$\mathcal{P}$Torch: Narrowing the Gap Between Projection and Gradient–Based Learning

Jeff Cyuzuzo Jambé*, **Jan Quan***, **Panagiotis Patrinos**
KU Leuven (STADIUS)

Abstract

Projection-based learning offers a gradient-free alternative to the conventional gradient-based approach, but the paradigm suffers from high memory costs, poor scaling with depth, and limited integration with modern deep learning infrastructure. We introduce $\mathcal{P}$Torch, a projection-based framework built directly on top of Torch's `autograd` engine that improves performance through fused projections and nonlinear relaxation, reducing the memory cost and making projection-based learning competitive with gradient-based baselines on shallow networks for MNIST and CIFAR-10. Lastly, we prove a vanishing-target theorem showing a layer-wise learning signal decays with depth, providing a projection-based analog of the vanishing-gradient problem and clarifying one of the central barriers to scaling these methods. $\mathcal{P}$Torch is available at https://github.com/cyuzuzo-j/PTorch.

1 Introduction

Modern deep learning relies almost exclusively on gradient-based optimization, with gradients computed end-to-end through backpropagation [50]. However, this paradigm imposes two structural requirements: every component of the neural network must admit a gradient (or a tractable subgradient), and gradients must propagate through the chain rule, thereby limiting the potential for parallelization. A further motivation for studying alternative approaches to learning is the risk of paradigm lock-in: the overwhelming focus on gradient-based training may cause the community to overlook other learning frameworks that, given comparable research investment, could ultimately prove competitive or even superior. Many alternative paradigms have been proposed in the literature, yet none has achieved widespread adoption, often due to challenges in optimization performance, scalability, or practical integration with modern deep learning systems.

A mathematically grounded alternative that has recently garnered attention is projection-based learning (PBL), which recasts learning as a feasibility problem [23, 9]. In its basic form, instead of minimizing a global loss function $\mathcal{L}$, this paradigm seeks a point ω at the intersection of multiple constraint sets C_i. These sets are chosen such that, whenever their intersection is nonempty, any point ω in the intersection yields neural network parameters θ that perfectly interpolate the training data. Conceptually, this paradigm shift takes the form

$$\underset{\theta\in\Theta}{\text{minimize}}\ \mathcal{L}(f(x;\theta),y) \quad \longrightarrow \quad \text{find } \omega \in \bigcap_{i=1}^{m} C_i.$$

Nevertheless, the paradigm still faces fundamental challenges. We highlight two challenges identified in [9, §6] that form the basis of our research question.

*Equal contribution. This work was carried out as part of the master's thesis of Jeff Cyuzuzo Jambé at KU Leuven.
E-mails: cyuzuzo.jambe@gmail.com, jan.quan@kuleuven.be, panos.patrinos@kuleuven.be

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

C1. **Memory overhead and limited systems integration.** The approaches in [23, 9] organize the constraints into two sets to enable Douglas–Rachford-type methods. To make projections onto these sets tractable, variables are duplicated and consistency is enforced through consensus constraints. This duplication introduces substantial memory overhead, which makes it difficult to scale the paradigm to deeper networks. A related challenge is that this two-set partition makes it difficult to fully benefit from the modern software optimizations that underpin state-of-the-art libraries such as Torch [47] and JAX [13].

C2. **Optimization performance.** The performance of projection-based learning still falls short of standard gradient-based optimization methods such as SGD [49] and Adam [35], even on shallow networks. Many of the algorithmic advances that have made gradient-based training effective still lack counterparts in PBL, while modern architectures and training heuristics have largely co-evolved with gradient-based optimization. Moreover, [9] observes empirically that performance deteriorates with network depth.

These limitations naturally lead us to our main research question.

> Can we design a projection-based training framework that integrates natively with modern deep learning toolchains, scales without prohibitive memory overhead, and achieves performance competitive with gradient-based methods on shallow networks?

Contributions We address these challenges by introducing $\mathcal{P}$Torch. Concretely, our contributions are as follows.

