Forward Target Propagation: A Forward-Only Approach to Global Error Credit Assignment via Local Losses

Anonymous authors
Paper under double-blind review

Abstract

Training neural networks has traditionally relied on backpropagation (BP), a gradient-based algorithm that, despite its widespread success, suffers from key limitations in both biological and hardware perspectives. These include backward error propagation by symmetric weights, non-local credit assignment, update locking, and frozen activity during backward passes. We propose Forward Target Propagation (FTP), a biologically plausible and computationally efficient alternative that replaces the backward pass with a second forward pass. FTP estimates layer-wise targets using only feedforward computations, eliminating the need for symmetric feedback weights or learnable inverse functions, hence enabling modular and local learning. We evaluate FTP on fully connected networks, CNNs, and RNNs,demonstrating accuracies competitive with BP on fully connected and sequential tasks, and consistently outperforming existing forward-only and target-propagation baselines across all architectures, including convolutional networks on CIFAR-10 and CIFAR-100. In simulated low-precision and noisy-weight settings that emulate analog in-memory hardware, FTP shows improved robustness relative to BP, and Multiply-and-Accumulate (MAC)-count analysis indicates substantially lower computational cost than other biologically inspired methods such as target propagation variants and forward-only learning algorithms. With its minimal computational overhead, forward-only nature, and hardware compatibility, FTP provides a promising direction for energy-efficient on-device learning and neuromorphic computing.

1 Introduction

Backpropagation (BP) has been the foundational algorithm for training neural networks, driving the success of deep learning in tasks such as image recognition, language modeling, and decision-making (Rumelhart et al., 1986; Vaswani et al., 2017; Brown et al., 2020). Despite its proven effectiveness, BP faces key limitations in biological plausibility and hardware compatibility. A major issue is its reliance on symmetric weight transport between forward and backward passes, which conflicts with biological learning, where synaptic updates are not symmetric (Whittington & Bogacz, 2019). Moreover, on edge devices and emerging analog in-memory hardware such as resistive random access memory (RRAM) and phase-change memory (PCM) crossbars, symmetric transport poses further challenges (Yi et al., 2023; 2022; Shafiee et al., 2016). Each training iteration requires weight matrices to be written and verified in transposed form for backward error propagation. However, achieving perfect forward–backward symmetry in analog hardware is often unfeasible (Li et al., 2018), leading to performance degradation. This issue is amplified in edge applications, where low-bit precision constraints worsen the problem (Yang & Sze, 2019). Additionally, BP propagates a global error signal backward through multiple layers, unlike the localized learning signals of biological neural systems (Lillicrap et al., 2020). Its weight updates depend on information from distant layers, implying neurons must access far-off signals, a non-locality absent in biology. BP also suffers from vanishing and exploding gradients, where values become too small for meaningful updates or too large, destabilizing training, especially in deep and recurrent networks (Bengio et al., 1994). Furthermore, BP assumes layer-wise processing and exact error computation, whereas biological systems learn in a noisier, more heuristic manner rather than through precise gradients (Crick, 1989; Scellier et al., 2023). These challenges have driven interest in biologically plausible alternatives that better capture how the brain may perform credit assignment (Lillicrap et al., 2020;

2016; Bengio, 2014; Hinton, 2022; Millidge et al., 2021). This paradigm is worth exploring because the brain achieves remarkable efficiency and adaptability through local learning rules, providing valuable insights for developing energy-efficient learning algorithms.

In this work, we introduce Forward Target Propagation (FTP), a biologically inspired, forward-only learning framework that addresses key limitations of backpropagation and demonstrate its theoretical basis, empirical competitiveness, efficiency across architectures and tasks, scalability among forward learning approaches, and compatibility with emerging hardware. Our key contributions are summarized as:

- We introduce Forward Target Propagation (FTP), a biologically plausible, forward-only learning algorithm that enables local credit assignment without relying on backward error propagation or symmetric weight transport.
- We evaluate FTP across fully connected, convolutional, and recurrent neural networks on image classification and multivariate time-series forecasting tasks, showing that FTP matches or exceeds the recent bioplausible baselines at lower computational cost, approaches BP on fully connected and recurrent tasks, and narrows the gap to BP on convolutional benchmarks.
- We present theoretical justification and empirical evidence showing an steady alignment between FTP and backpropagation, measured via the angular similarity between their respective gradient directions during training.
- We assess FTP's robustness under simulated low-precision and noisy-weight conditions, and estimate its computational cost on TinyML benchmarks, showing that FTP attains near-BP-level Mutiply-and-Accumulate (MAC) counts while requiring substantially fewer operations than other biologically plausible algorithms. These results suggest its suitability for edge and neuromorphic computing.

2 Related Work

2.1 Conventional and Biologically Plausible Methods

Backpropagation (BP) has long been the backbone of neural network training, relying on forward and backward passes to compute error gradients. In the forward pass, inputs propagate through the network to the output layer, where prediction errors are computed against target values. The backward pass then sends error derivatives through the network using the same weights used in the forward pass, and the outer product of activity vectors from both passes forms the gradient matrices of the global objective. Despite its success, BP is often criticized for its biological implausibility, raising questions about whether such mechanisms exist in real biological systems (Rumelhart et al., 1986). To address the limitations of backpropagation, several biologically inspired learning algorithms have been proposed. Feedback Alignment (FA) (Lillicrap et al., 2016) replaces symmetric feedback with fixed random weights, demonstrating that networks can learn without exact weight mirroring. Direct Feedback Alignment (DFA) extends this idea by projecting output errors directly to hidden units via random matrices (Nøkland, 2016). A simplified variant, Direct Random Target Projection (DRTP), further removes the need for global error signals by using target labels to update hidden layers (Frenkel et al., 2021). Although these methods mitigate issues such as weight transport and update locking, they remain limited in biological plausibility due to their reliance on global error signals and frozen network dynamics. Moreover, DRTP has shown reduced accuracy and poor scalability compared to more recent approaches.

Equilibrium Propagation offers a biologically inspired alternative to backpropagation by computing gradients through equilibrium states in energy-based models (Scellier & Bengio, 2017; Scellier et al., 2023; Yi et al., 2023). Target Propagation (TP) and Difference Target Propagation (DTP) assign credit via layer-wise targets rather than global error gradients, thus avoiding a full backward pass (Bengio, 2014; Lee et al., 2015; Meulemans et al., 2020; Ernoult et al., 2022). While DTP improves target estimation with correction terms, both methods become costly in deep networks. Local Representation Alignment (LRA-E) refines this idea by computing local targets from downstream errors through fixed feedback pathways, removing the need for gradients or symmetric weights (Ororbia & Mali, 2019). Its recursive variant, rec-LRA, scales this mechanism by aligning internal representations layer-by-layer, enabling parallel updates in deep architectures (Ororbia et al., 2023). Though effective on large datasets, both methods introduce additional computational and

memory overhead and need careful tuning for stability. Fixed Weight DTP (FWDTP) simplifies DTP with fixed random feedback matrices but suffers performance degradation in deeper networks (Shibuya et al., 2023).

These biologically inspired training schemes aim to match BP's performance while mitigating the weight transport problem. However, challenges such as non-local credit assignment, frozen activity, update locking, and high computational cost remain, underscoring the need for further exploration in this domain.

2.2 Forward Learning Methods

Forward learning methods offer a biologically plausible alternative to backpropagation by avoiding explicit backward passes and instead utilizing feedforward mechanisms to propagate learning signals. PEPITA is one such forward learning method (Dellaferrera & Kreiman, 2022). It avoids backpropagation by performing a second forward pass where inputs are modulated by output errors through a fixed random matrix (Dellaferrera & Kreiman, 2022). Weight updates come from the difference between standard and modulated activations, enabling local learning without symmetric weights. This removes the weight symmetry constraint of backpropagation but faces the issue of scalability beyond two hidden layers. Forward-Forward Learning (FF), introduced by Hinton (Hinton, 2022), eliminates backward error propagation by training solely with forward passes. It compares neuron activations under "positive" and "negative" data, reinforcing those aligned with the desired output. By avoiding backward gradients, FF addresses weight transport and frozen activity. Yet the absence of top-down influence limits inter-layer coordination, partly alleviated by adding recurrent connections at the expense of higher computational complexity. FF also struggles to scale and underperforms beyond simple fully connected (FC) networks, such as convolutional neural networks (CNNs) (Lv et al., 2025). Predictive Forward-Forward Learning (PFF) extends FF with a generative circuit that predicts neural activity to guide local updates (Ororbia & Mali, 2023). Although more biologically plausible and less reliant on error feedback than predictive coding, PFF incurs extra cost from repeated prediction steps.

Despite these advancements, forward learning methods remain limited in efficiency, performance, and scalability. Although they address key biological implausibilities of BP such as symmetric weight transport and global error gradients, they often introduce new issues, including greater computational demands and poor scaling in deeper networks.

3 Forward Target Propagation

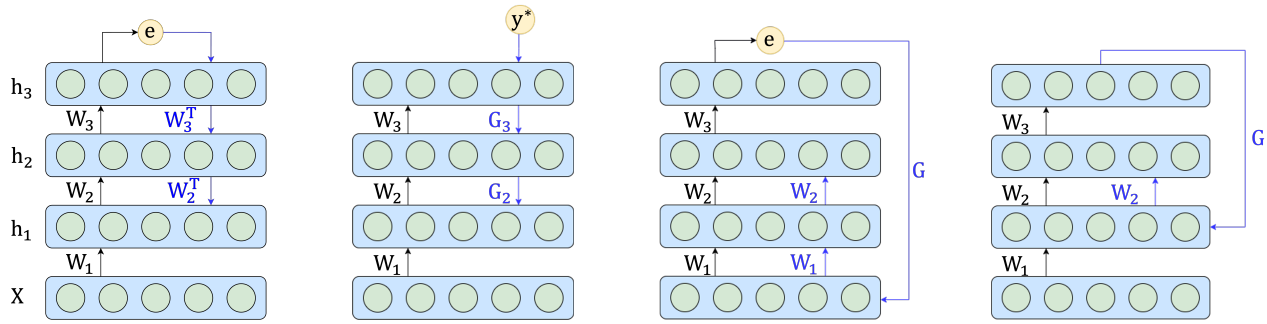
3.1 Overview of Algorithm

Forward Target Propagation (FTP) is a novel algorithm that replaces the backward pass in neural networks with a second forward pass to compute learning signals. At the core of FTP lies the concept of layer-wise targets similar to target propagation algorithms (Bengio, 2014; Lee et al., 2015; Meulemans et al., 2020; Shibuya et al., 2023), which define the desired activations that each layer should produce during learning. The process begins with a standard forward pass (first forward pass), where the input data $\boldsymbol{X}$ is propagated through the network to generate the output activation $\boldsymbol{h}_L$ based on the current weights. After obtaining this prediction, the target for the first hidden layer, denoted as $\boldsymbol{\tau}_1$, is estimated using a difference-corrected projection. Specifically, the final-layer output $\boldsymbol{h}_L$ and the ground-truth target $\boldsymbol{y}$ are independently projected through a fixed random matrix $\boldsymbol{G}$, initialized with zero mean and small standard deviation. The resulting estimate is computed as:

$$\boldsymbol{\tau}_1 = \sigma(\boldsymbol{G}\boldsymbol{y}) - \sigma(\boldsymbol{G}\boldsymbol{h}_L) + \boldsymbol{h}_1, \tag{1}$$

where $\sigma(\cdot)$ is the element-wise nonlinearity, and $\boldsymbol{h}_1$ is the activation of the first hidden layer from the first forward pass. The estimated targets then serve as inputs for the second forward pass, during which the same feedforward weights used in the first forward computation are reused to propagate the target signal and estimate layer-wise targets. Finally, the weights are updated by minimizing a local loss between activations and targets, thereby enabling layer-wise learning without backward gradients or symmetric feedback.

This forward-only update features a pipelined training by overlapping minibatches: once the first layer's weights are updated and the next layer's target is being computed, the following minibatch can begin its first



(a) BP (b) DTP (c) PEPITA (d) FTP

Figure 1: Configuration of various learning methods: a) BP, b) DTP, c) PEPITA, and d) FTP. Black arrows represent forward paths, while blue arrows indicate error/target paths. Each $\boldsymbol{W}$ denotes the learnable weight matrices of hidden layers, whereas the top-down feedback path includes **learnable** matrices ($\boldsymbol{G}_3, \boldsymbol{G}_2$ for DTP) or **randomly initialized and frozen** matrix $\boldsymbol{G}$ (for PEPITA and FTP).

forward pass. In contrast, BP requires all layers to complete backward gradient computation and weight updates before processing the next minibatch, resulting in strict update locking. FTP thus allows overlapping computation of minibatches, hereby solves update locking partially and can lead to improved wall-clock efficiency over a full training cycle than BP. For comparison, the configuration of various learning methods-BP, DTP, PEPITA, and FTP, is shown in Figure 1.

3.2 Target Estimation and Learning

In a feedforward neural network, the process begins with input data $\boldsymbol{X}$ and a corresponding output-layer target $\boldsymbol{y}$. In supervised classification $\boldsymbol{y}$ is the class label, but FTP places no such restriction on it; any target defined at the output layer is admissible. The activation values for the input layer (layer 0) are given by: $\boldsymbol{h}_0 = \boldsymbol{X}$. For each subsequent layer i (where $i = 1, 2, \ldots, L$, and L is the final layer), the activation values are computed as:

$$\boldsymbol{h}_i = \sigma(\boldsymbol{W}_i \boldsymbol{h}_{i-1}) \tag{2}$$

where $\boldsymbol{W}_i$ is the weight matrix for layer i, $\boldsymbol{h}_{i-1}$ is the activation from the previous layer, and σ is the activation function applied element-wise (e.g., *tanh*, sigmoid).
FTP assigns each layer a target via forward propagation of the initial target signal. We estimate the first target $\boldsymbol{\tau}_1$, which is the target for the first hidden layer, using a fixed random matrix $\boldsymbol{G}$, through a difference-corrected projection that contrasts the label and current output in a transformed space:

$$\boldsymbol{\tau}_1 = \sigma(\boldsymbol{G}\boldsymbol{y}) - \sigma(\boldsymbol{G}\boldsymbol{h}_L) + \boldsymbol{h}_1 \tag{3}$$

This formulation provides a top-down feedback mechanism to estimate layer-wise targets without relying on the global error gradient propagation. The target for the first hidden layer from Equation (3) reflects how its activation should shift to ultimately produce the correct output, $\boldsymbol{h}_L \approx \boldsymbol{y}$. Although $\boldsymbol{h}_1 \in \mathbb{R}^{d_1}$ and $\boldsymbol{h}_L, \boldsymbol{y} \in \mathbb{R}^{d_L}$, the fixed random projection $\boldsymbol{G}$ maps $\boldsymbol{h}_L, \boldsymbol{y}$ into $\mathbb{R}^{d_1}$. The difference $\sigma(\boldsymbol{G}\boldsymbol{y}) - \sigma(\boldsymbol{G}\boldsymbol{h}_L)$ captures the directional discrepancy between the desired output and current prediction in the projected space. Adding this correction to $\boldsymbol{h}_1$ yields a target $\boldsymbol{\tau}_1$ that nudges the first hidden layer toward a representation more likely to result in $\boldsymbol{h}_L \approx \boldsymbol{y}$ after forward propagation.

