FGFF, the activity values are now scaled by the corresponding neuron activity scaling factors w_{ij}, hence the new layer-wise goodness function is written as:

$$g_{\text{pos, neg};i} = \frac{1}{n}\sum_{j=1}^{n} x_{ij}^2 w_{ij} \tag{3}$$

where w_{ij} indicates the scaling factor for the j_{th} neuron in the i_{th} layer, and x_{ij} stands for the activity output of the neuron in the layer.

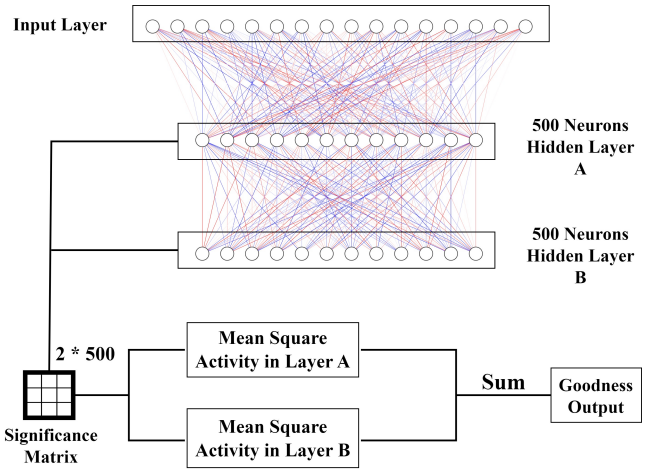


Fig. 3: A sample neural network architecture used in the Fine-Grained Forward-Forward Algorithm. The significance matrix is another terminology for the collection of the neuron activity scaling factors.

B. Learning in neuron activity scaling factors

The Fine-Grained Forward-Forward algorithm refines the traditional training process of FF by incorporating the neuron activity scaling factors. These factors are crucial trainable parameters that dynamically adjust each neuron's influence on the network's goodness function. The addition of these neuron activity scaling factors leads to an increase in the total number of trainable parameters, which is directly proportional to the product of the number of hidden layers $(m-1)$ and the number of neurons per layer (n). However, this increase is minimal compared to the core architectural parameters of the network, which amount to $(m-1) \times n^2$.

During training, each neuron outputs an activation value as inputs propagate through the network. These activations are squared and then scaled by the corresponding scaling factors. Subsequently, the scaled activations are averaged to determine the positive and negative goodness values, central to the algorithm's loss function, see equation (1). This loss function is recalculated for each epoch to ensure accurate gradient descent based on the current network state.

Updates to the neuron activity scaling factors are governed by the derivative of the loss function. From a mathematical standpoint, the adjustments made to the neuron activity scaling factors enhance the distinction between positive and negative goodness values. This increased disparity aids in sharpening the decision boundaries of the network, resulting in more precise classifications. Moreover, by incorporating the scaling factors, the algorithm fosters new connections between neighboring solutions in the solution space. This additional connectivity in the solution space reduces the number of local optima, decreasing the likelihood that the training process becomes trapped in shallow local optima. Consequently, this results in a more effective and exploratory training regime, promoting a robust learning environment that approaches better solutions.

C. Impact of neuron activity scaling factors

A critical aspect of FGFF is the role of neuron activity scaling factors during both training and prediction phases. These scaling factors enhance the learning process by allowing the network to assign varying levels of importance to different neurons, leading to a more nuanced and effective learning of patterns over the epochs.

Interestingly, even when the scaling factors are discarded during the prediction phase—utilised solely during training—the model still demonstrates performance comparable to scenarios where scaling factors are used thoroughly. This indicates that the network successfully learns the underlying patterns during training, rather than relying heavily on the scaling factors for decision-making during prediction. This robustness is particularly beneficial for a backpropagation-free neural network training algorithm like FGFF. By simplifying the prediction phase and eliminating the need for scaling factors, the FGFF achieves prediction efficiency identical to the original Forward-Forward algorithm but delivers more accurate results during deployment.

D. The fine weights optimizer

In contrast to the Forward-Forward algorithm, FGFF features two separate optimizers: one is responsible for updating the parameters of the neural network layer (weights and

[1]The scaling factors are initialized to values of one to align the fine-grained forward-forward algorithm with the standard forward-forward algorithm at the outset. This initialization ensures that, in the absence of modifications to the scaling factors, the algorithm reverts to the behavior of the original forward-forward algorithm.

biases), and another for adjusting the fine weights. This dual optimization approach introduces an additional hyperparameter—the learning rate for the fine weights optimizer.