1. **A Torch-native projection framework.** We exploit a connection between cyclic projections and the backward traversal of Torch's `autograd` computational graph to develop an extensible, Torch-native projection framework that can benefit from modern software optimizations.
2. **Reducing memory overhead and improving performance.** To mitigate memory constraints, we introduce a fused projection strategy for cyclic projections that avoids explicitly duplicating variables for bilinear layers, which are fundamental components of many neural networks. Further, we apply modern first-order optimizers such as Adam [35] and Muon [34] to the weight projection residuals as nonlinear relaxation schemes, which substantially improve empirical performance. Together, these modifications yield substantial empirical gains, enabling several $\mathcal{P}$Torch configurations to match or exceed the evaluated gradient-based baselines on shallow networks. We also consider projections in the ℓ_∞ geometry and derive a novel and efficient algorithm for computing the ℓ_∞ projection for bilinear layers.
3. **Vanishing targets.** We prove that, in the Euclidean setting, the propagated target signal contracts layer by layer under a Lipschitz condition, yielding exponential attenuation with depth. This behavior is analogous to the vanishing-gradient problem, thereby limiting the scalability to deeper networks. Successfully scaling these models presents a clear avenue of future work: discovering the projection-based equivalents of techniques such as He initialization [31], batch normalization [33], and residual connections [30].

1.1 Related work

Projection-based algorithms for feasibility problems Projection-based methods are a classical approach to feasibility problems. Examples include cyclic projections [29], Bregman projections [14], Dykstra's algorithm [21], and averaged projections [3]. For two sets ($m = 2$), the Douglas–Rachford method [20, 22] has been studied extensively and applied to a wide range of problems [7, 42]. Multiple sets can be handled through a product-space reformulation [48], while direct multi-set extensions include cyclic and averaged Douglas–Rachford schemes [11]. Convergence theory for these methods has been developed primarily in the convex setting [5], and there have been advances in the nonconvex setting as well [55, 2, 40]. Inconsistent feasibility problems, where the constraint sets have empty intersection, have also received attention. For closed convex sets, approaches include minimizing weighted sums of squared distances to the sets [18] and, in the two-set case, seeking best approximation pairs [6].

Projection-based learning Early accounts of projection-based methods applied to learning include alternating projection neural networks [44] and generalized-projection methods for training multilayer networks [60]. In [27], internal network states are introduced as variables, and learning seeks representations that make the input-to-hidden and hidden-to-output mappings consistent. This formulation obviates the need for a global loss function. Auxiliary variables representing internal states have also been used to decouple network layers in [15, 54].

A projection-based framework for modern neural networks that allows for nondifferentiable layers is developed in [23]. Weights and activations are treated as joint variables subject to local bilinear, activation, data, and consensus constraints. These constraints are organized into two sets, allowing local projections within each set to be computed in parallel across data and network architecture. Training is then performed using a relaxed Douglas–Rachford method, and the framework is extended to accommodate corrupted data and incorrect labels. The underlying constraint splitting and bilinear projection operators are further analyzed in [38]. This approach has also been extended to Boolean threshold networks through local threshold and architectural consistency constraints [24].

Building on [23], PJAX [9] provides a JAX-based software library that composes local projection operators and supports parallel execution across the computation graph. It accommodates output loss functions through proximal operators and supports multiple projection algorithms, including alternating projections, cyclic projections, and Douglas–Rachford. The library supports standard feedforward, convolutional, and recurrent architectures with GPU/TPU acceleration. In addition, [9] offers an excellent resource that explains PBL in detail, including various techniques and trade-offs within the paradigm. Further related work is discussed in Appendix A.

2 Neural network training as a feasibility problem

In this section, we review the conceptual framework for projection-based learning introduced by [23] and establish the notation used throughout the paper.