To transmit this signal to deeper layers, i.e., $i = 2, 3, \ldots, L-1$, we propagate targets recursively using the same feedforward weights used in the first forward pass:

$$\boldsymbol{\tau}_i = \sigma(\mathbf{W}_i \boldsymbol{\tau}_{i-1}), \tag{4}$$

Each layer's weights are then updated by minimizing a local loss that encourages alignment between activation and target:

$$\mathcal{L}_i = \begin{cases} \|\boldsymbol{h}_i - \text{stop_grad}(\boldsymbol{\tau}_i)\|_2^2 & \text{if } i < L \\ \mathcal{L} & \text{if } i = L, \end{cases} \tag{5}$$

where $\mathcal{L}$ is the global loss (e.g., cross-entropy, mean-squared loss) used at the final layer. The operator stop_grad$(\cdot)$ marks the target $\boldsymbol{\tau}_i$ as a constant during differentiation: although both $\boldsymbol{h}_i$ and $\boldsymbol{\tau}_i$ depend on $\boldsymbol{W}_i$, the gradient is taken only with respect to the activation. The FTP weight update that is obtained by performing standard gradient descent on the local loss $\mathcal{L}_i$, leads to:

$$\Delta \mathbf{W}_i = -\eta(\boldsymbol{h}_i - \boldsymbol{\tau}_i)\, \boldsymbol{h}_{i-1}^\top \tag{6}$$

where η is the learning rate.

This formulation demonstrates that FTP enables weight updates using only locally available information, where each update implicitly reflects a top-down influence from the global target through fixed projections. FTP also eliminates the need to compute the first hidden layer's activation from the input level in second forward pass, unlike other forward-learning algorithms that do so using modulated inputs or negative image samples (Dellaferrera & Kreiman, 2022; Hinton, 2022). Furthermore, as shown in Appendix D with more restricted conditions, this mechanism introduces the global error $\boldsymbol{e}$ through this forward target propagation, which produces updates that align closely with the BP gradients and exhibit similarity to Gauss-Newton directions. The complete training procedure is outlined in Algorithm 1.

FTP also differs from existing forward-learning methods in its degree of biological plausibility. PEPITA, for example, computes an explicit global error at the output layer and projects it back to the input using a fixed feedback matrix. A second forward pass with perturbed input induces a modified hidden activation, which is compared to the original activations to drive learning. This requires explicit error computation and assumes sensory neurons can access and respond to internal errors- an implausible assumption, as early sensory neurons act as feedforward encoders without receiving task-level feedback (Gilbert & Li, 2013). Moreover, explicit error signaling is biologically unsupported, as noted by Guerguiev et al. (Guerguiev et al., 2017), who argue for dendritic compartmentalization as the mechanism for integrating feedforward and feedback signals. In contrast, FTP generates the target for the first hidden layer by combining its feedforward activation $\boldsymbol{h}_1$ with a top-down projection of the output-layer target through $\boldsymbol{G}$, avoiding explicit error computation and input perturbation. This is consistent with cortical pyramidal neuron structure, where basal dendrites receive feedforward inputs and apical dendrites receive top-down feedback, supporting local plasticity through dendritic mismatch (Guerguiev et al., 2017). This correspondence applies to the first hidden layer, which is the only site at which FTP introduces a feedback signal. For layers $2, \ldots, L-1$, FTP requires no feedback pathway at all. The target $\boldsymbol{\tau}_i = \sigma(\boldsymbol{W}_i \boldsymbol{\tau}_{i-1})$ is carried by the same feedforward synapses that compute $\boldsymbol{h}_i$, so each neuron expresses two activity states at different times, a prediction state in the first forward pass and a target state in the second, with plasticity driven by their mismatch. Kohan et al. (2018) identify precisely this as the core mechanism shared by biologically plausible learning rules: a hidden neuron expressing its current prediction firing rate and its target firing rate at two sufficiently separated timescales, such that closing the gap between them drives the output toward the target. They further argue, as does Crick (1989), that the problematic assumption in backpropagation and its variants is not weight symmetry alone but the presupposition of systematic feedback connectivity to every neuron, for which the neocortex provides no evidence. FTP requires a single fixed top-down projection to one layer, whereas BP requires transposed weights at every layer, DTP a learned inverse at every layer, and FA a random feedback matrix at every layer. This is consistent with Kohan et al. (2022), who show that neurons lacking feedback connections can nonetheless receive a global learning signal through their feedforward connectivity.

Algorithm 1: FTP

Given: Input $\boldsymbol{X}$, output target $\boldsymbol{y}$
$\boldsymbol{h}_0 = \boldsymbol{X}$, $\boldsymbol{\tau}_L = \boldsymbol{y}$
# First Forward Pass
for $i = 1$ to L **do**
  $\boldsymbol{h}_i = \sigma(\boldsymbol{W}_i \boldsymbol{h}_{i-1})$
end for
# Estimate First Target)
$\boldsymbol{\tau}_1 = \sigma(\boldsymbol{G}\boldsymbol{\tau}_L) - \sigma(\boldsymbol{G}\boldsymbol{h}_L) + \boldsymbol{h}_1$
# Second Forward Pass
for $i = 1$ to L **do**
  if $1 < i < L$ **then**
    $\boldsymbol{\tau}_i = \sigma(\boldsymbol{W}_i \boldsymbol{\tau}_{i-1})$
  end if
  $\Delta \boldsymbol{W}_i = -\eta(\boldsymbol{h}_i - \boldsymbol{\tau}_i)\, \boldsymbol{h}_{i-1}^\top$
end for

4 Results and Discussion

4.1 Experimental Setup

We evaluated the performance of FTP against conventional BP and other biologically plausible algorithms such as PEPITA and DTP across two task categories: image classification using FC and CNN architectures, and time-series forecasting using recurrent neural networks (RNNs). FTP's performance on a task without labels, such as an autoencoder, is presented in Appendix E.4, where the objective is reconstruction rather than classification.

All models were trained using cross-entropy loss for classification tasks and mean squared error for time-series forecasting, optimized by SGD with momentum. For consistency across evaluations, each model family (FC, CNN, and RNN) was implemented using a consistent architecture across all corresponding datasets. Architectural specifications, activation functions, initialization schemes, and training schedules are provided in the Appendix E. Furthermore, we evaluate the impact of initialization choice of feedback matrix, $\boldsymbol{G}$ which is elaborated in Appendix F.

4.2 FTP for Image Classification

We evaluated FTP on FC and CNN networks for image classification. FC models were tested on MNIST, FMNIST, and CIFAR-10, while CNNs were evaluated on MNIST, CIFAR-10, and CIFAR-100. Results are summarized in Table 1. Across FC architectures, FTP consistently outperforms DTP and achieves accuracy comparable to BP and PEPITA, but with significantly lower computational cost as shown in Table 2. On CNNs, FTP matches or exceeds the bio-plausible baselines, performing on par with PEPITA and well ahead of DTP, which demonstrates its ability to learn spatial features effectively.

Table 1: Test accuracy [%] achieved by BP, DTP, PEPITA, and FTP in the experiments for fully connected (FC) networks and convolutional neural networks (CNNs).

	FC Networks			**CNNs**		
Algorithm	MNIST	FMNIST	CIFAR-10	MNIST	CIFAR-10	CIFAR-100
BP	$98.27_{\pm 0.08}$	$89.10_{\pm 0.12}$	$55.31_{\pm 0.26}$	$98.74_{\pm 0.05}$	$64.88_{\pm 0.18}$	$33.83_{\pm 0.26}$
DTP	$96.51_{\pm 0.34}$	$85.87_{\pm 0.41}$	$48.67_{\pm 0.19}$	$97.23_{\pm 0.22}$	$52.76_{\pm 0.27}$	$23.51_{\pm 0.58}$
PEPITA	$98.05_{\pm 0.11}$	$88.41_{\pm 0.14}$	$52.45_{\pm 0.28}$	$98.41_{\pm 0.24}$	$56.17_{\pm 0.62}$	$26.77_{\pm 0.87}$
FTP	$97.98_{\pm 0.25}$	$87.24_{\pm 0.21}$	$52.57_{\pm 0.37}$	$98.28_{\pm 0.35}$	$56.32_{\pm 0.82}$	$26.84_{\pm 1.13}$

Table 2: MACs (millions) for BP, DTP, PEPITA, and FTP on FC networks, along with percentage change in MAC (%) with respect to BP.

Dataset		BP	DTP	PEPITA	FTP
MNIST	MAC (mil.)	2.00	2.94	2.81	**2.02**
	MAC (%)	0.00%	47.00%	40.50%	**1.00%**
CIFAR-10	MAC (mil.)	6.69	9.97	9.86	**6.71**
	MAC (%)	0.00%	49.03%	47.38%	**0.30%**
CIFAR-100	MAC (mil.)	6.72	10.01	10.18	**6.93**
	MAC (%)	0.00%	48.96%	51.49%	**3.12%**

Table 2 also shows that FTP requires much fewer Multiply-and-Accumulate (MAC) operations per input sample than other bio-plausible methods such as PEPITA and DTP, while remaining close to BP in computational cost. This highlights FTP's practical efficiency in shallow-to-moderately deep architectures, making it a compelling forward-only alternative for both dense and convolutional models, especially in resource-constrained settings.

4.3 FTP for Sequential Tasks

We evaluated FTP on three standard time-series forecasting benchmarks, Electricity, METR-LA (Traffic), and Solar Energy (Lai et al., 2017), using a RNN trained to predict the 25th time sample from the previous

24. As in FC and CNN settings, FTP estimates targets using a difference-corrected projection through a fixed random matrix. Detailed formulation of RNNs with FTP and its update rule are presented in Appendix E.3. We evaluated model performance using two standard time-series metrics: Root Relative Squared Error (RRSE) and the Pearson correlation coefficient (CORR). These metrics are defined in Appendix G.

Table 3: Performance of RNN-based models on time-series datasets using BP, PEPITA, and FTP. ↑ (↓) indicates that higher (lower) values are better.

Method	Electricity		METR-LA		Solar	
	RRSE↓	CORR↑	RRSE↓	CORR↑	RRSE↓	CORR↑
BP (BPTT)	0.1059 ± 0.0013	0.9302 ± 0.0011	0.4680 ± 0.0083	0.8750 ± 0.0014	0.1161 ± 0.0009	0.9919 ± 0.0009
PEPITA-RNN	0.1214 ± 0.0019	0.9929 ± 0.0009	0.4419 ± 0.0065	0.8967 ± 0.0022	0.1170 ± 0.0017	0.9932 ± 0.0022
FTP-RNN	0.1219 ± 0.0010	0.9935 ± 0.0010	0.4398 ± 0.0073	0.8987 ± 0.0019	0.1177 ± 0.0011	0.9931 ± 0.0019

RNNs are inherently deep due to their temporal unrolling, and Table 3 shows that FTP consistently performs well across all datasets in RNNs. FTP has higher correlation coefficient than BP on all three benchmarks and remains competitive in RRSE. On METR-LA, FTP achieves both the lowest RRSE and highest CORR. These results suggest that FTP is particularly effective at capturing temporal structure and underlying patterns in time-series data, a crucial aspect of real-world forecasting tasks. We further evaluate longer input windows ($\{48, 96, 120\}$ timesteps) in Appendix I; FTP remains stable across all window lengths and continues to match or exceed BP in correlation. This highlights FTP's potential as a biologically plausible alternative to BPTT, with improved robustness and compatibility with analog or on-device learning environments, which is discussed in Section 4.6.

4.4 Depth Scaling of FTP

To assess the depth scaling of FTP, we evaluated both deep FC networks (all hidden layers of 1024 units) and convolutional architectures. Details on architecture choice are described in Appendix H. As evident from Table 4, across datasets, BP achieves the highest accuracy, but FTP consistently remains close to BP, while being far more stable and better than PEPITA, which collapses with depth. Prior work has shown that forward-only learning schemes without extra learnable parameters, such as FF, were restricted to at most four hidden layers (Hinton, 2022), while PEPITA was originally limited to one hidden layer (Dellaferrera & Kreiman, 2022) and required auxiliary techniques such as normalization or weight decay to stabilize deeper FC models (Srinivasan et al., 2023). Potential scalability for CNNs using FTP is discussed in Appendix H. Our results demonstrate that FTP can train both deeper FC and convolutional networks without additional learning (Gong et al., 2025) or stabilization techniques, and produces predictions directly in a single forward pass during inference, unlike FF, which requires testing all labels explicitly for each sample.

Table 4: Test accuracy (Mean ± Std) of BP, PEPITA, and FTP across datasets and hidden layer depths for FC networks. Here, L = Total number of layers in network.