By setting the learning rate for the fine weights optimizer to zero, FGFF disables any updates to the neuron activity scaling factors, resulting in these factors remaining at their initial value of 1. Consequently, FGFF reverts to the standard Forward-Forward algorithm. Therefore, with appropriate tuning of this hyperparameter, the worst-case performance of FGFF is equivalent to the best performance achievable by the Forward-Forward algorithm. This ensures that FGFF, at a minimum, performs no worse than the FF algorithm, while offering the potential improvements in accuracy across all datasets.

E. Mathematical parallels with feature and constraint violation detectors

In section 10 (future work) of the Forward-Forward algorithm paper [13], Hinton proposed the potential benefit of incorporating a set of feature detectors to maximise their squared activity and a set of constraint violation detectors to minimise their squared activity [18]. While the Fine-Grained Forward-Forward Algorithm (FGFF) does not explicitly employ feature detectors, its design mathematically aligns with this proposed concept. Specifically, the neuron activity scaling factors in FGFF are optimised in such a way that they maximise the squared activity of some neurons, akin to feature detectors, and minimise the squared activity of other neurons, similar to constraint violation detectors.

IV. Experiments

This section demonstrates the performance analysis of FGFF and compares it with the Forward-Forward algorithm and the traditional backpropagation algorithm on three benchmark datasets: MNIST, FashionMNIST [15], and CIFAR-10 [16].

A. Performance study of FGFF

A number of network architectures were trained on multiple datasets to evaluate the effectiveness of FGFF, FF, and BP training algorithms. This includes MNIST, FashionMNIST, and CIFAR-10.

Model	BP	FF	FGFF
2x50 Relu	97.65%	74.71%	87.13%
2x500 Relu	98.27%	85.82%	90.88%
2x1000 Relu	98.75%	85.83%	92.95%

TABLE I: Test accuracies (%) on MNIST dataset for backpropagation (BP), Forward-Forward (FF), and fine-grained Forward-Forward (FGFF) algorithms.

The results on MNIST are summarized in Table I. FGFF consistently outperformed FF across all architectures.

The results in Table II demonstrate that increasing the number of neurons in each layer tends to improve model performance on the Fashion-MNIST dataset, similar to the observations in Table I for MNIST.

Algorithm 1 The Fine-Grained Forward-Forward Algorithm

Initialize the fine-grained weight matrix w_f of dimension $m \times n$ with all ones, where m is the number of layers excluding the input layer, and n is the number of neurons in such layers.
$epochs \leftarrow 200$
$i, j \leftarrow 0$
$\mathbf{x}$, target_labels $\leftarrow$ load data
positive_data, negative_data $\leftarrow$ create data # use one-hot encoding
while $i < m$ **do**
    # Greedily train each layer, this process can be parallelized
    while $j < epochs$ **do**
        $o_p \leftarrow$ Forward(positive_data) ▷ positive output
        $o_n \leftarrow$ Forward(negative_data) ▷ negative output
        $w_{f;i} \leftarrow w_f[i, :]$
        # $Square(o_p, o_n)$ squares each element in the tensors o_p and o_n
        $g_{\text{pos}}, g_{\text{neg}} \leftarrow$ Mean(Square$(o_p, o_n) \times \mathbf{w}_{f;i}$)
        $L \leftarrow$ Loss$(g_{\text{pos}}, g_{\text{neg}})$
        $\delta w_f \leftarrow \alpha_f \frac{dL}{dw_f}$, $\delta w \leftarrow \alpha \frac{dL}{dw}, \delta b \leftarrow \frac{dL}{db}$
        # Perform weights and biases update based on L
        $w_f \leftarrow w_f - \delta w_f$ ▷ the fine-grained significance matrix
        $w \leftarrow w - \delta w$ ▷ weightings for interconnecting neurons
        $b \leftarrow b - \delta b$ ▷ biases for the neurons
        $j \leftarrow j + 1$
    end while
    positive_data $\leftarrow$ Forward(positive_data)
    negative_data $\leftarrow$ Forward(negative_data)
    $i \leftarrow i + 1$
end while

function FORWARD(x)
    $x_n \leftarrow$ Normalize(x) ▷ L2 normalization
    output $\leftarrow x_n \times W^T$
    return RELU(output)
end function

B. Experiments on CIFAR-10

Due to the intrinsic complexities of the CIFAR-10 dataset, the experiment employed a neural network with three layers, each containing 800 neurons. The learning rate was set to 0.01, and the training was conducted for 1200 epochs. The results of this experiment are presented in Table IV.