2.1 Intuitive description

Conceptually, a neural network can be viewed as a sequence of interlocking constraints. Each layer imposes a precise mathematical relationship: its output must equal a certain transformation of its inputs. The overall goal of training is for the network to fit a target dataset. By fixing the network inputs and outputs to the data, we impose global boundary conditions that may violate the network's internal constraints. Training therefore amounts to resolving these local inconsistencies. Rather than computing global gradients that must be propagated through the chain rule, we aim to solve a local optimization problem at each layer:

What is the minimal adjustment to the local weights, inputs, and outputs needed to exactly re-satisfy the layer constraint?

In $\mathcal{P}$Torch, we apply these local corrections while traversing the computation graph from the output back to the input, thereby propagating a learning signal without requiring gradients of a global loss.

2.2 Mathematical description

Consider the minibatch $\{(x_1, y_1), \dots, (x_B, y_B)\}$ and suppose we have a neural network with N layers and parameters $\theta^{(1)}, \dots, \theta^{(N)}$. For ease of presentation, we assume the network has a chain structure, although the extension to more general topologies is straightforward. In particular, the forward pass for a single sample (x_b, y_b) can be written as

$$\begin{aligned} h_b^{(0)} &= x_b \\ h_b^{(i)} &= f_i(h_b^{(i-1)}, \theta^{(i)}), \quad i = 1, \dots, N \end{aligned} \tag{1}$$

for some functions $f_1, \dots, f_N$. The output of the network is $h^{(N)}$.

Example 2.1. A standard multilayer perceptron block consists of a linear layer $f_i(h, W^{(i)}) = W^{(i)}h$ (where we have the parameter $\theta^{(i)} = W^{(i)}$), followed by an activation function $f_{i+1}(h) = \Sigma(h)$. The latter has no learnable parameters.

We model neural network training as a feasibility problem in high-dimensional space following [23]. For each sample b, let $h_b^{(0)}, \dots, h_b^{(N)}$ denote the hidden states, with parameters $\theta^{(1)}, \dots, \theta^{(N)}$ shared across the minibatch. Further, we assume that $f_i = (f_{i1}, \dots, f_{iK_i})$, that is, the output of the ith layer can be written as K_i distinct blocks

$$h_b^{(i)} = (h_{b1}^{(i)}, \dots, h_{bK_i}^{(i)}) = (f_{i1}(h_b^{(i-1)}, \theta^{(i)}), \dots, f_{iK_i}(h_b^{(i-1)}, \theta^{(i)})).$$

We introduce the copies $h_b^{(i-1,k)}$ and $\theta_b^{(i,k)}$ for each $k = 1, \dots, K_i$, and let ω denote the variable containing all our variables. For the ith layer, bth sample, and kth output block, we then obtain the constraint set

$$A_{bik} \coloneqq \{\omega : h_{bk}^{(i)} = f_{ik}(h_b^{(i-1,k)}, \theta_b^{(i,k)})\}.$$

Moreover, all copies of the same variable must be consistent, which we handle by imposing consensus constraints. In particular, we must have $h_b^{(i-1,1)} = \cdots = h_b^{(i-1,K_i)}$ and similarly for all copies representing $\theta^{(i)}$ (note that these are duplicated across the batch and the output blocks). Throughout this paper, we use $h_b^{(i-1)}$ and $\theta^{(i)}$ to denote these common values when consensus holds. The feasibility problem then consists of finding

$$\text{find } \omega \in \left(\bigcap_{b=1}^{B} \bigcap_{i=1}^{N} \bigcap_{k=1}^{K_i} A_{bik} \right) \bigcap (B_{\text{in}} \cap B_{\text{out}}) \bigcap \left(\bigcap_{i=1}^{N} C_i \right), \tag{P}$$

where the three types of constraints are given by

1. **A**rchitectural constraints: Every layer must satisfy its corresponding input-output transformation. In particular, if we denote $A_{bi} \coloneqq \bigcap_{k=1}^{K_i} A_{bik}$, then $\omega \in A_{bi} \cap C_i$ ensures that $h_b^{(i)} = f_i(h_b^{(i-1)}, \theta^{(i)})$ for the bth data point and the ith layer.
2. **B**atch constraints: The inputs and outputs of the network are fixed to match the training data, i.e., $\omega \in B_{\text{in}} \cap B_{\text{out}}$ means that $h_b^{(0)} = x_b$ and $h_b^{(N)} = y_b$ for each $b = 1, \dots, B$.
3. **C**onsensus constraints: Copies representing the same underlying variable must agree. Concretely, $\omega \in C_i$ enforces $h_b^{(i-1,1)} = \cdots = h_b^{(i-1,K_i)}$ for each sample b, together with equality among parameter copies representing $\theta^{(i)}$.