Dataset	Algorithm	L = 3	L = 4	L = 5	L = 6
MNIST	BP	98.48±0.09	98.55±0.05	98.60±0.06	98.60±0.05
	PEPITA	98.19±0.03	94.63±0.21	89.12±0.35	81.71±0.80
	FTP	**98.36**±0.07	**98.15**±0.11	**98.24**±0.08	**98.20**±0.07
CIFAR-10	BP	57.83±0.08	58.06±0.19	57.86±0.39	57.65±0.37
	PEPITA	52.46±0.17	52.78±0.56	44.88±0.19	32.72±0.36
	FTP	**53.88**±0.24	**54.81**±0.18	**54.72**±0.16	**54.93**±0.13
CIFAR-100	BP	30.08±0.17	29.31±0.28	28.50±0.11	28.55±0.29
	PEPITA	24.80±0.32	18.05±0.16	1.20±0.11	6.57±0.48
	FTP	**26.27**±0.27	**27.12**±0.20	**27.85**±0.36	**28.35**±0.54

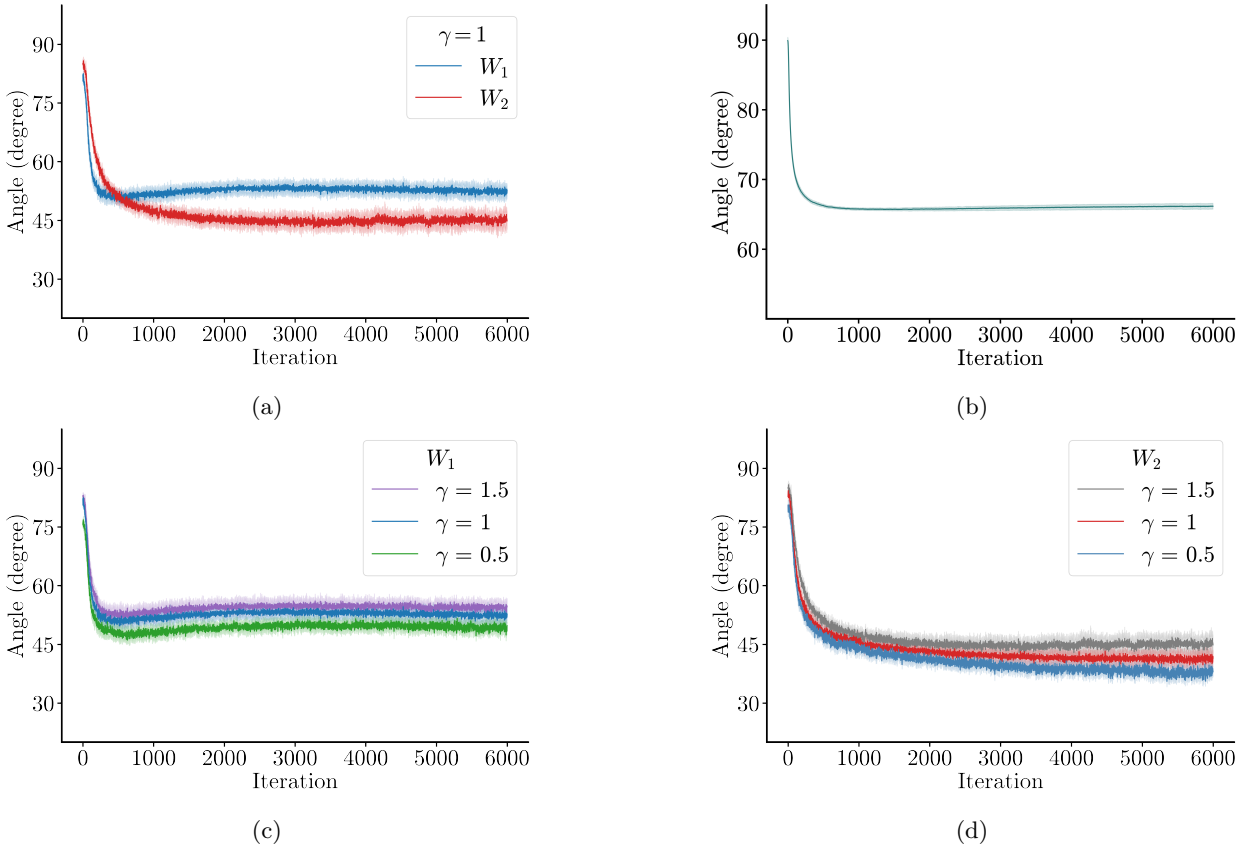


Figure 2: Evolution of alignment during training: (a) Alignment between gradient directions from FTP and BP with $\gamma = 1$; (b) Alignment between products of forward weight matrices ($\boldsymbol{W}_2^T \boldsymbol{W}_3^T$) and the feedback matrix $\boldsymbol{G}$; (c,d) Alignment of gradients for $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ under varying γ.

4.5 Alignment of FTP with Backpropagation

Gradient alignment measures how closely alternative learning rules match BP's parameter update directions. This metric is particularly useful in the context of biologically plausible algorithms (Lillicrap et al., 2016; Nøkland, 2016; Shervani-Tabar & Rosenbaum, 2023), where exact gradient computation is often replaced by approximate or learned feedback pathways. We quantify the alignment by computing the cosine angle between flattened gradient vectors of an approximate method and BP throughout training. A smaller angle indicates a closer match in update directions, which often correlates with improved learning performance and better convergence. To analyze the gradient alignment of FTP with BP, we use a fully connected network with two hidden layers, trained on MNIST for 100 epochs, and average the results over 10 random seeds. A scaling parameter γ, introduced in detail below, is set to 1 at this point, which corresponds to the standard FTP target. The alignment angle for the output layer always remains at zero during training, since FTP and BP share identical gradient expressions for the final layer. As shown in Figure 2a, the alignment angle for both hidden layers starts near 90°, indicating no initial alignment between the FTP and BP gradients. This is expected, as the fixed random feedback matrix $\boldsymbol{G}$ used in FTP is uncorrelated with the forward weights at initialization, unlike BP, which computes exact gradients via the chain rule. As training progresses, however, the angle steadily decreases, which reflects that FTP's updates increasingly align with those of BP.

To understand how the magnitude of the error signal affects alignment, we introduce a hyperparameter γ that scales the FTP target as

$$\boldsymbol{\tau}_1 = \gamma(\sigma(\boldsymbol{G}\boldsymbol{y}) - \sigma(\boldsymbol{G}\boldsymbol{h}_L)) + \boldsymbol{h}_1 \tag{7}$$

Setting $\gamma = 1$ yields the standard FTP formulation. We evaluated its effect over the range [0.1, 1.5], and here we report results for $\gamma = 0.5$ and $\gamma = 1.5$ in Figure 2c and 2d to highlight the impact of error signal scaling. Each setting was evaluated over 10 random seeds; solid lines show the mean and shaded regions denote ± 1 standard deviation. When $\gamma = 0.5$, we observe stronger alignment in both hidden layers, compared to the

case of $\gamma = 1$. Conversely, increasing γ to 1.5 degrades the alignment. These results indicate that γ acts as a regularizer, which governs error-signal strength and directional alignment with BP. When γ is large, the FTP target shift becomes overly aggressive, potentially pushing the hidden representations too far from their initial state, which can destabilize learning and lead to misaligned updates. In contrast, a smaller γ produces more conservative target shifts, allowing the network to adjust gradually in directions that are more naturally aligned with the forward pathway and ultimately closer to the BP gradients.

In addition to gradient alignment, we analyze the structural alignment between the product of forward weights (i.e., $\boldsymbol{W}_2^T \boldsymbol{W}_3^T$) and the projection matrix $\boldsymbol{G}$, following FA(Lillicrap et al., 2016). We flatten them and compute the angle between the flattened vectors throughout training, which is depicted in Figure 2b. Initially, the angle is close to 90°, reflecting a random orientation. Over time, the angle steadily decreases, indicating that FTP promotes consistent forward–backward correlations. This behavior contrasts with the anti-alignment in PEPITA (Dellaferrera & Kreiman, 2022), and supports the hypothesis (Lillicrap et al., 2016) that biologically plausible learning rules such as FTP naturally give rise to structured and coordinated representations.

4.6 Performance of FTP in Emerging Hardware and Edge Devices

BP relies on symmetry between forward and backward weights, which is difficult to maintain on emerging analog accelerators such as RRAM and other non-volatile memories, due to non-idealities including noise, programming errors, and variability. These asymmetries disrupt gradient flow and substantially degrade BP performance under low-bit precision. We demonstrate in Appendix N how the performance of BP is susceptible to such non-ideal conditions.

Programming errors, a key source of non-ideality, accumulate differently in FTP and BP. In FTP, the feedback matrix $\boldsymbol{G}$ is randomly initialized and programmed only once at the start of training. Any programming error introduced then remains embedded in $\boldsymbol{G}$ throughout the training process. Since FTP reuses the same forward matrices for both activations and layer-wise targets, any programming error in matrices affects both pathways in a consistent manner, making it inherently robust to hardware-induced asymmetry and stable even in noisy, low-precision settings. In contrast, BP requires continual reprogramming of forward and backward matrices–unless complicated bidirectional peripheral circuitry is implemented (Wan et al., 2022). Independent reprogramming of forward and backward matrices often violates the symmetry, especially in analog crossbars (e.g., RRAM, PCM), where these two are stored in physically different memory cells (Yi et al., 2023). Here, *symmetry* refers to the proportion of elements in the backward weight matrix that remain uncorrupted relative to the forward matrix. Reduced symmetry means backward weights increasingly deviate from forward-transpose, introducing new, uncorrelated errors at every step.

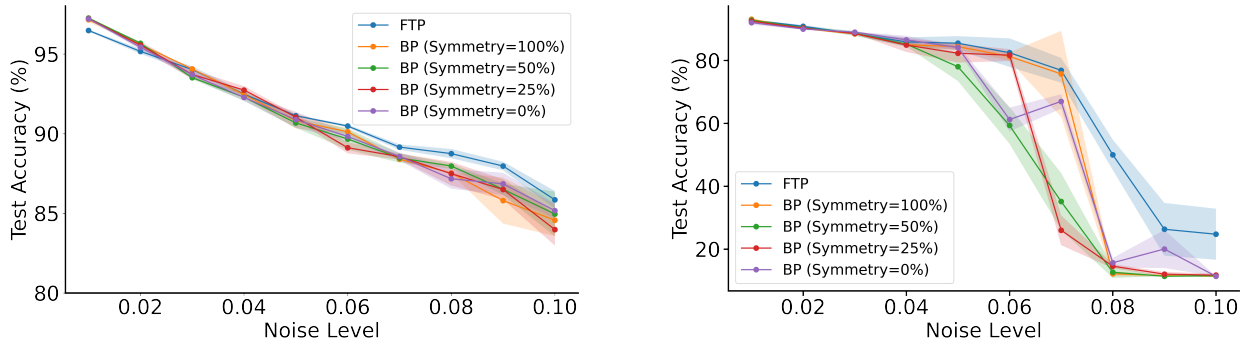


Figure 3: Performance of FTP and BP when programming errors are considered for (a) 8-bit and (b) 4-bit precision devices.

To evaluate this effect, we injected additive Gaussian noise into weights to simulate programming errors. The noise level, shown on the x-axis of Figure 3, is parameterized by α, scaling the standard deviation of noise distribution as $\text{std} = \alpha \times |w|$, where w is the weight being corrupted (Xiao et al., 2023). Thus, larger weights incur proportionally larger noise, with higher α producing stronger perturbations. We also compare against

FA, which uses fixed random feedback matrices and avoids reprogramming backward paths in hardware, making it a relevant robustness baseline (see Appendix J for detailed discussion). We also show robustness of FTP compared to other forward learning methods in Appendix K. As shown in Figure 3 and Appendix J, FTP consistently shows improved robustness under programming errors, especially as noise levels increase. Notably, in 4-bit systems, FTP maintains higher accuracy than BP at high noise levels, underscoring its robustness to programming errors in low-precision and non-ideal hardware.

4.7 Hardware-Aware Analysis of FTP's Efficiency

To assess FTP's suitability for resource-constrained environments such as TinyML applications, we estimated the number of MAC operations based on dataset sizes and epoch counts aligned with the MLCommons/Tiny Benchmark (MLCommons, 2023), following a methodology similar to Pau & Aymone (2023). The MAC counts are analytical estimates derived from network topology, dataset sizes, and epoch counts; no model in Table 5 was trained, and no accuracy is reported for it. This estimation offers a hardware-aware perspective on training efficiency under realistic deployment constraints. As shown in the table, FTP consistently achieves lower or comparable MAC counts relative to BP and substantially lower computational overhead compared to other forward-only methods such as FF, PEPITA, and DTP. We report MAC counts as a proxy for arithmetic workload rather than energy, since memory access and data movement often dominate total energy on real accelerators; end-to-end energy measurement on physical hardware is left to future work. However, FTP's data-access pattern is broadly similar to BP's: to update a given layer, both read that layer's stored activation, the incoming activation from the layer below, and the feedforward weight matrix, and the per-layer target FTP propagates forward carries the same read cost as the per-layer error BP propagates backward. Because FTP reuses these feedforward weights in their original orientation, it also avoids the transposed-weight reads and backward-matrix reprogramming that BP requires in case of analog accelerators. Thus the arithmetic savings of FTP are unlikely to be offset by added data movement. Analytically, FTP's update pattern also admits pipelined scheduling, which would reduce wall-clock training time relative to BP's update-locked schedule; we sketch this argument in Appendix M. Taken together with FTP's robustness under simulated noise, these estimates position FTP as a promising candidate for on-device learning in resource-constrained systems.

Table 5: Comparison of MAC (millions) for TinyML datasets with various algorithms and percentage change in MAC (%) with respect to BP.

Learning Method	DS-CNN/SC		MobileNet/VWW		ResNet/CIFAR10		AE/ToyADMOS	
	MAC	MAC (%)	MAC	MAC (%)	MAC	MAC (%)	MAC	MAC (%)
BP	7.70	0.00%	22.40	0.00%	37.10	0.00%	0.73	0.00%
FF	10.98	42.58%	31.52	40.70%	50.42	35.89%	1.09	49.51%
PEPITA	8.03	4.25%	22.95	2.47%	37.60	1.35%	1.23	69.10%
DTP	17.15	122.70%	50.78	126.70%	76.46	106.10%	1.49	104.00%
FTP	**7.89**	**2.51%**	**22.47**	**0.33%**	**37.46**	**0.96%**	**0.90**	**23.03%**

5 Conclusion

We have introduced Forward Target Propagation (FTP), a biologically plausible and computationally lightweight learning algorithm that serves as a forward-only alternative to backpropagation. Across fully connected, convolutional, and recurrent architectures, FTP matches or outperforms existing forward-only and target-propagation methods while remaining close to BP on dense and recurrent tasks, at a substantially lower computational cost than the other bio-plausible algorithms. A key feature of FTP is its ability to assign global credit through purely local losses, thereby eliminating the need for weight symmetry, a non-local objective, and backward signal propagation. Among recent bio-inspired and forward-only learning algorithms, our results show that FTP delivers better accuracy with more reliable depth scaling and greater computational efficiency, yielding lower multiply-and-accumulate (MAC) operations than FF, PEPITA, and DTP. FTP also exhibits resilience to low-bit precision and noisy hardware conditions, where conventional BP tends to degrade. These advantages position FTP as a prospective solution for deployment in TinyML and neuromorphic

systems. While further development is needed to fully realize its potential at scale, our findings establish FTP as a robust, computationally efficient, and biologically plausible learning framework for embedded AI systems and neuromorphic computing.

References

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Yoshua Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. *ArXiv*, abs/1407.7906, 2014. URL https://api.semanticscholar.org/CorpusID:1412698.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, January 1989. ISSN 1476-4687. doi: 10.1038/337129a0. URL http://dx.doi.org/10.1038/337129a0.

Giorgia Dellaferrera and Gabriel Kreiman. Error-driven input modulation: Solving the credit assignment problem without a backward pass. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4937–4955. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/dellaferrera22a.html.

Maxence M Ernoult, Fabrice Normandin, Abhinav Moudgil, Sean Spinney, Eugene Belilovsky, Irina Rish, Blake Richards, and Yoshua Bengio. Towards scaling difference target propagation by learning backprop targets. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5968–5987. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/ernoult22a.html.

Charlotte Frenkel, Martin Lefebvre, and David Bol. Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. *Frontiers in Neuroscience*, 15:629892, 2021. doi: 10.3389/fnins.2021.629892. URL https://www.frontiersin.org/articles/10.3389/fnins.2021.629892/full.

Charles D. Gilbert and Wu Li. Top-down influences on visual processing. *Nature Reviews Neuroscience*, 14(5): 350–363, April 2013. ISSN 1471-0048. doi: 10.1038/nrn3476. URL http://dx.doi.org/10.1038/nrn3476.