Notably, achieving comparable accuracy with FGFF necessitated an additional 1800 epochs for FF, highlighting the efficiency of FGFF in training neural networks on complex datasets such as CIFAR-10.

Model	BP	FF	FGFF
2x50 Relu	87.52%	60.76%	73.28%
2x500 Relu	89.34%	52.24%	75.63%
2x1000 Relu	89.47%	51.64%	77.18%

TABLE II: Test accuracies (%) on Fashion-MNIST dataset for back-propagation (BP), Forward-Forward (FF), fine-grained Forward-Forward (FGFF), and tuned FF models with various architectures.

Algorithm	Train Acc	Test Acc
BP	90.12%	52.08%
FF	38.33%	37.67%
FGFF	50.73%	42.67%

TABLE III: Train accuracies (%) and Test accuracies (%) on CIFAR-10 dataset for back-propagation (BP), Forward-Forward (FF), fine-grained Forward-Forward (FGFF).

C. Epoch time analysis

In this section, we evaluate the performance of FGFF relative to FF and the conventional BP by comparing the time taken per epoch for each algorithm. The findings of these comparisons are detailed in Table IV.

Algorithm	Time taken per epoch
BP	13.1754 s
FF	0.1066 s
FGFF	0.1073 s

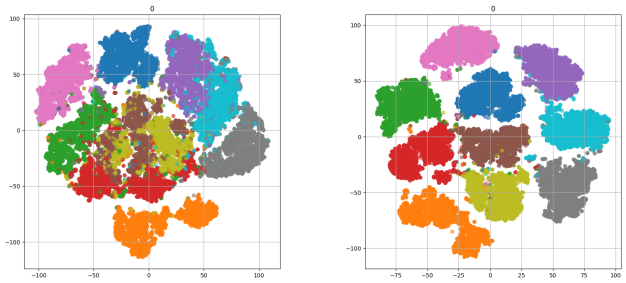
TABLE IV: Time taken per epoch (s) on Fashion-MNIST for back-propagation (BP), Forward-Forward (FF), fine-grained Forward-Forward (FGFF). The training was conducted using an NVIDIA RTX GeForce 4060 GPU on Windows 11.

BP requires a significantly longer time per epoch, as shown in the table. This extended time requirement is due to the complex nature of BP, which involves precise and symmetric weight updates that must be calculated and applied during training. These updates, while effective, are biologically implausible because such precision and symmetry are not feasible in biological neural networks. In contrast, FF and FGFF are designed to be more aligned with how learning occurs in the brain, making them biologically plausible.

From Table IV, we observe that FGFF takes approximately 0.1073 seconds per epoch, which is only around 0.65% slower than FF, which takes 0.1066 seconds per epoch. Despite the slight increase in time, FGFF's performance can be significantly better, achieving up to 25.54% improvement in accuracy as shown in Table II. This highlights FGFF's efficiency and effectiveness, making it a highly practical choice in scenarios where both computational efficiency and performance are critical.

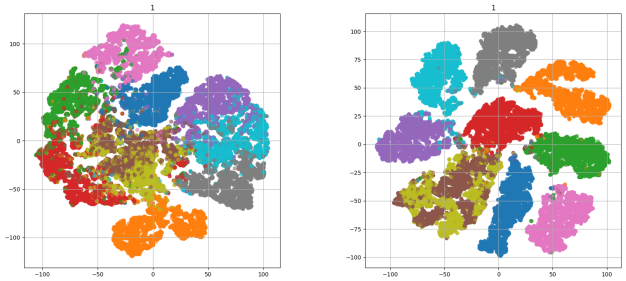
D. t-SNE visualisation

To have a deeper understanding of how our model processes and transforms the input data through each layer, and compare the feature learning capabilities of FGFF with FF, a neural network of 2x500 ReLU was trained on the MNIST dataset. The input data and layer-wise output feature map is visualised using t-SNE [19], in Fig. 4 and Fig. 5. During the training, the activations (goodness) are calculated. After training a layer, the positive goodness values are visualised.