Example 2.2. Consider the linear layer $h_b^{(i)} = W^{(i)} h_b^{(i-1)}$ as in Example 2.1. Suppose that $W^{(i)}$ has N_{out} rows $w_k^{(i)}$. Then, we have $K_i = N_{\text{out}}$ and $A_{bik} = \{\omega : h_{bk}^{(i)} = (w_b^{(i,k)})^\top h_b^{(i-1,k)}\}$. The consensus constraint C_i enforces $h_b^{(i-1,1)} = \cdots = h_b^{(i-1,K_i)}$ for each b and $w_1^{(i,k)} = \cdots = w_B^{(i,k)}$ for each $k = 1, \dots, K_i$, with common values $h_b^{(i-1)}$ and $w_k^{(i)}$, respectively.

Further, we recall that the feasibility problem (P) is tightly related to minimizing the objective $F(\omega)$ defined by

$$F(\omega) \coloneqq \frac{1}{2} \left\{ \sum_{b=1}^{B} \sum_{i=1}^{N} \sum_{k=1}^{K_i} \operatorname{dist}^2(A_{bik}, \omega) + \sum_{B \in \{B_{\text{in}}, B_{\text{out}}\}} \operatorname{dist}^2(B, \omega) + \sum_{i=1}^{N} \operatorname{dist}^2(C_i, \omega) \right\} \tag{2}$$

where $\operatorname{dist}^2(C, \omega) = \min_{\tilde{\omega} \in C} \|\omega - \tilde{\omega}\|^2$ for a closed set C. Since every term is nonnegative, ω solves (P) if and only if $F(\omega) = 0$.

> To summarize, if we find an ω solving (P), then the associated parameters $\theta^{(1)}, \dots, \theta^{(N)}$ (which are equal over all data points due to the consensus constraints) allow the neural network to perfectly interpolate the data.

The main computational device in PBL is the projection operator. In particular, for a closed set C, we define the Euclidean projection of a point ω onto C by

$$\Pi_C(\omega) \in \operatorname*{arg\,min}_{\tilde{\omega} \in C} \|\omega - \tilde{\omega}\|_2.$$

For nonconvex sets, this projection may not be unique.

Example 2.3. For the consensus set $C = \{(x_1, \dots, x_m) : x_1 = \cdots = x_m\}$, we have that $\Pi_C((y_1, \dots, y_m)) = (\tilde{y}, \dots, \tilde{y})$ where $\tilde{y}$ is the average $\frac{1}{m} \sum_{i=1}^{m} y_i$. Other well-known projections for neural networks can be found in [23, 9] and Appendix E.

3 The $\mathcal{P}$Torch algorithm

This section develops the $\mathcal{P}$Torch algorithm step by step. We begin with cyclic projections, which naturally aligns with the backward traversal of Torch's computational graph. We then introduce fused projections to reduce the memory overhead caused by explicit variable duplication, followed by nonlinear relaxation to improve optimization performance. We aim to motivate each design choice by the challenges C1 and C2.

3.1 Cyclic projections

Our first design choice is to retain the individual constraint sets in (P) instead of partitioning them into the two sets required by Douglas–Rachford methods. Cyclic projections are a natural fit because they operate directly on these sets, applying the corresponding projections successively in a fixed order. This choice is useful for two reasons. First, it allows the individual projection and consensus operations to be fused later without explicitly materializing duplicated variables. Second, its structure aligns naturally with the backward traversal of Torch's computational graph, enabling seamless integration.