James Gong, Bruce Li, and Waleed Abdulla. Mono-forward: Backpropagation-free algorithm for efficient neural network training harnessing local errors. *ArXiv*, abs/2501.09238, 2025. URL https://api.semanticscholar.org/CorpusID:275570158.

Jordan Guerguiev, Timothy P Lillicrap, and Blake A Richards. Towards deep learning with segregated dendrites. *eLife*, 6:e22901, dec 2017. ISSN 2050-084X. doi: 10.7554/eLife.22901. URL https://doi.org/10.7554/eLife.22901.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. December 2015. doi: 10.1109/iccv.2015.123. URL http://dx.doi.org/10.1109/ICCV.2015.123.

Geoffrey E. Hinton. The forward-forward algorithm: Some preliminary investigations. *ArXiv*, abs/2212.13345, 2022. URL https://api.semanticscholar.org/CorpusID:254537921.

Chia-Hsiang Kao and Bharath Hariharan. Counter-current learning: A biologically plausible dual network approach for deep learning. *ArXiv*, abs/2409.19841, 2024. URL https://api.semanticscholar.org/CorpusID:272987991.

Adam Kohan, Edward A. Rietman, and Hava T. Siegelmann. Signal propagation: A framework for learning and inference in a forward pass. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Adam A. Kohan, Edward A. Rietman, and Hava T. Siegelmann. Error forward-propagation: Reusing feedforward connections to propagate errors in deep learning. *arXiv preprint arXiv:1808.03357*, 2018.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL https://api.semanticscholar.org/CorpusID:18268744.

Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2017. URL https://api.semanticscholar.org/CorpusID:4922476.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases*, pp. 498–515, Cham, 2015. Springer International Publishing. ISBN 978-3-319-23528-8.

Can Li, Daniel Belkin, Yunning Li, Peng Yan, Miao Hu, Ning Ge, Hao Jiang, Eric Montgomery, Peng Lin, Zhongrui Wang, Wenhao Song, John Paul Strachan, Mark Barnell, Qing Wu, R. Stanley Williams, J. Joshua Yang, and Qiangfei Xia. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature Communications*, 9(1), June 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-04484-2. URL http://dx.doi.org/10.1038/s41467-018-04484-2.

Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1):13276, Nov 2016. ISSN 2041-1723. doi: 10.1038/ncomms13276. URL https://doi.org/10.1038/ncomms13276.

Timothy P. Lillicrap, Adam Santoro, Luke Marris, Colin J. Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, Jun 2020. ISSN 1471-0048. doi: 10.1038/s41583-020-0277-3. URL https://doi.org/10.1038/s41583-020-0277-3.

Changze Lv, Jingwen Xu, Yiyang Lu, Xiaohua Wang, Zhenghua Wang, Zhibo Xu, Di Yu, Xin Du, Xiaoqing Zheng, and Xuanjing Huang. Dendritic localized learning: Toward biologically plausible algorithm, 2025. URL https://arxiv.org/abs/2501.09976.

Alexander Meulemans, Francesco Carzaniga, Johan Suykens, João Sacramento, and Benjamin F Grewe. A theoretical framework for target propagation. *Advances in Neural Information Processing Systems*, 33: 20024–20036, 2020.

Beren Millidge, Anil. K. Seth, and Christopher L. Buckley. Predictive coding: a theoretical and experimental review. *ArXiv*, abs/2107.12979, 2021. URL https://api.semanticscholar.org/CorpusID:236447532.

MLCommons. Mlcommons tiny benchmark github repository. https://github.com/mlcommons/tiny, 2023. (Date last accessed 18-February-2023).

Audun Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2016.

Alexander Ororbia and Ankur Arjun Mali. The predictive forward-forward algorithm. *ArXiv*, abs/2301.01452, 2023. URL https://api.semanticscholar.org/CorpusID:255415949.

Alexander G. Ororbia and Ankur Mali. Biologically motivated algorithms for propagating local target representations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4651–4658, July 2019. ISSN 2159-5399. doi: 10.1609/aaai.v33i01.33014651. URL http://dx.doi.org/10.1609/aaai.v33i01.33014651.

Alexander G. Ororbia, Ankur Mali, Daniel Kifer, and C. Lee Giles. Backpropagation-free deep learning with recursive local representation alignment. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37 (8):9327–9335, June 2023. ISSN 2159-5399. doi: 10.1609/aaai.v37i8.26118. URL http://dx.doi.org/10.1609/aaai.v37i8.26118.

Danilo Pietro Pau and Fabrizio Maria Aymone. Suitability of forward-forward and pepita learning to mlcommons-tiny benchmarks. In *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pp. 1–6, 2023. doi: 10.1109/COINS57856.2023.10189239.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning representations by back-propagating errors*, volume 323. Springer Science and Business Media LLC, October 1986. doi: 10.1038/323533a0. URL http://dx.doi.org/10.1038/323533a0.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

Benjamin Scellier, Maxence Ernoult, Jack Kendall, and Suhas Kumar. Energy-based learning algorithms for analog computing: A comparative study. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. URL https://openreview.net/forum?id=jey92TQfUG.

Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. Isaac: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pp. 14–26. IEEE Press, 2016. ISBN 9781467389471. doi: 10.1109/ISCA.2016.12. URL https://doi.org/10.1109/ISCA.2016.12.

Navid Shervani-Tabar and Robert Rosenbaum. Meta-learning biologically plausible plasticity rules with random feedback pathways. *Nature Communications*, 14(1), March 2023. ISSN 2041-1723. doi: 10.1038/s41467-023-37562-1. URL http://dx.doi.org/10.1038/s41467-023-37562-1.

Tatsukichi Shibuya, Nakamasa Inoue, Rei Kawakami, and Ikuro Sato. Fixed-weight difference target propagation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(8):9811–9819, June 2023. ISSN 2159-5399. doi: 10.1609/aaai.v37i8.26171. URL http://dx.doi.org/10.1609/aaai.v37i8.26171.

R. Srinivasan, Francesca Mignacco, M. Sorbaro, Maria Refinetti, Avi Cooper, Gabriel Kreiman, and Giorgia Dellaferrera. Forward learning with top-down feedback: Empirical and analytical characterization. *ArXiv*, abs/2302.05440, 2023. URL https://api.semanticscholar.org/CorpusID:256808264.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Weier Wan, Rajkumar Kubendran, Clemens Schaefer, Sukru Burc Eryilmaz, Wenqiang Zhang, Dabin Wu, Stephen Deiss, Priyanka Raina, He Qian, Bin Gao, Siddharth Joshi, Huaqiang Wu, H.-S. Philip Wong, and Gert Cauwenberghs. A compute-in-memory chip based on resistive random-access memory. *Nature*, 608(7923):504–512, Aug 2022. ISSN 1476-4687. doi: 10.1038/s41586-022-04992-8. URL https://doi.org/10.1038/s41586-022-04992-8.

James C.R. Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in Cognitive Sciences*, 23(3):235–250, March 2019. ISSN 1364-6613. doi: 10.1016/j.tics.2018.12.005. URL http://dx.doi.org/10.1016/j.tics.2018.12.005.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv*, abs/1708.07747, 2017. URL https://api.semanticscholar.org/CorpusID:702279.

T P Xiao, B Feinberg, C H Bennett, V Prabhakar, P Saxena, V Agrawal, and Marinella. On the accuracy of analog neural network inference accelerators. *IEEE Circuits and Systems Magazine*, 22(4):26–48, 2023.

Tien-Ju Yang and Vivienne Sze. Design considerations for efficient deep neural networks on processing-in-memory accelerators. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pp. 22.1.1–22.1.4, 2019. doi: 10.1109/IEDM19573.2019.8993662.

Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J. Joshua Yang, and He Qian. Fully hardware-implemented memristor convolutional neural network. *Nature*, 577(7792):641–646, Jan 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-1942-4. URL https://doi.org/10.1038/s41586-020-1942-4.

Su-in Yi, A. Alec Talin, Matthew J. Marinella, and R. Stanley Williams. Physical compact model for three-terminal sonos synaptic circuit element. *Advanced Intelligent Systems*, 4(9):2200070, 2022. doi: https://doi.org/10.1002/aisy.202200070. URL https://advanced.onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202200070.

Su-in Yi, Jack D. Kendall, R. Stanley Williams, and Suhas Kumar. Activity-difference training of deep neural networks using memristor crossbars. *Nature Electronics*, 6(1):45–51, Jan 2023. ISSN 2520-1131. doi: 10.1038/s41928-022-00869-w. URL https://doi.org/10.1038/s41928-022-00869-w.

Mohammed A. Zidan, John Paul Strachan, and Wei D. Lu. The future of electronics based on memristive systems. *Nature Electronics*, 1(1):22–29, Jan 2018. ISSN 2520-1131. doi: 10.1038/s41928-017-0006-8. URL https://doi.org/10.1038/s41928-017-0006-8.

A Limitations and Future Work

FTP is designed for supervised tasks, where a target is defined at the output layer; it is not a self-supervised rule and does not construct its own objective, though it applies without modification to any label-free objective that supplies such a target, such as reconstruction, which is a natural extension to explore (demonstrated in Appendix E.4). Within this setting, several limitations bound our claims. Our evaluation covers small- to moderate-scale architectures, with depth verified from three to six layers (Appendix H); we therefore do not claim scalability to deep standard or residual networks, on which no forward-only rule has yet been demonstrated (Dellaferrera & Kreiman, 2022; Srinivasan et al., 2023; Kao & Hariharan, 2024; Lv et al., 2025; Hinton, 2022), and a full-precision gap to BP remains on convolutional neural networks. Extending FTP to deeper models, larger datasets, and attention-based architectures, where projecting the output target through the attention mechanism is the open problem, with small-scale language modelling against BP-trained baselines as the appropriate first test, is left to future work. In the recurrent setting, FTP reduces but does not eliminate the storage of intermediate temporal activations, retaining a per-timestep eligibility trace while removing BPTT's reverse-time traversal and weight transport. Our efficiency and robustness evidence rests on Multiply-and-Accumulate (MAC) counts and simulated low-precision noise rather than physical hardware, so we treat energy and runtime benefits as prospective; end-to-end measurement on instrumented hardware remains future work. Finally, our alignment analysis is theoretically limited to a single-sample linear setting, which we discuss further in Appendix D.

B Broader Impact of Our Work

Energy consumption is an increasingly central concern in machine learning, particularly for edge and on-device deployment where power budgets are tight. Our work proposes Forward Target Propagation (FTP), a biologically inspired, forward-only learning algorithm that offers significant advantages in robustness and hardware compatibility compared to standard backpropagation (BP). These characteristics make FTP particularly relevant for enabling sustainable training and deployment of large-scale models in both cloud and edge settings. Below, we outline key broader impacts of our approach:

Compatibility with Quantization and Noisy Hardware: Emerging memory technologies such as Resistive Random Access Memory (RRAM) and Phase-Change Memory (PCM) offer compact, analog, and energy-efficient platforms for neural computation, but suffer from non-idealities such as stochastic noise, variability and limited precision. Unlike BP, which requires high-precision symmetric weight updates, FTP is inherently robust to such hardware imperfections due to its forward-only and fixed-feedback structure. Our simulations indicate that FTP retains its performance better than BP under low-bit precision and noisy conditions, making it particularly well suited for deployment on in-memory computing systems.

Enabling Sustainable, On-Device Intelligence: By aligning algorithmic design with the constraints and strengths of emerging and edge hardware, FTP opens the door to efficient on-device learning in applications such as autonomous sensors, wearable devices, and edge robotics. We stress that these are potential benefits rather than demonstrated ones: our experiments establish reduced MAC counts and robustness to simulated noise, not a measured reduction in energy consumption. Confirming an energy or environmental benefit would require deployment on physical hardware with power instrumentation, which we identify as an important direction for future work.

C Biological Properties of FTP

Table 6 compares various learning rules across key criteria relevant to biological plausibility. A checkmark (✓) indicates that the corresponding issue is addressed (i.e., the property is satisfied), while a cross (✗) indicates that the issue remains unsolved. The tilde symbol (∼) denotes a partial solution. Notably, FTP satisfies all the listed criteria, with a partial resolution of update locking.

Table 6: Comparison of learning rules based on key metrics.

Learning Rule	Forward Only	Weight Symmetry	Local	Activity Freezing	Update Unlocked
BP	✗	✗	✗	✗	✗
DTP	✗	✓	✓	✗	✗
PEPITA	✓	✓	✓	✓	∼
FF	✓	✓	✓	✓	✓
FTP	✓	✓	✓	✓	∼

D Analysis of FTP in Network with Two Hidden Layers

We consider a neural network with two hidden layers with activation $\sigma(\cdot)$. In Section D.1, we proved that (under some conditions) the weight update direction for the hidden layers in FTP is within 90° of that of BP. With further restricted assumptions, we showed in Section D.2, that the update direction for the hidden neurons in FTP is aligned with the Gauss-Newton direction. We followed the structure of the proofs from (Lillicrap et al., 2016).

For an input $\boldsymbol{x} \in \mathbb{R}^{d_x}$, we have

$$\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_1\boldsymbol{x}) \tag{8}$$
$$\boldsymbol{h}_2 = \sigma(\boldsymbol{W}_2\boldsymbol{h}_1) \tag{9}$$
$$\boldsymbol{h}_3 = \sigma(\boldsymbol{W}_3\boldsymbol{h}_2) \tag{10}$$

where $\boldsymbol{h}_3 \in \mathbb{R}^{d_y}$ is the output, and $\boldsymbol{h}_1 \in \mathbb{R}^{d_1}, \boldsymbol{h}_2 \in \mathbb{R}^{d_2}$ are hidden neurons. In forward target propagation (FTP), the losses to train the weight matrices $\boldsymbol{W}_1, \boldsymbol{W}_2$ and $\boldsymbol{W}_3$ are as follows:

$$\mathcal{L}_1 = \frac{1}{2}||\,\text{stop_grad}(\boldsymbol{\tau}_1) - \boldsymbol{h}_1||^2 \tag{11}$$
$$\mathcal{L}_2 = \frac{1}{2}||\,\text{stop_grad}(\boldsymbol{\tau}_2) - \boldsymbol{h}_2||^2 \tag{12}$$
$$\mathcal{L}_3 = \frac{1}{2}||\boldsymbol{y} - \boldsymbol{h}_3||^2 \tag{13}$$

where $\boldsymbol{y} \in \mathbb{R}^{d_y}$ is the target value at the output layer and $\boldsymbol{\tau}_1, \boldsymbol{\tau}_2$ are target values (suggested by FTP) at the hidden layers according to the following.

$$\boldsymbol{\tau}_1 = \boldsymbol{h}_1 + \sigma(\boldsymbol{G}\boldsymbol{y}) - \sigma(\boldsymbol{G}\boldsymbol{h}_3) \tag{14}$$
$$\boldsymbol{\tau}_2 = \sigma(\boldsymbol{W}_2\boldsymbol{\tau}_1) \tag{15}$$

$\boldsymbol{G}$ is a $d_1 \times d_y$ projection matrix where $G_{i,j}$ can be distributed according to $\mathcal{N}(0,1)$. Because $\boldsymbol{\tau}_1$ and $\boldsymbol{\tau}_2$ are detached via stop_grad, every gradient below is taken only with respect to the activation $\boldsymbol{h}_i$, even though both $\boldsymbol{h}_i$ and $\boldsymbol{\tau}_i$ depend on $\boldsymbol{W}_i$; consequently $\frac{1}{2}\|\text{stop_grad}(\boldsymbol{\tau}_i) - \boldsymbol{h}_i\|^2$ and $\frac{1}{2}\|\boldsymbol{h}_i - \text{stop_grad}(\boldsymbol{\tau}_i)\|^2$ yield the same local update.