(a) FF (b) FGFF

Fig. 4: t-SNE visualisations of first hidden layer output feature maps of positive data after 200 epochs. Different colors correspond to different classes



(a) FF (b) FGFF

Fig. 5: t-SNE visualisations of the second hidden layer output feature maps of positive data after 200 epochs. Different colors correspond to different classes

The feature maps reveal that both methods effectively learn features that distinguish between classes. However, FGFF demonstrates a more distinct data separation compared to FF.

V. Conclusion

In this paper, we introduced the Fine-Grained Forward-Forward algorithm (FGFF). A novel backpropagation-free training methodology that builds upon the Forward-Forward (FF) algorithm. FGFF incorporates neuron activity scaling factors, enabling more precise and effective weight adjustments. Our experiments on MNIST, FashionMNIST, and CIFAR-10 datasets demonstrated that FGFF consistently outperforms FF

in accuracy and efficiency, stepping closer to the traditional backpropagation algorithm (BP). Overall, FGFF represents a significant advancement in backpropagation-free neural network training, providing a viable alternative to traditional methods with potential applications in various domains.

REFERENCES

[1] Song, Yuhang, et al. "Can the Brain Do Backpropagation?—Exact Implementation of Backpropagation in Predictive Coding Networks." *Advances in Neural Information Processing Systems*, 33 (2020): 22566-22579.

[2] Rumelhart, David E., Hinton, Geoffrey E., & Williams, Ronald J. (1986). "Learning representations by back-propagating errors." *Nature*.

[3] Lake, Brenden M., Ullman, Tomer D., Tenenbaum, Joshua B., and Gershman, Samuel J. (2017). "Building Machines That Learn and Think Like People." *Behavioural and Brain Sciences*.

[4] Nøkland, Arild. (2016). "Direct feedback alignment provides learning in deep neural networks." *Advances in Neural Information Processing Systems*, 29.

[5] Crafton, Brian, Parihar, A., Gebhardt, Evan, and Raychowdhury, A. (2019). "Direct feedback alignment with sparse connections for local learning," *Frontiers in Neuroscience*, vol. 13.

[6] Lillicrap, Timothy P., Cownden, D., Tweed, D., and Akerman, C. (2016). "Random synaptic feedback weights support error backpropagation for deep learning," *Nature Communications*, vol. 7.

[7] Nøkland, Arild. (2016). "Direct feedback alignment provides learning in deep neural networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, NIPS'16, p. 1045–1053, Curran Associates Inc.

[8] Krotov, Dmitry, and Hopfield, John J. (2019). "Unsupervised learning by competing hidden units," *Proceedings of the National Academy of Sciences*, vol. 116, no. 16, pp. 7723–7731.

[9] Amato, G., Carrara, Fabio, Falchi, F., Gennaro, C., and Lagani, Gabriele. (2019). "Hebbian learning meets deep convolutional neural networks," in *ICIAP*.

[10] Gupta, M., Ambikapathi, A., & Ramasamy, S. (2021). "HebbNet: A Simplified Hebbian Learning Framework to Do Biologically Plausible Learning." In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 3115-3119). IEEE. doi:10.1109/ICASSP39728.2021.9414241.

[11] Gandhi, S., Gala, R., Kornberg, J., & Sridhar, A. (2023). Extending the Forward Forward Algorithm. arXiv preprint arXiv:2307.04205.

[12] Ororbia, A., & Mali, A. (2023). The Predictive Forward-Forward Algorithm. arXiv preprint arXiv:2301.01452.

[13] Hinton, Geoffrey. (2022). "The Forward-Forward Algorithm: Some Preliminary Investigations." Google Brain.

[14] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

[15] Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. Retrieved from https://github.com/zalandoresearch/fashion-mnist

[16] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

[17] Goodfellow, Ian; Bengio, Yoshua; & Courville, Aaron. (2016). "Deep Learning." MIT Press.

[18] Welling, M., Williams, C., and Agakov, F. (2003). "Extreme components analysis." *Advances in Neural Information Processing Systems*, 16.

[19] van der Maaten, L.J.P., and Hinton, G.E. (2008). "Visualizing high-dimensional data using t-sne." *Journal of Machine Learning Research*, 9:2579–2605.