The connection between a projection step and Torch's backward pass follows from a simple observation. For any x and constraint set C, the projection update can be written as $x \leftarrow x - \big(x - \Pi_C(x)\big)$. Consequently, the projection residual $x - \Pi_C(x)$ can be passed through the same interface normally used for gradients. Note that it is important that $x - \Pi_C(x)$ can be computed locally.

> If $x - \Pi_C(x)$ is written to the `.grad` attribute during the backward pass, an (S)GD step with step size 1 maps x directly to $\Pi_C(x)$. Applying this procedure sequentially during the backward traversal therefore implements cyclic projections on Torch's computational graph.

In fact, in the Euclidean setting, this observation admits an optimization interpretation. Whenever Π_C is single-valued, we have

$$\nabla_x \left(\tfrac{1}{2}\operatorname{dist}^2(C, x)\right) = x - \Pi_C(x),$$

so the projection update $x \leftarrow \Pi_C(x)$ is exactly a unit-step gradient descent update on the squared distance to C. Since cyclic projections apply these updates successively to the individual constraint sets, they can therefore be interpreted as incremental gradient descent [10] on the objective F in (2). Importantly, this is not equivalent to backpropagation on a global loss. Each incremental step uses only the gradient of a local term associated with a single constraint set, without differentiating through the network. We elaborate on this connection in Appendix B.1.

This implementation also makes $\mathcal{P}$Torch modular: a projection-based layer can serve as a drop-in replacement for its standard Torch counterpart, requiring only an implementation of the local projection operator, while leaving the usual Torch training code unchanged. The resulting cyclic projections procedure is summarized in Algorithm 1. However, a direct implementation still requires the duplicated variables introduced in (P) to be materialized explicitly. In the next section, we show how the projection and consensus operations can be fused to avoid this duplication.

Algorithm 1 Cyclic projections on the Torch computational graph.

Sample minibatch $\{(x_1, y_1), \ldots, (x_B, y_B)\}$
$h_b^{(0)} \leftarrow x_b, \quad h_b^{(i)} \leftarrow f_i(h_b^{(i-1)}, \theta^{(i)}), \quad i = 1, \ldots, N,\ b = 1, \ldots, B$ ▷ Forward pass
$h_b^{(N)} \leftarrow y_b, \quad b = 1, \ldots, B$ ▷ Enforce Batch Constraint (output)
Initialize copies in ω with corresponding values
for $i = N$ **downto** 1 **do** ▷ Traverse computational graph backwards
  for $b = 1$ **upto** B, $k = 1$ **upto** K_i **do**
    $\omega \leftarrow \Pi_{A_{bik}}(\omega)$ ▷ Enforce Architectural Constraint
  $\omega \leftarrow \Pi_{C_i}(\omega)$ ▷ Enforce Consensus Constraint
$h_b^{(0)} \leftarrow x_b, \quad b = 1, \ldots, B$ ▷ Enforce Batch Constraint (input)

3.2 Fusing projections

We now move onto reducing the memory overhead in C1 by fusing projections, thereby obviating the need for the explicit materialization of the duplicated variables. We focus on the bilinear projection underlying linear and convolutional layers. A naive implementation that explicitly materializes the duplicated variables requires constructing an $N_{\text{out}} \times B \times N_{\text{in}}$ tensor, where $N_{\text{in}}, N_{\text{out}}$ are the input and output dimensions of the linear layer, and B is the batch size. Instead, we aim to compute $\Pi_{C_i} \circ \Pi_{A_i}$ where $A_i \coloneqq \bigcap_{b=1}^{B} \bigcap_{k=1}^{K_i} A_{bik}$ without explicitly materializing the duplicated variables.