Note that we considered mean squared error as the loss $\mathcal{L}_3$ at the output layer for our theoretical analysis. Furthermore, the other assumptions, e.g., a single sample with linear networks, and specific initialization of the weight matrices, are considerably more restrictive than the conditions of our empirical experiments. Hence, the analysis presented here does not provide a complete theoretical evidence for our empirical results across multiple models and datasets.

D.1 Alignment in Weight Update Direction Between FTP and BP

In this section, we consider the same neural network described above, except for linear activation functions.

Denoting error signal as $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{h}_3$, we have the following gradient direction for each weight matrix.

$$\frac{\partial \mathcal{L}_3}{\partial \boldsymbol{W}_3} = -\boldsymbol{e}\boldsymbol{h}_2^T \tag{16}$$

$$\frac{\partial \mathcal{L}_2}{\partial \boldsymbol{W}_2} = -(\boldsymbol{\tau}_2 - \boldsymbol{h}_2)\boldsymbol{h}_1^T \tag{17}$$

$$\frac{\partial \mathcal{L}_1}{\partial \boldsymbol{W}_1} = -(\boldsymbol{\tau}_1 - \boldsymbol{h}_1)\boldsymbol{x}^T \tag{18}$$

Note the FTP has the same gradient for $\boldsymbol{W}_3$ as in BP. From Equations (14) and (15), we have

$$\boldsymbol{\tau}_1 - \boldsymbol{h}_1 = \boldsymbol{G}\boldsymbol{y} - \boldsymbol{G}\boldsymbol{h}_3 = \boldsymbol{G}\boldsymbol{e}$$
$$\boldsymbol{\tau}_2 - \boldsymbol{h}_2 = \boldsymbol{W}_2\boldsymbol{\tau}_1 - \boldsymbol{W}_2\boldsymbol{h}_1 = \boldsymbol{W}_2(\boldsymbol{\tau}_1 - \boldsymbol{h}_1) = \boldsymbol{W}_2\boldsymbol{G}\boldsymbol{e}$$

Finally, we get the gradient direction for $\boldsymbol{W}_1, \boldsymbol{W}_2$ under FTP as:

$$\frac{\partial \mathcal{L}_2}{\partial \boldsymbol{W}_2} = -\boldsymbol{W}_2\boldsymbol{G}\boldsymbol{e}\boldsymbol{h}_1^T \tag{19}$$

$$\frac{\partial \mathcal{L}_1}{\partial \boldsymbol{W}_1} = -\boldsymbol{G}\boldsymbol{e}\boldsymbol{x}^T \tag{20}$$

For BP,

$$\frac{\partial \mathcal{L}_2}{\partial \boldsymbol{W}_2} = -\boldsymbol{W}_3^T\boldsymbol{e}\boldsymbol{h}_1^T \tag{21}$$

$$\frac{\partial \mathcal{L}_1}{\partial \boldsymbol{W}_1} = -\boldsymbol{W}_2^T\boldsymbol{W}_3^T\boldsymbol{e}\boldsymbol{x}^T \tag{22}$$

We hypothesize the following alignment between FTP and BP's gradient direction for any non-zero $\boldsymbol{e}$:

$$\langle \boldsymbol{G}\boldsymbol{e}, \boldsymbol{W}_2^T\boldsymbol{W}_3^T\boldsymbol{e}\rangle > 0 \tag{23}$$

$$\langle \boldsymbol{W}_2\boldsymbol{G}\boldsymbol{e}, \boldsymbol{W}_3^T\boldsymbol{e}\rangle > 0 \tag{24}$$

Lemma 1. *If we initialize $\boldsymbol{W}_1$ and $\boldsymbol{W}_3$ with zero entries and $\boldsymbol{W}_2$ being initialized with $\boldsymbol{A}$, then there exist some scalars $s_1, s_{\boldsymbol{W}_1}, s_{\boldsymbol{W}_2}, s_{\boldsymbol{W}_3}$ at every time step (during training using FTP) such that,*

$$\boldsymbol{h}_1 = s_1\boldsymbol{G}\boldsymbol{y} \tag{25}$$

$$\boldsymbol{W}_1 = s_{\boldsymbol{W}_1}\boldsymbol{G}\boldsymbol{y}\boldsymbol{x}^T \tag{26}$$

$$\boldsymbol{W}_2 = \boldsymbol{A}(\boldsymbol{I} + s_{\boldsymbol{W}_2}\boldsymbol{G}\boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T) \tag{27}$$

$$\boldsymbol{W}_3 = s_{\boldsymbol{W}_3}\boldsymbol{y}(\boldsymbol{A}\boldsymbol{G}\boldsymbol{y})^T \tag{28}$$

Proof. Here, we provide the proof by induction. At initialization ($t = 0$) Equations (25) to (28) satisfies with $s_1 = s_{\boldsymbol{W}_1} = s_{\boldsymbol{W}_2} = s_{\boldsymbol{W}_3} = 0$. Now, if this is true for any $t > 0$, then we need to prove this holds for $t + 1$.

$$\boldsymbol{h}_3^{(t)} = \boldsymbol{W}_3^{(t)}\boldsymbol{h}_2^{(t)} \tag{29}$$

$$= s_1^{(t)}s_{\boldsymbol{W}_3}^{(t)}\boldsymbol{y}(\boldsymbol{A}\boldsymbol{G}\boldsymbol{y})^T\boldsymbol{W}_2^{(t)}\boldsymbol{G}\boldsymbol{y} \tag{30}$$

$$= s_1^{(t)}s_{\boldsymbol{W}_3}^{(t)}\boldsymbol{y}(\boldsymbol{A}\boldsymbol{G}\boldsymbol{y})^T\boldsymbol{A}(\boldsymbol{I} + s_{\boldsymbol{W}_2}^{(t)}\boldsymbol{G}\boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T)\boldsymbol{G}\boldsymbol{y} \tag{31}$$

$$
\begin{aligned}
(\boldsymbol{AGy})^T\boldsymbol{A}(\boldsymbol{I}+s^{(t)}_{\boldsymbol{W}_2}\boldsymbol{Gy}(\boldsymbol{Gy})^T)\boldsymbol{Gy} &= (\boldsymbol{Gy})^T\boldsymbol{A}^T\boldsymbol{A}(\boldsymbol{I}+s^{(t)}_{\boldsymbol{W}_2}\boldsymbol{Gy}(\boldsymbol{Gy})^T)\boldsymbol{Gy} \\
&= ||\boldsymbol{AGy}||^2 + s^{(t)}_{\boldsymbol{W}_2}||\boldsymbol{AGy}||^2||\boldsymbol{Gy}||^2
\end{aligned}
$$

Now,

$$
\boldsymbol{h}_3^{(t)} = s_1^{(t)}s^{(t)}_{\boldsymbol{W}_3}\left(||\boldsymbol{AGy}||^2 + s^{(t)}_{\boldsymbol{W}_2}||\boldsymbol{AGy}||^2||\boldsymbol{Gy}||^2\right)\boldsymbol{y} \tag{32}
$$

$$
= s_3^{(t)}\boldsymbol{y} \quad (\text{denoting } s_3^{(t)} = s_1^{(t)}s^{(t)}_{\boldsymbol{W}_3}\left(||\boldsymbol{AGy}||^2 + s^{(t)}_{\boldsymbol{W}_2}||\boldsymbol{AGy}||^2||\boldsymbol{Gy}||^2\right)) \tag{33}
$$

$$
\boldsymbol{e}^{(t)} = \boldsymbol{y} - \boldsymbol{h}_3^{(t)} = (1-s_3^{(t)})\boldsymbol{y} \tag{34}
$$

$$
\begin{aligned}
\boldsymbol{W}_1^{(t+1)} &= \boldsymbol{W}_1^{(t)} + \eta_1\boldsymbol{Ge}^{(t)}\boldsymbol{x}^T \\
&= s^{(t)}_{\boldsymbol{W}_1}\boldsymbol{Gyx}^T + \eta_1\boldsymbol{G}(1-s_3^{(t)})\boldsymbol{yx}^T \\
&= s^{(t+1)}_{\boldsymbol{W}_1}\boldsymbol{Gyx}^T \quad \left(\text{where } s^{(t+1)}_{\boldsymbol{W}_1} = s^{(t)}_{\boldsymbol{W}_1} + \eta_1(1-s_3^{(t)})\right)
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{W}_2^{(t+1)} &= \boldsymbol{W}_2^{(t)} + \eta_2\boldsymbol{W}_2^{(t)}\boldsymbol{Ge}^{(t)}\boldsymbol{h}_1^{T(t)} \\
&= \boldsymbol{A} + s^{(t)}_{\boldsymbol{W}_2}\boldsymbol{AGy}(\boldsymbol{Gy})^T + \eta_2\left(\boldsymbol{A} + s^{(t)}_{\boldsymbol{W}_2}\boldsymbol{AGy}(\boldsymbol{Gy})^T\right)\boldsymbol{G}(1-s_3^{(t)})\boldsymbol{y}(s_1^{(t)}\boldsymbol{Gy})^T \\
&= \boldsymbol{A} + (s^{(t)}_{\boldsymbol{W}_2} + \eta_2 s_1^{(t)}(1-s_3^{(t)}))\boldsymbol{AGy}(\boldsymbol{Gy})^T + \eta_2 s_1^{(t)}s^{(t)}_{\boldsymbol{W}_2}(1-s_3^{(t)})\boldsymbol{AGy}(\boldsymbol{Gy})^T\boldsymbol{Gy}(\boldsymbol{Gy})^T \\
&= \boldsymbol{A} + (s^{(t)}_{\boldsymbol{W}_2} + \eta_2 s_1^{(t)}(1-s_3^{(t)}))\boldsymbol{AGy}(\boldsymbol{Gy})^T + \eta_2 s_1^{(t)}s^{(t)}_{\boldsymbol{W}_2}(1-s_3^{(t)})\boldsymbol{AGy}||\boldsymbol{Gy}||^2(\boldsymbol{Gy})^T \\
&= \boldsymbol{A} + s^{(t+1)}_{\boldsymbol{W}_2}\boldsymbol{AGy}(\boldsymbol{Gy})^T
\end{aligned}
$$

Here, $s^{(t+1)}_{\boldsymbol{W}_2} = s^{(t)}_{\boldsymbol{W}_2} + \eta_2 s_1^{(t)}(1-s_3^{(t)}) + \eta_2 s_1^{(t)}s^{(t)}_{\boldsymbol{W}_2}(1-s_3^{(t)})||\boldsymbol{Gy}||^2$

$$
\begin{aligned}
\boldsymbol{W}_3^{(t+1)} &= \boldsymbol{W}_3^{(t)} + \eta_3\boldsymbol{e}^{(t)}\boldsymbol{h}_2^{T(t)} \\
&= s^{(t)}_{\boldsymbol{W}_3}\boldsymbol{y}(\boldsymbol{AGy})^T + \eta_3 s_1^{(t)}(1-s_3^{(t)})\boldsymbol{y}(\boldsymbol{W}_2^{(t)}\boldsymbol{Gy})^T \\
&= s^{(t)}_{\boldsymbol{W}_3}\boldsymbol{y}(\boldsymbol{AGy})^T + \eta_3 s_1^{(t)}(1-s_3^{(t)})\boldsymbol{y}(\boldsymbol{A}(\boldsymbol{I}+s^{(t)}_{\boldsymbol{W}_2}\boldsymbol{Gy}(\boldsymbol{Gy})^T)\boldsymbol{Gy})^T \\
&= s^{(t)}_{\boldsymbol{W}_3}\boldsymbol{y}(\boldsymbol{AGy})^T + \eta_3 s_1^{(t)}(1-s_3^{(t)})\left(1+s^{(t)}_{\boldsymbol{W}_2}||\boldsymbol{Gy}||^2\right)\boldsymbol{y}(\boldsymbol{AGy})^T \\
&= s^{(t+1)}_{\boldsymbol{W}_3}\boldsymbol{y}(\boldsymbol{AGy})^T \quad (\text{where } s^{(t+1)}_{\boldsymbol{W}_3} = s^{(t)}_{\boldsymbol{W}_3} + \eta_3 s_1^{(t)}(1-s_3^{(t)})\left(1+s^{(t)}_{\boldsymbol{W}_2}||\boldsymbol{Gy}||^2\right))
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{h}_1^{(t+1)} &= \boldsymbol{W}_1^{(t+1)}\boldsymbol{x} \\
&= \left(\boldsymbol{W}_1^{(t)} + \eta_1\boldsymbol{Ge}^{(t)}\boldsymbol{x}^T\right)\boldsymbol{x} \\
&= \boldsymbol{h}_1^{(t)} + \eta_1(1-s_3^{(t)})\boldsymbol{Gy}||\boldsymbol{x}||^2 \\
&= s_1^{(t)}\boldsymbol{Gy} + \eta_1(1-s_3^{(t)})||\boldsymbol{x}||^2\boldsymbol{Gy} \\
&= s_1^{(t+1)}\boldsymbol{Gy} \quad (\text{where } s_1^{(t+1)} = s_1^{(t)} + \eta_1(1-s_3^{(t)})||\boldsymbol{x}||^2)
\end{aligned}
$$

□

Remark 1. *Initializing the scalars $s_1 = s_{\boldsymbol{W}_1} = s_{\boldsymbol{W}_2} = s_{\boldsymbol{W}_3} = 0$ at $t = 0$, they stays positive from $t = 2$ by appropriately selecting learning rates $\eta_i > 0$ such that $s_3^{(t)} \leq 1$.*

Theorem 1. *Under the same conditions in Lemma 1, the FTP and BP's (non-zero) gradient direction for $\boldsymbol{W}_1, \boldsymbol{W}_2$ are within $90°$ of each other, i.e.*

$$\langle \boldsymbol{G}\boldsymbol{e}, \boldsymbol{W}_2^T \boldsymbol{W}_3^T \boldsymbol{e} \rangle > 0 \tag{35}$$

$$\langle \boldsymbol{W}_2 \boldsymbol{G}\boldsymbol{e}, \boldsymbol{W}_3^T \boldsymbol{e} \rangle > 0 \tag{36}$$

Proof. Note $\langle \boldsymbol{G}\boldsymbol{e}, \boldsymbol{W}_2^T \boldsymbol{W}_3^T \boldsymbol{e} \rangle = \langle \boldsymbol{W}_2 \boldsymbol{G}\boldsymbol{e}, \boldsymbol{W}_3^T \boldsymbol{e} \rangle$