Derivation Consider the bilinear constraint set $A = \{(h, w, h^+) \in \mathbb{R}^{N_{\text{in}}} \times \mathbb{R}^{N_{\text{in}}} \times \mathbb{R} : h^+ = w^\top h\}$. Given (h, w, h^+), we consider the Euclidean projection

$$(\bar{h}, \bar{w}, \bar{h}^+) \in \operatorname*{arg\,min}_{(\tilde{h}, \tilde{w}, \tilde{h}^+) \in A} \|\tilde{h} - h\|_2^2 + \|\tilde{w} - w\|_2^2 + (\tilde{h}^+ - h^+)^2.$$

As shown in Appendix E.2, this problem reduces to solving for a scalar t using Newton's method, after which the projections take the form $\bar{w} = uw + vh$ and $\bar{h} = uh + vw$ with $u = \frac{1}{1-t^2}$ and $v = \frac{t}{1-t^2}$. Now consider a linear layer with weight matrix $W \in \mathbb{R}^{N_{\text{out}} \times N_{\text{in}}}$ and a batch of inputs $H \in \mathbb{R}^{B \times N_{\text{in}}}$. We denote the kth row of W by $w_k^\top$ and the bth input by h_b, so that $h_{bk}^+ = w_k^\top h_b$. Each pair (k, b) yields its own scalar t_{kb}, and hence coefficients u_{kb} and v_{kb}, and $\bar{w}_b^{(k)} = u_{kb} w_k + v_{kb} h_b$ and $\bar{h}_b^{(k)} = u_{kb} h_b + v_{kb} w_k$. As in Example 2.3, the resulting consensus projection $\bar{w}_k$ is given by

$$\bar{w}_k = \frac{1}{B} \sum_{b=1}^{B} \bar{w}_b^{(k)} = \frac{1}{B} \left(w_k \sum_{b=1}^{B} u_{kb} + \sum_{b=1}^{B} v_{kb} h_b \right).$$

Crucially, this expression can be evaluated without computing the $\bar{w}_b^{(k)}$ copies explicitly. Let $U, V \in \mathbb{R}^{N_{\text{out}} \times B}$ contain the coefficients u_{kb} and v_{kb}. Then, it follows that

$$\overline{W} = \frac{1}{B}[W \odot (U \mathbf{1}_{B \times N_{\text{in}}}) + VH]$$

where $\odot$ denotes the Hadamard product and $\mathbf{1}_{B \times N_{\text{in}}}$ is the all-ones matrix. In particular, the $N_{\text{out}} \times B \times N_{\text{in}}$ tensor of projected weights of each sample never needs to be materialized. Similarly, the row-wise projections also produce distinct projected inputs $\bar{h}_b^{(k)}$, which require the consensus

$$\bar{h}_b = \frac{1}{N_{\text{out}}} \left(h_b \sum_{k=1}^{N_{\text{out}}} u_{kb} + \sum_{k=1}^{N_{\text{out}}} v_{kb} w_k \right) \implies \overline{H} = \frac{1}{N_{\text{out}}} [H \odot (U^\top \mathbf{1}_{N_{\text{out}} \times N_{\text{in}}}) + V^\top W].$$

Thus, both the weight and input consensus operations can be evaluated without explicitly constructing the duplicated variables. While this approach substantially reduces the memory overhead, the underlying algorithm is still vanilla cyclic projections, which does not achieve good optimization performance, cf. C2, as also shown in Table 3. This is the issue we tackle in the following sections.

3.3 Nonlinear relaxation

The incremental gradient interpretation discussed above suggests a natural way to improve optimization performance, by passing the residuals through standard first-order optimizers such as Adam [35] or Muon [34]. We only apply this relaxation to the weight residuals, since the hidden states are sample-dependent auxiliary variables that are recalculated each forward pass, making a persistent optimizer state less natural and requiring more memory. Because $\mathcal{P}$Torch already channels projection residuals through Torch's `autograd`, implementing this requires no algorithmic modifications, allowing standard optimizers to be invoked exactly as they are in a conventional training loop.

3.4 The base $\mathcal{P}$Torch algorithm

Combining the components above yields the base $\mathcal{P}$Torch algorithm in Algorithm 2. Additionally, following [9], we replace the strict projection onto the output constraint set with a proximal step associated with the loss $\mathcal{L}$. This relaxes the requirement of exact interpolation at the output. Further motivation for this choice based on the incremental gradient interpretation is provided in Appendix B.2. We use FusedProj_i to denote the fused projection $\Pi_{C_i} \circ \Pi_{A_i}$, as described in Section 3.2. Finally, $\text{Optimizer}(\theta, g)$ denotes one step of the chosen optimizer on θ, treating g as the "gradient".

Algorithm 2 Base $\mathcal{P}$Torch algorithm

1: Sample minibatch $\{(x_1, y_1), \ldots, (x_B, y_B)\}$
2: $h_b^{(0)} \leftarrow x_b, \quad h_b^{(i)} \leftarrow f_i(h_b^{(i-1)}, \theta^{(i)}), \quad i = 1, \ldots, N,\ b = 1, \ldots, B$ ▷ Forward pass
3: $h_b^{(N)} \leftarrow \mathrm{prox}_{\lambda \mathcal{L}(\cdot, y_b)}(h_b^{(N)}), \quad b = 1, \ldots, B$ ▷ Output proximal step
4: **for** $i = N$ **downto** 1 **do** ▷ Traverse computational graph backwards
5: $(\bar{h}^{(i-1)}, \bar{\theta}^{(i)}, \bar{h}^{(i)}) \leftarrow \mathrm{FusedProj}_i(h^{(i-1)}, \theta^{(i)}, h^{(i)})$ ▷ Fused projection
6: $(h^{(i-1)}, h^{(i)}) \leftarrow (\bar{h}^{(i-1)}, \bar{h}^{(i)})$ ▷ Direct projection hidden states
7: $\theta^{(i)} \leftarrow \mathrm{Optimizer}(\theta^{(i)}, \theta^{(i)} - \bar{\theta}^{(i)})$ ▷ Nonlinear relaxation
8: $h_b^{(0)} \leftarrow x_b, \quad b = 1, \ldots, B$ ▷ Enforce Batch Constraint (input)

3.5 Projection in a different geometry

As an optional heuristic extension beyond the Euclidean setting, motivated by the connection of Adam [35] to the ℓ_∞ norm [59], we also consider an ℓ_∞ variant of the projection onto the bilinear constraint set. The local optimization objective then takes the form

$$\underset{\tilde{h} \in \mathbb{R}^{N_{\text{in}}}, \tilde{w} \in \mathbb{R}^{N_{\text{in}}}, \tilde{h}^+ \in \mathbb{R}}{\text{minimize}} \ \|(\tilde{h}, \tilde{w}, \tilde{h}^+) - (h, w, h^+)\|_\infty \quad \text{s.t.} \quad \tilde{w}^\top \tilde{h} = \tilde{h}^+ \tag{3}$$

A naive approach to solving (3) requires a sorting procedure, incurring a $\mathcal{O}(N_{\text{in}} \log N_{\text{in}})$ cost. Instead, we derive a sort-free Newton solver for which the total cost is $\mathcal{O}(T N_{\text{in}})$, where T denotes the number of Newton iterations, typically a single step. The full derivation is provided in Appendix E.3. Note that, unlike the ℓ_2 variant, this geometry does not enjoy an incremental gradient interpretation.

While this heuristic can improve final model accuracy, it comes with a memory trade-off. Unlike the ℓ_2 projection, the ℓ_∞ projection generally updates different coordinates of the weight vector in different ways. Therefore, the projection can no longer be computed based on two scalar coefficients as in Section 3.2. Consequently, it retains the heavier $\mathcal{O}(N_{\text{in}} N_{\text{out}} B)$ memory footprint. In essence, we have one geometry that often performs better empirically and another that is more memory-efficient, but not one that achieves both simultaneously. A compromise is to utilize both geometries. For example, we can use the memory-efficient ℓ_2 projections for the backbone and an ℓ_∞ projection for the classification head. This hybrid design allows us to balance performance and memory efficiency for larger networks. In fact, in many cases, the hybrid geometry even improves performance (Table 5).