From Equations (27) and (28) of Lemma 1, we have

$$\begin{aligned}
\boldsymbol{W}_3 \boldsymbol{W}_2 &= s_{\boldsymbol{W}_3} \boldsymbol{y}(\boldsymbol{A}\boldsymbol{G}\boldsymbol{y})^T \boldsymbol{A}(\boldsymbol{I} + s_{\boldsymbol{W}_2} \boldsymbol{G}\boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T) \\
&= s_{\boldsymbol{W}_3} \boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T \boldsymbol{A}^T \boldsymbol{A}(\boldsymbol{I} + s_{\boldsymbol{W}_2} \boldsymbol{G}\boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T) \\
&= s_{\boldsymbol{W}_3} \boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T \boldsymbol{A}^T \boldsymbol{A} + s_{\boldsymbol{W}_3} s_{\boldsymbol{W}_2} ||\boldsymbol{A}\boldsymbol{G}\boldsymbol{y}||^2 \boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T \\
&= s_{\boldsymbol{W}_3} \boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T \boldsymbol{A}^T \boldsymbol{A} + s_{3,2} \boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T \quad (\text{denoting } s_{3,2} = s_{\boldsymbol{W}_3} s_{\boldsymbol{W}_2} ||\boldsymbol{A}\boldsymbol{G}\boldsymbol{y}||^2)
\end{aligned} \tag{37}$$

Now for the weight matrix of the first hidden layer, $\boldsymbol{W}_1$

$$\begin{aligned}
\langle \boldsymbol{G}\boldsymbol{e}, \boldsymbol{W}_2^T \boldsymbol{W}_3^T \boldsymbol{e} \rangle &= (\boldsymbol{G}\boldsymbol{e})^T (\boldsymbol{W}_3 \boldsymbol{W}_2)^T \boldsymbol{e} \\
&= s_{\boldsymbol{W}_3} (1 - s_3)^2 (\boldsymbol{G}\boldsymbol{y})^T (\boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T \boldsymbol{A}^T \boldsymbol{A})^T \boldsymbol{y} + s_{3,2} (1 - s_3)^2 (\boldsymbol{G}\boldsymbol{y})^T (\boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T)^T \boldsymbol{y} \\
&= s_{\boldsymbol{W}_3} (1 - s_3)^2 ||\boldsymbol{A}\boldsymbol{G}\boldsymbol{y}||^2 ||\boldsymbol{y}||^2 + s_{3,2} (1 - s_3)^2 ||\boldsymbol{G}\boldsymbol{y}||^2 ||\boldsymbol{y}||^2 > 0 \quad \forall \boldsymbol{y} \neq \boldsymbol{0}
\end{aligned}$$

The last inequality follows from $s_{\boldsymbol{W}_2}, s_{\boldsymbol{W}_3}$ being positive scalars (Remark 1). □

D.2 Relation with Gauss-Newton Update for Hidden Neurons

In linear neural network with two hidden layers, the feedback signal (under FTP) for update in $\boldsymbol{h}_1$ is $\boldsymbol{G}\boldsymbol{e}$, and we postulate that it is aligned with the Gauss-Newton update signal, $(\boldsymbol{W}_3 \boldsymbol{W}_2)^\dagger$. If we can show there exists a relation such as $s\boldsymbol{G}\boldsymbol{e} = (\boldsymbol{W}_3 \boldsymbol{W}_2)^\dagger \boldsymbol{e}$ with a positive scalar s, then we can conclude update direction for the hidden neurons $\boldsymbol{h}_1$ is aligned with the Gauss-Newton direction.

Theorem 2. *Under the same conditions in Lemma 1 and $\boldsymbol{W}_2$ being initialized with $\boldsymbol{A}$ such that $\boldsymbol{A}^T \boldsymbol{A} = \boldsymbol{I}$ (requires $d_2 \geq d_1$), there exists a positive scaler s such that*

$$s\boldsymbol{G}\boldsymbol{e} = (\boldsymbol{W}_3 \boldsymbol{W}_2)^\dagger \boldsymbol{e} \tag{38}$$

Proof. From Equation (37) with $\boldsymbol{A}^T \boldsymbol{A} = \boldsymbol{I}$,

$$\begin{aligned}
\boldsymbol{W}_3 \boldsymbol{W}_2 &= s_{\boldsymbol{W}_3} \boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T + s_{\boldsymbol{W}_3} s_{\boldsymbol{W}_2} ||\boldsymbol{G}\boldsymbol{y}||^2 \boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T \\
&= s'_{3,2} \boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T \quad (\text{where } s'_{3,2} = s_{\boldsymbol{W}_3} + s_{\boldsymbol{W}_3} s_{\boldsymbol{W}_2} ||\boldsymbol{G}\boldsymbol{y}||^2)
\end{aligned}$$

Showing $s\boldsymbol{G}\boldsymbol{e} = (\boldsymbol{W}_3 \boldsymbol{W}_2)^\dagger \boldsymbol{e}$ is equivalent to $s\boldsymbol{G}\boldsymbol{y} = (\boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T)^\dagger \boldsymbol{y}$ since we have $\boldsymbol{e} = (1 - s_3)\boldsymbol{y}$ from Equation (34).

$$
\begin{aligned}
(\boldsymbol{y}(\boldsymbol{G}\boldsymbol{y})^T)^\dagger \boldsymbol{y} &= (\boldsymbol{y}^T\boldsymbol{G}^T)^\dagger \boldsymbol{y}^\dagger \boldsymbol{y} \\
&= (\boldsymbol{G}^T)^\dagger (\boldsymbol{y}^T)^\dagger \boldsymbol{y}^\dagger \boldsymbol{y} \\
&= (\boldsymbol{G}^T)^\dagger (\boldsymbol{y}^T)^\dagger \boldsymbol{y}^T (\boldsymbol{y}^T)^\dagger \\
&= (\boldsymbol{G}^T)^\dagger (\boldsymbol{y}^T)^\dagger \\
&= ((\boldsymbol{G}\boldsymbol{y})^T)^\dagger \\
&= \boldsymbol{G}\boldsymbol{y}((\boldsymbol{G}\boldsymbol{y})^T\boldsymbol{G}\boldsymbol{y})^{-1} \\
&= ||\boldsymbol{G}\boldsymbol{y}||^{-2}\boldsymbol{G}\boldsymbol{y} \\
&= s\boldsymbol{G}\boldsymbol{y}
\end{aligned}
$$

□

Remark 2. *When $d_2 \geq d_1$, we can construct $\boldsymbol{A}$ by selecting d_1 orthonormal column vectors such that $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$. This adds further restriction to the initialization scheme mentioned in Lemma 1.*

E Model Architecture and Implementation Details

We evaluated the performance of FTP, BP, DTP, and PEPITA across three model families–fully connected networks (FC), convolutional neural networks (CNN), and recurrent neural networks (RNN)–on image classification and multivariate time-series forecasting tasks. While Algorithm 1 and Eqs. (1)–(4) are presented using FC networks for clarity, the underlying principles of FTP apply equally to CNNs and RNNs with only minor modifications. The core learning algorithm and update equations remain unchanged across architectures; the only adjustment required is reshaping the feedback projection to match the dimensionality of the corresponding hidden activations. For example, in CNNs, a feedback matrix $G \in \mathbb{R}^{C\times(C_1\cdot H_1\cdot W_1)}$ projects the output predictions or labels $y \in \mathbb{R}^{B\times C}$ into a correction matrix of shape $(B, C_1 \cdot H_1 \cdot W_1)$. This is then reshaped into a tensor of shape (B, C_1, H_1, W_1), matching the spatial dimensions of the first hidden layer activations. Subsequent targets are propagated forward as in the first forward pass, and weight updates follow the same principles as in the FC case.

All models were trained using stochastic gradient descent (SGD) with momentum 0.9, batch size 64, and cross-entropy loss. Unless otherwise stated, all hidden layers used the *tanh* activation function, and all parameters (including the projection matrix $\boldsymbol{G}$ in FTP) were initialized using He initialization (He et al., 2015).

E.1 Fully Connected Networks (FC)

We evaluated FC networks on MNIST (Lecun et al., 1998), Fashion-MNIST (FMNIST) (Xiao et al., 2017), and CIFAR-10 (Krizhevsky, 2009). For MNIST and FMNIST, input images were flattened to 784 dimensions; for CIFAR datasets, to 3072 dimensions. The architecture consisted of two hidden layers with 1024 and 128 neurons, followed by a softmax output layer with 10 units for MNIST/FMNIST/CIFAR-10. A dropout rate of 0.1 was applied after each hidden layer. All models were trained for 100 epochs, with the learning rate decayed by a factor of 10 at epochs 60 and 90.

E.2 Convolutional Neural Networks (CNN)

We applied CNNs to the same four image classification datasets: MNIST (Lecun et al., 1998), CIFAR-10, and CIFAR-100 (Krizhevsky, 2009). The CNN architecture consisted of a 2D convolutional layer with 32 output channels and a 5×5 kernel, followed by a 2×2 max pooling layer. The output feature maps were flattened and passed to a softmax output layer with 10 or 100 units, depending on the dataset. All CNN models used *tanh* activations except the final layer where softmax activation function was used. The models were trained for 100 epochs, and followed the same learning rate schedule as the FC models (decayed at epochs 60 and 90).

E.3 Recurrent Neural Networks (RNN)

For a single recurrent layer with input weights $\boldsymbol{W}_{xh}$, recurrent weights $\boldsymbol{W}_{hh}$, and readout $\boldsymbol{W}_{hy}$, the first forward pass over an input window $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ is:

$$\boldsymbol{h}_t = \sigma_h(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h), \qquad \hat{\boldsymbol{y}} = \boldsymbol{W}_{hy}\boldsymbol{h}_T + \boldsymbol{b}_y, \tag{39}$$

with $\boldsymbol{h}_0 = \boldsymbol{0}$ and σ_h the activation function.

FTP forms the target sequence by re-running the same recurrence with a single modification: the initial hidden state is set to the difference-corrected projection of the output error through a fixed random matrix $\boldsymbol{G} \in \mathbb{R}^{d_h \times d_{\text{out}}}$,

$$\boldsymbol{\tau}_0 = \sigma(\boldsymbol{G}\boldsymbol{y}) - \sigma(\boldsymbol{G}\hat{\boldsymbol{y}}), \tag{40}$$

after which targets propagate forward through the same recurrent weights used in the first pass:

$$\boldsymbol{\tau}_t = \sigma_h(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{\tau}_{t-1} + \boldsymbol{b}_h), \qquad t = 1, \ldots, T. \tag{41}$$

The two passes are therefore identical apart from their initial condition: the first begins from $\boldsymbol{h}_0 = \boldsymbol{0}$, the second from $\boldsymbol{\tau}_0$. The error enters the computation exactly once and is carried forward by the recurrence, in direct analogy to $\boldsymbol{\tau}_i = \sigma(\boldsymbol{W}_i\boldsymbol{\tau}_{i-1})$ in the feedforward case (Section 3.2). Because $\boldsymbol{W}_{xh}$ and $\boldsymbol{W}_{hh}$ are shared across timesteps, the unrolled network is a weight-tied layer stack, and target propagation through it is the same recursion applied to that stack. No error is injected at any subsequent timestep, and no feedback pathway is required beyond the single projection $\boldsymbol{G}$.

Each timestep is trained by a local loss between its activation and its target, and the readout by the task loss:

$$\mathcal{L}_t = \tfrac{1}{2}\left\|\boldsymbol{h}_t - \texttt{stop_grad}(\boldsymbol{\tau}_t)\right\|_2^2, \qquad \mathcal{L}_{\text{out}} = \tfrac{1}{2}\left\|\hat{\boldsymbol{y}} - \boldsymbol{y}\right\|_2^2. \tag{42}$$

The shared weights accumulate the local gradients across the unrolled sequence, with each timestep's contribution computed solely from its own local loss and $\boldsymbol{h}_{t-1}$ held fixed:

$$\Delta\boldsymbol{W}_{xh} = -\eta \sum_{t=1}^{T} \left(\boldsymbol{h}_t - \boldsymbol{\tau}_t\right) \odot \sigma_h'(\boldsymbol{z}_t)\, \boldsymbol{x}_t^\top, \tag{43}$$

$$\Delta\boldsymbol{W}_{hh} = -\eta \sum_{t=1}^{T} \left(\boldsymbol{h}_t - \boldsymbol{\tau}_t\right) \odot \sigma_h'(\boldsymbol{z}_t)\, \boldsymbol{h}_{t-1}^\top, \tag{44}$$

$$\Delta\boldsymbol{W}_{hy} = -\eta \left(\hat{\boldsymbol{y}} - \boldsymbol{y}\right) \boldsymbol{h}_T^\top, \tag{45}$$

where $\boldsymbol{z}_t = \boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h$ and $\sigma_h'(\boldsymbol{z}_t)$ is derivative of the activation function. Updates are applied with momentum 0.9, as in the feedforward experiments.

Holding $\boldsymbol{h}_{t-1}$ fixed when differentiating $\mathcal{L}_t$ is what distinguishes FTP-RNN from BPTT: the computational graph is never traversed in reverse temporal order, no error is transported through transposed weight matrices, and all computation proceeds forward in time. FTP-RNN does, however, retain the storage of per-timestep activations, since $\boldsymbol{h}_t$ must be held until τ_t is computed. This corresponds to a synaptic eligibility trace rather than a stored computational graph, and we discuss it as a limitation in Section A.

Algorithm 2: FTP for Recurrent Networks

Given: Input window $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$, output target $\boldsymbol{y}$
$\boldsymbol{h}_0 = \boldsymbol{0}$
# First Forward Pass
for $t = 1$ **to** T **do**
  $\boldsymbol{h}_t = \sigma_h(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h)$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{W}_{hy}\boldsymbol{h}_T + \boldsymbol{b}_y$
Initialize Target State
$\boldsymbol{\tau}_0 = \texttt{stop_grad}\big(\sigma(\boldsymbol{G}\boldsymbol{y}) - \sigma(\boldsymbol{G}\hat{\boldsymbol{y}})\big)$
# Second Forward Pass (Targets)
for $t = 1$ **to** T **do**
  $\tau_t = \texttt{stop_grad}\big(\sigma_h(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{\tau}_{t-1} + \boldsymbol{b}_h)\big)$
end for
# Local Updates
for $t = 1$ **to** T **do**
  $\Delta\boldsymbol{W}_{xh} \mathrel{-}= \eta\nabla_{\boldsymbol{W}_{xh}}\mathcal{L}_t, \quad \Delta\boldsymbol{W}_{hh} \mathrel{-}= \eta\nabla_{\boldsymbol{W}_{hh}}\mathcal{L}_t$
end for
$\Delta\boldsymbol{W}_{hy} = -\eta\nabla_{\boldsymbol{W}_{hy}}\mathcal{L}_{\text{out}}$

We evaluated RNNs on three multivariate time-series datasets:

- *Electricity*: Electricity consumption data from 321 clients, recorded every 15 minutes from 2012 to 2014, and resampled to hourly resolution.
- *METR-LA*: Hourly road occupancy rates from 2015–2016, recorded by sensors on Los Angeles-area loop-detector data.
- *Solar-Energy*: Solar power output sampled every 10 minutes during 2006 from 137 photovoltaic plants in Alabama.

Each task was structured as a sequence prediction problem, where a sliding window of the previous T time steps was used to predict the next. We evaluate $T \in \{24, 48, 96, 120\}$; results for $T = 24$ appear in Table 3, and the remaining input windows in Appendix I. The RNN architecture consisted of a single recurrent layer with 512 hidden units. All RNN models were trained for 500 epochs, with the learning rate decayed by a factor of 10 at epochs 300 and 450.

E.4 Autoencoder

FTP is target-driven rather than label-driven. What the algorithm requires is a signal at the output layer, which the fixed matrix $\boldsymbol{G}$ projects into the hidden layers; nothing in the derivation of Eqs. (1)–(6) assumes that this signal is a one-hot class vector. FTP is designed as a supervised learning rule, but it works for all those cases which may not have class labels as output. Image Reconstruction is the natural such case: the target is the input itself.

To demonstrate this concretely, we trained an autoencoder with the architecture: $784 \to 512 \to 128 \to 512 \to 784$, with the 128-dimensional bottleneck serving as the learned representation. All layers are bias-free with `tanh` hidden activations and a sigmoid reconstruction layer, initialized with He-uniform initialization as in the FC experiments. A dropout rate of 0.1 is applied after each hidden layer, with the same mask presented in both forward passes.

The only change relative to the supervised formulation is the target: $\boldsymbol{y} = \boldsymbol{X}$, the input image itself. The projection $\boldsymbol{G} \in \mathbb{R}^{512\times784}$ maps the pixel-space target into the first hidden layer, drawn from a He-uniform distribution. All local update rules are unchanged.

The output layer is trained with binary cross-entropy on a sigmoid output, so that the derivative with respect to the output pre-activation is exactly $\hat{\boldsymbol{y}} - \boldsymbol{y}$ and the final-layer update retains the same form as in the softmax–cross-entropy case. Models were trained for 100 epochs using SGD with momentum 0.9, learning rate 0.01, and batch size 128, with the learning rate decayed by a factor of 10 at epochs 60 and 90. The BP baseline is identical except that gradients are backpropagated through the network.

For the linear probe, the encoder is frozen and the 128-dimensional bottleneck is extracted with dropout disabled. A single linear layer $\mathbb{R}^{128} \to \mathbb{R}^{10}$ is trained on the frozen features with cross-entropy loss, using SGD with momentum of 0.9 and learning rate 10^{-2}, batch size 128, for 100 epochs. Results are averaged over 10 seeds.

Performance is assessed along two axes. Reconstruction loss measures the fidelity of the learned encoding–decoding map, while a linear probe, a single linear classifier trained on the frozen 128-dimensional bottleneck, quantifies how much class-relevant structure the network recovers without supervision. Table 7 reports both. FTP attains reconstruction error slightly below that of BP, and the two methods differ by 0.23 on the linear probe, within one standard deviation across seeds.

Table 7: Autoencoder trained on MNIST without labels. Linear probe is trained on the frozen 128-dimensional bottleneck. Mean ± std over 10 seeds.

Method	Reconstruction loss (BCE) ↓	Linear-probe accuracy (%) ↑
BP	0.0924 ± 0.007	91.51 ± 0.09
FTP	0.0901 ± 0.010	91.28 ± 0.16

F Sensitivity to Initialization and Dimensionality of Feedback Matrix $\boldsymbol{G}$

The feedback matrix $\boldsymbol{G}$ in FTP plays a crucial role in projecting the local error signal (Eq. 1). In our experiments, we initialized $\boldsymbol{G}$ using He Uniform (He et al., 2015), scaled by a constant factor (0.05) to regulate the strength of the feedback signal. Since $\boldsymbol{G}$ directly affects the magnitude of projected error, its initialization impacts learning dynamics and gradient alignment.

To assess sensitivity to initialization schemes, we additionally trained the FC model with *He Normal* initialization (He et al., 2015), using a similar scaling factor. As shown in Table 8, FTP yields comparable performance under both schemes, suggesting that while initialization matters, the method is not overly sensitive to the exact distribution as long as the scale is appropriately controlled.

Table 8: FTP accuracy (%) under different initialization schemes for $\boldsymbol{G}$. Mean $\pm$ std over 5 trials.

Initialization	**MNIST**	**FMNIST**	**CIFAR-10**
He Uniform	$97.98_{\pm 0.25}$	$87.24_{\pm 0.21}$	$52.57_{\pm 0.37}$
He Normal	$97.51_{\pm 0.31}$	$87.33_{\pm 0.18}$	$53.44_{\pm 0.26}$

The dimensionality of $\boldsymbol{G}$ depends on the number of output classes and the hidden layer size. As shown in Table 1, FTP achieves strong performance on both CIFAR-10 and CIFAR-100, which differ significantly in class count (10 vs. 100), indicating generalizability across different output dimensionalities.

To further isolate the effect of class count, we reformulated MNIST into 5-class and 2-class variants (e.g., grouping digits by adjacent pairs or by even/odd labels). As shown in Table 9, FTP maintains performance comparable to BP across all class settings. These results indicate that FTP is robust to variations in $\boldsymbol{G}$'s dimensionality induced by class count.

Table 9: Accuracy (%) on MNIST variants with reduced class count. Mean ± std over 5 trials.

Method	MNIST (10)	MNIST (5)	MNIST (2)
BP	98.27±0.08	98.14±0.06	98.95±0.04
FTP	97.98±0.25	97.81±0.16	98.66±0.12

G Metrics for Evaluation of RNNs

Recurrent neural networks are commonly evaluated not only by accuracy but also by statistical measures that capture both error magnitude and temporal correlation. In this work, we use Root Relative Squared Error (RRSE) and the Pearson correlation coefficient (CORR), defined as follows:

$$\text{RRSE} = \sqrt{\frac{\sum_t (y_t - \hat{y}_t)^2}{\sum_t (y_t - \bar{y})^2}}, \quad \text{CORR} = \frac{\sum_t (y_t - \bar{y})(\hat{y}_t - \bar{\hat{y}})}{\sqrt{\sum_t (y_t - \bar{y})^2}\sqrt{\sum_t (\hat{y}_t - \bar{\hat{y}})^2}} \tag{46}$$

where y_t is the ground truth value, $\hat{y}_t$ is the predicted value, $\bar{y}$ and $\bar{\hat{y}}$ are the means of the ground truth and predictions, respectively.

RRSE (Root Relative Squared Error) normalizes the prediction error relative to the variance of the true series, which ensures comparability across datasets of different scales. A lower RRSE value indicates better predictive performance. CORR (Pearson correlation coefficient) measures the linear relationship between the predicted and actual sequences, capturing how well the model tracks temporal patterns. Values of CORR closer to 1 imply stronger correlation and more faithful temporal modeling. Together, these two complementary metrics provide a robust evaluation: RRSE quantifies absolute predictive accuracy, while CORR highlights alignment with the temporal dynamics of the data.

H Depth Scaling of Forward Learning Algorithms

Our initial experiments in Section 4.2 used a two-layer FC network with hidden dimensions of 1024–128, but under this setting PEPITA exhibited unstable training for deeper networks. To ensure a fair comparison, and following the setup in Srinivasan et al. (2023), we standardized the hidden layer size to 1024 units across all layers, allowing us to scale architectures to 2–5 layers while maintaining stability for both FTP and PEPITA. Consistent with prior observations by Srinivasan et al. (2023), PEPITA struggled to scale beyond three hidden layers: normalization improved convergence for deeper models but accuracy still declined with depth. In our experiments without normalization (Table 4), PEPITA's performance degraded as depth increased, whereas FTP remained stable and achieved accuracy close to BP without requiring normalization or additional techniques such as weight decay or weight mirroring. These results suggest that FTP scales more reliably to deeper FC networks without auxiliary mechanisms, whereas PEPITA shows reduced stability beyond shallow models. On the other hand, Forward-Forward Learning has been limited to four hidden layers (Hinton, 2022). This points to a significant improvement over prior forward-only learning approaches.

For convolutional networks, we extended the baseline 1-Conv design by adding a second convolutional layer (3×3, 64 channels). In these settings, FTP again outperforms PEPITA, demonstrating its potential to scale effectively to CNNs.

I Forecasting at Longer Input Windows for Sequential Tasks

The sequential experiments in Section 4.3 use a 24-timestep input window. To test whether FTP remains stable as the unrolled recurrent depth grows, we repeat the evaluation on all three benchmarks: Electricity, METR-LA (Traffic), and Solar Energy (Lai et al., 2017) at input windows of $\{24, 48, 96, 120\}$ timesteps. The architecture, initialization, optimizer, and training schedule are identical to those in Appendix E.3 and Section 4.3; only the window length changes. We report Root Relative Squared Error (RRSE) and the

Table 10: Test accuracy (Mean ± Std) of BP, PEPITA, and FTP on CNNs with 1 and 2 convolutional layers.

Dataset	Algorithm	1-Conv	2-Conv
MNIST	BP	98.74±0.05	99.05±0.03
	PEPITA	98.41±0.24	98.35±0.05
	FTP	**98.28**±0.35	**98.69**±0.07
CIFAR-10	BP	64.88±0.18	69.42±0.13
	PEPITA	56.17±0.62	57.01±0.58
	FTP	**56.32**±0.82	**59.57**±0.13
CIFAR-100	BP	33.83±0.26	37.75±0.21
	PEPITA	26.77±0.87	27.61±0.20
	FTP	**26.84**±1.13	**29.95**±0.22

Pearson correlation coefficient (CORR), defined in Appendix G, as mean ± std over 10 random seeds, and compare FTP-RNN against BP (BPTT) and PEPITA-RNN. The 24-timestep rows reproduce Table 3.

Table 11: Forecasting performance across input windows $\{24, 48, 96, 120\}$ on the Electricity, METR-LA, and Solar benchmarks (mean ± std over 10 seeds). ↑ (↓) indicates that higher (lower) values are better.

Win.	Method	Electricity		METR-LA		Solar	
		RRSE↓	CORR↑	RRSE↓	CORR↑	RRSE↓	CORR↑
24	BP (BPTT)	0.1059 ± 0.0013	0.9302 ± 0.0011	0.4680 ± 0.0083	0.8750 ± 0.0014	0.1161 ± 0.0009	0.9919 ± 0.0009
	PEPITA-RNN	0.1214 ± 0.0019	0.9929 ± 0.0009	0.4419 ± 0.0065	0.8967 ± 0.0022	0.1170 ± 0.0017	0.9932 ± 0.0022
	FTP-RNN	0.1219 ± 0.0010	0.9935 ± 0.0010	0.4398 ± 0.0073	0.8987 ± 0.0019	0.1177 ± 0.0011	0.9931 ± 0.0019
48	BP (BPTT)	0.1203 ± 0.0005	0.9933 ± 0.0010	0.4413 ± 0.0012	0.8975 ± 0.0007	0.1090 ± 0.0006	0.9940 ± 0.0010
	PEPITA-RNN	0.1682 ± 0.0013	0.9904 ± 0.0018	0.4607 ± 0.0018	0.8878 ± 0.0011	0.1394 ± 0.0015	0.9902 ± 0.0012
	FTP-RNN	0.1282 ± 0.0010	0.9954 ± 0.0008	0.4521 ± 0.0015	0.8948 ± 0.0010	0.1144 ± 0.0005	0.9917 ± 0.0017
96	BP (BPTT)	0.1230 ± 0.0008	0.9930 ± 0.0006	0.4421 ± 0.0006	0.8971 ± 0.0008	0.1089 ± 0.0004	0.9941 ± 0.0012
	PEPITA-RNN	0.1699 ± 0.0011	0.9902 ± 0.0007	0.4601 ± 0.0019	0.8880 ± 0.0020	0.1390 ± 0.0009	0.9903 ± 0.0012
	FTP-RNN	0.1399 ± 0.0013	0.9919 ± 0.0014	0.4372 ± 0.0013	0.8939 ± 0.0020	0.1139 ± 0.0009	0.9914 ± 0.0012
120	BP (BPTT)	0.1214 ± 0.0007	0.9932 ± 0.0005	0.4413 ± 0.0008	0.8975 ± 0.0004	0.1090 ± 0.0009	0.9940 ± 0.0014
	PEPITA-RNN	0.1724 ± 0.0017	0.9903 ± 0.0011	0.4621 ± 0.0015	0.8870 ± 0.0013	0.1395 ± 0.0006	0.9902 ± 0.0014
	FTP-RNN	0.1387 ± 0.0010	0.9942 ± 0.0004	0.4557 ± 0.0016	0.8954 ± 0.0013	0.1170 ± 0.0008	0.9915 ± 0.0010

FTP remains stable as the unrolled depth grows: across a fivefold increase in window length (24 to 120), RRSE stays within a narrow band on every dataset and CORR remains above 0.89 on METR-LA and above 0.99 on Electricity and Solar. FTP-RNN outperforms PEPITA-RNN on both metrics, on every dataset, at the 48, 96, and 120-step windows, with non-overlapping RRSE error bands; the two are comparable at the 24-step window, so FTP-RNN's advantage over PEPITA-RNN grows as the window length increases. It also matches or exceeds BP in correlation at several settings, achieving the highest CORR of any method on Electricity at the 48 and 120-step windows (0.9954 and 0.9942, versus BP's 0.9933 and 0.9932), and the lowest RRSE of any method on METR-LA at the 96-step window (0.4372 versus BP's 0.4421), in each case with separated error bands. BP retains a modest RRSE advantage on Electricity and Solar. Overall, lengthening the input window does not degrade FTP relative to BPTT, indicating that its forward-only target propagation assigns credit reliably across longer temporal unrolls.

J Robustness of FTP compared to Feedback Alignment

To evaluate the robustness of FTP against Feedback Alignment (FA)(Lillicrap et al., 2016), we conducted experiments under the same setup as described in Section 4.6, across 10 independent trials. Unlike backprop-

agation, FA employs fixed random feedback matrices, eliminating the need to reprogram backward paths in hardware. Although FA is less biologically plausible than FTP, its fixed feedback makes it an important baseline for robustness under hardware constraints. We tested both 8-bit and 4-bit weight precision settings under varying levels of additive noise (0, 0.02, 0.06, 0.1). The results are reported in Table 12.

Table 12: Robustness comparison of BP, FA, and FTP under 8-bit and 4-bit precision with increasing noise levels. Accuracy reported as mean ± std over 10 trials.

	Noise Levels			
Method (8-bit)	**0**	**0.02**	**0.06**	**0.1**
BP (symmetry = 100%)	**97.91**±0.08	**95.60**±0.07	**90.81**±0.17	84.56±0.81
FA	97.50±0.08	94.29±0.19	88.10±0.53	81.54±1.26
FTP	97.49±0.19	95.17±0.22	90.48±0.29	**85.85**±0.56

	Noise Levels			
Method (4-bit)	**0**	**0.02**	**0.06**	**0.1**
BP (symmetry = 100%)	**94.48**±0.11	90.12±0.15	81.25±1.34	11.35±0.00
FA	93.69±0.34	90.24±0.38	78.34±6.71	17.83±10.54
FTP	93.52±0.25	**90.89**±0.27	**82.48**±2.53	**24.73**±16.98

As reflected in Figure 3 and the Table 12, we observe that under ideal conditions (i.e., no noise), BP and FA achieve slightly higher accuracy than FTP in low bit-precision settings. However, as noise increases, FTP shows consistent and greater robustness. This suggests that FTP has inherent robustness properties that enable it to maintain stable learning dynamics under realistic hardware constraints such as low-bit precision and noise.

Importantly, since FTP uses the same forward matrices to compute both the standard activations and the layer-wise targets, any noise introduced in these matrices affects both pathways in a consistent manner. In contrast, BP requires separate programming of the forward and backward matrices in each iteration, which means programming noise can independently corrupt the two directions. This also applies for FA. This difference may have a greater impact on BP's updates compared to FTP, which may explain FTP's more stable performance in these noisy settings. In real analog hardware, additional non-idealities, such as device drift, retention loss, and other errors from the system could further exacerbate this asymmetry for BP, especially due to its reliance on reprogramming of backward matrices. Investigating these broader effects through real hardware implementation remains an important direction for future work.

K Comparison with Additional Forward-Learning Baselines

Here, we compare FTP against a broader set of forward-learning methods along with PEPITA: Fixed-Weight Difference Target Propagation (FW-DTP) (Shibuya et al., 2023), which replaces DTP's learned inverses with fixed random feedback, and Signal Propagation (Sigprop) (Kohan et al., 2022), which delivers a global learning signal through the feedforward pathway. We evaluate both under the two lenses already used in the main text: the multiply–accumulate (MAC) cost analysis of Section 4.2 and the low-precision programming-noise protocol of Section 4.6 (and Appendix J).

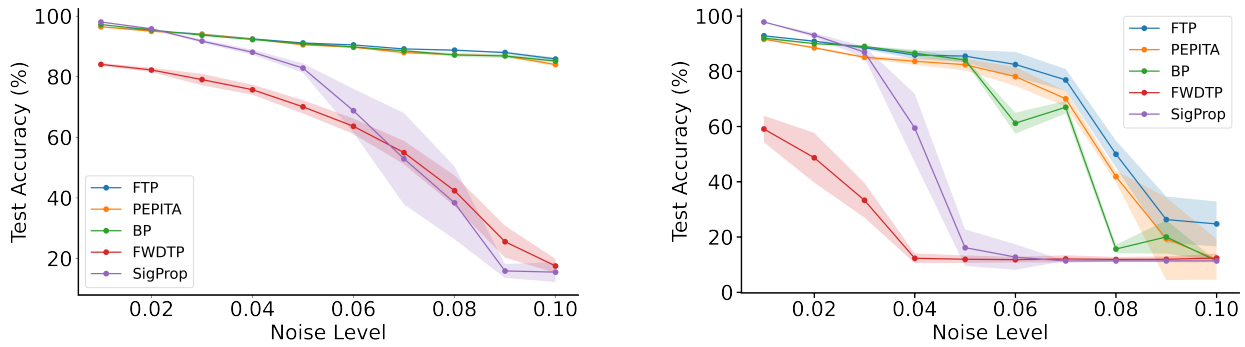
Computational cost. Table 13 reports the per-sample MAC count of each method as a percentage increase over BP on MNIST, CIFAR-10, and CIFAR-100. FW-DTP and Sigprop both require 32–53% more MACs than BP, in the same range as PEPITA and DTP, whereas FTP stays within 1–3% of BP.

Robustness under quantization and programming noise. We repeat the programming-noise protocol of Section 4.6 on MNIST, injecting additive Gaussian noise of level α (with standard deviation std $= \alpha \times |w|$)

Table 13: Per-sample MAC cost of forward-learning methods, reported as percentage increase over BP (↓ is better).

Method	MNIST	CIFAR-10	CIFAR-100
FW-DTP	+53.3%	+33.8%	+33.8%
Sigprop	+41.1%	+32.1%	+33.3%
PEPITA	+40.5%	+47.4%	+51.5%
FTP	**+1.0%**	**+0.3%**	**+3.1%**

at both 8-bit and 4-bit precision, and compare FTP against BP (here evaluated at 0% weight symmetry, as discussed in Section N), PEPITA, FWDTP, and SigProp. Figure 4 shows the resulting accuracy–noise curves.



(a) (b)

Figure 4: Performance of FTP and other algorithms when programming errors are considered for (a) 8-bit and (b) 4-bit precision devices.

At 8-bit precision (Figure 4a), FTP, PEPITA, and BP remain tightly clustered and degrade only gently across the full noise range, all retaining roughly 85% accuracy at $\alpha = 0.10$, with FTP marginally highest under the strongest noise. The two added baselines fare worse: SigProp starts highest but collapses toward chance once α exceeds about 0.05, and FWDTP declines steadily from the outset, falling below all other methods for most of the range.

At 4-bit precision (Figure 4b) the separation sharpens. FWDTP fails earliest, reaching near-chance accuracy by $\alpha \approx 0.04$, and SigProp collapses shortly after around $\alpha \approx 0.05$. PEPITA and BP track FTP until $\alpha \approx 0.07$ and then drop steeply, with BP reaching chance accuracy by $\alpha = 0.10$. FTP degrades the most gracefully of all methods, retaining the highest accuracy as noise grows and remaining well above chance where BP has already collapsed. Overall, FTP is the only method that is simultaneously within a few percent of BP's MAC cost and more robust than BP under low-precision programming noise.

L Choice of Target Projection: $\sigma(Gy) - \sigma(Gh_L)$ vs. $\sigma(G(y - h_L))$

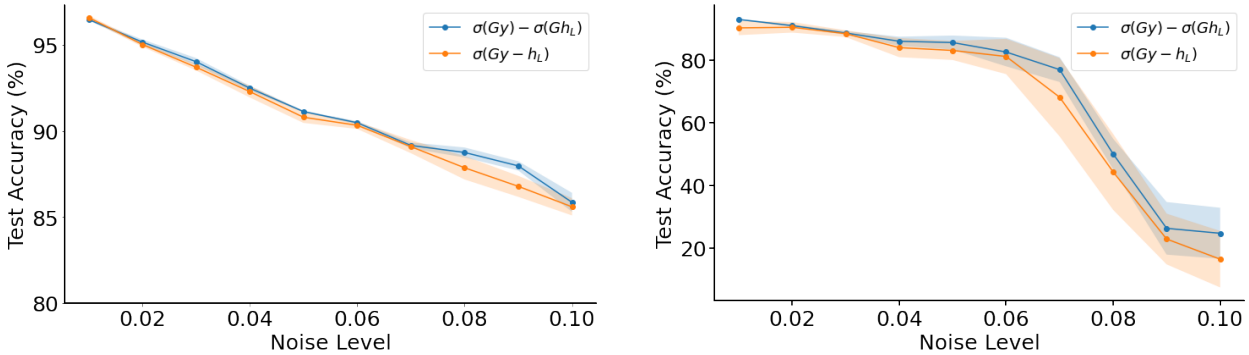
A natural question is why FTP constructs the first-layer target from the difference of two separately projected quantities, $\sigma(Gy) - \sigma(Gh_L)$, rather than from a single projection of their difference, $\sigma(G(y - h_L))$. The theoretical derivation in Appendix D is unchanged between the two forms, and on inspection they turn out to be closely related.

At the projection scale used throughout the paper, the two forms are near-equivalent. Because G is He-uniform, the projected pre-activations remain small, so σ is effectively linear and the two coincide to first order, and the derivation in Appendix D applies to either since the derivation considers linear activation. So,

$$\sigma(Gy) - \sigma(Gh_L) \;\approx\; Gy - Gh_L \;=\; Ge = G(y - h_L) \;\approx\; \sigma(G(y - h_L)) \tag{47}$$

Our choice follows difference target propagation (Lee et al., 2015), where the target for layer i is constructed as $\tau_i = h_i + g_i(\tau_{i+1}) - g_i(h_{i+1})$, i.e., the difference between two separately transformed quantities rather than a single transformation of their difference. FTP's first-hidden-layer target $\tau_1 = h_1 + \sigma(Gy) - \sigma(Gh_L)$ is the direct analogue, with a fixed random projection in place of a learned inverse. Our initial experiments also found this form marginally more robust under low-precision perturbation, and we retained it on that basis.

We compared the two forms systematically under the programming-noise protocol of Section 4.6, at 8-bit and 4-bit precision, over 10 seeds (mean and standard deviation shown as shaded regions in Figure 5). At 8-bit (Figure 5a) the two are effectively indistinguishable, differing by under one point at nearly every noise level with overlapping error bands. At 4-bit (Figure 5b) the difference form leads at most noise levels – by roughly 2.4 points on average for $\alpha \leq 0.08$ – and exhibits markedly lower variance across seeds, though the error bands overlap throughout and the ordering is not strictly uniform. We therefore present this as the reason we retained the difference form, not as evidence that it is necessary: consistent with the near-equivalence above, the two perform comparably.



(a) (b)

Figure 5: Test accuracy on MNIST under programming noise of level α for the two target projections, $\sigma(Gy) - \sigma(Gh_L)$ and $\sigma(G(y - h_L))$, at (a) 8-bit and (b) 4-bit precision. Shaded regions denote the standard deviation over 10 seeds.

M Memory and Runtime Efficiency of FTP

This section presents an analytical comparison, and the pipelining benefit described below is a property of the update schedule of FTP. We analyze the memory usage and runtime cost of FTP in comparison to BP and DFA (Nøkland, 2016), particularly in the context of resource-constrained hardware.

Memory Usage. The memory footprint during training arises from two primary sources: (i) activation storage during the forward pass, and (ii) storage of trainable parameters and feedback matrices.

Activation storage: All methods (BP, DFA, FTP) require retaining activations during the forward pass until error signals or targets are computed. Thus, peak memory usage for storing activations is comparable across all three methods.

Parameter storage: BP stores only the feedforward weights. FTP stores the same feedforward weights along with a single feedback matrix $\boldsymbol{G}$. DFA, on the other hand, maintains a separate fixed feedback matrix for each hidden layer, resulting in the largest weight memory footprint. Notably, in deeper networks, the overhead of storing a single $\boldsymbol{G}$ in FTP becomes negligible compared to the total number of trainable parameters.

Runtime Efficiency. FTP performs a standard forward pass, computes the final target, and then recursively generates intermediate layer targets while updating weights layer-by-layer. Unlike BP, which updates layers in reverse order and enforces update locking, FTP permits pipelined execution. Specifically, once the first layer's weights are updated and the next target is being computed, the subsequent minibatch can begin its forward

pass. This enables partial overlap between batches, potentially reducing wall-clock training time–particularly in hardware implementations that support pipelining.

DFA computes all layer gradients in parallel using fixed feedback matrices after the forward pass. The inter-minibatch delay in DFA corresponds to the time required to update all weights simultaneously. In contrast, for FTP, the delay is primarily determined by the computation of the first target and update of the first layer.Therefore, FTP's runtime should be comparable to DFA, with a modest overhead for target computation, and would improve on BP under a pipelined schedule. Verifying this on hardware that supports pipelined execution remains future work.

N Impact of Strict Weight Symmetry Requirement of BP in Emerging Analog Hardware

Backpropagation (BP) relies on backward matrices to propagate errors, which must maintain symmetry with the forward weight matrices in principle mathematically, which is easily achievable in the case of digital computers. However, in analog computing hardware such as RRAM and other non-volatile memories, non-idealities such as programming errors, thermal noise, and random telegraph noise can disrupt this symmetry, leading to significant degradation in performance. Our results, as shown in Figure 6, demonstrate the vulnerability of BP under these conditions, particularly in devices with low-bit precision. Here we considered the cases when some weight parameters in a matrix got corrupted due to device non-idealities, and the asymmetry here refers to the amount of weights getting corrupted by 10% margin from where the values should be. For instance, in 4-bit systems (Zidan et al., 2018; Yao et al., 2020), test accuracy drops below 80% when only 20% of the matrices are affected by such errors. The vulnerability is even more pronounced in 3-bit systems, where a mere 5% mismatch between forward and backward matrices causes test accuracy to plummet below 60%. As asymmetry increases, test accuracy steadily declines, eventually reaching approximately 15%, which is equivalent to random guess. The standard deviation of test accuracy also increases, which indicates greater instability in performance. Each case was tested multiple times, and the shadowed region in Figure 6 represents the standard deviation of test accuracy across trials. In contrast, FTP utilizes fixed, random backward matrices for target estimation, eliminating the dependency on symmetric forward-backward matrix relationships. This design inherently makes FTP resilient to hardware-induced noise and non-idealities, ensuring robust performance even in low-bit precision environments.

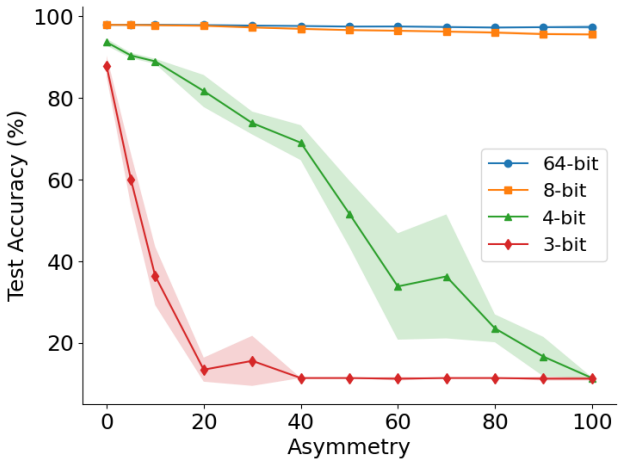


Figure 6: Performance of BP when asymmetry is introduced in backward matrices due to read noise.

O Compute Resources

All experiments were conducted on a local workstation configured with the following specifications: an AMD Ryzen Threadripper PRO 5955WX processor (16 cores, 4.00–4.50GHz), 256 GB of DDR4-3200 RAM, and dual NVIDIA GeForce RTX 4090 GPUs.

Experiments involving CNNs and RNNs were executed on the GPUs to leverage accelerated training, while fully connected network experiments and all evaluations in Section 4.4, Section 4.5 and Section 4.6 were performed on CPU only.

P Use of Large Language Models (LLMs)

Large language models (LLMs) were used in this work solely as a general-purpose assistive tool for grammar checking and text editing. All research ideas, experimental designs, analyses, and scientific claims were conceived, implemented, and validated solely by the authors.