4 Vanishing target signals in deep networks

In this section, we derive a fundamental limitation of the $\mathcal{P}$Torch algorithm, which we call the vanishing target theorem. Concretely, we observe that the learning signal attenuates with depth, analogous to vanishing gradients. We provide a formal characterization of this phenomenon for Euclidean projections. To this end, consider Algorithm 2 and define the *target signal magnitude* at layer i by

$$\delta^{(i)} := \|\bar{h}^{(i)} - h^{(i)}\|_2, \qquad i = 0, 1, \ldots, N$$

where the output target is denoted by $\bar{h}^{(N)} = [\mathrm{prox}_{\lambda \mathcal{L}(\cdot, y_b)}(h_b^{(N)})]_{b=1}^B$. Here, $h^{(i)} = [h_b^{(i)}]_{b=1}^B$ denotes the activations during the forward pass on line 2 of Algorithm 2, and $\bar{h}^{(i)}$ is as on line 5 of Algorithm 2.

Theorem 4.1. *Consider a backward pass of Algorithm 2 with Euclidean projections. Then, $\delta^{(i-1)} \leq \delta^{(i)}$ for each $i = 1, \ldots, N$. If, moreover, each f_{ik} is Lipschitz continuous in its first argument with constant L_i, then $\delta^{(i-1)} \leq \frac{L_i}{\sqrt{1+L_i^2}} \delta^{(i)}$. Consequently, we have the exponential attenuation*

$$\delta^{(i)} \leq c^{N-i} \delta^{(N)}, \qquad c := \max_{1 \leq j \leq N} \frac{L_j}{\sqrt{1+L_j^2}} < 1.$$

Proof. Fix $i \in \{1, \ldots, N\}$. When the backward pass reaches layer i, the argument of $\mathrm{FusedProj}_i$ is $(h^{(i-1)}, \theta^{(i)}, \bar{h}^{(i)})$. Let

$$(\tilde{h}_b^{(i-1,k)}, \tilde{\theta}_b^{(i,k)}, \tilde{h}_{bk}^{(i)}) \in \Pi_{A_{bik}}(h_b^{(i-1,k)}, \theta_b^{(i,k)}, \bar{h}_{bk}^{(i)})$$

denote the projection onto A_{bik} prior to the consensus projection. Due to the forward pass, we have $(h_b^{(i-1,k)}, \theta_b^{(i,k)}, h_{bk}^{(i)}) \in A_{bik}$ for every $b = 1, \dots, B$ and $k = 1, \dots, K_i$. It follows that

$$\|\tilde{h}_b^{(i-1,k)} - h_b^{(i-1,k)}\|_2^2 + \|\tilde{\theta}_b^{(i,k)} - \theta_b^{(i,k)}\|_2^2 + \|\tilde{h}_{bk}^{(i)} - \bar{h}_{bk}^{(i)}\|_2^2 \le \|\bar{h}_{bk}^{(i)} - h_{bk}^{(i)}\|_2^2. \tag{4}$$

The subsequent projection onto C_i averages all copies representing the same underlying variable as in Example 2.3. Therefore,

$$\begin{aligned}\|\bar{h}_b^{(i-1)} - h_b^{(i-1)}\|_2^2 &= \left\| \frac{1}{K_i} \sum_{k=1}^{K_i} (\tilde{h}_b^{(i-1,k)} - h_b^{(i-1)}) \right\|_2^2 \le \frac{1}{K_i} \sum_{k=1}^{K_i} \|\tilde{h}_b^{(i-1,k)} - h_b^{(i-1)}\|_2^2 \\ &\le \frac{1}{K_i} \sum_{k=1}^{K_i} \|\bar{h}_{bk}^{(i)} - h_{bk}^{(i)}\|_2^2 \le \sum_{k=1}^{K_i} \|\bar{h}_{bk}^{(i)} - h_{bk}^{(i)}\|_2^2 = \|\bar{h}_b^{(i)} - h_b^{(i)}\|_2^2.\end{aligned}$$

In the first equality, we additionally used $h_b^{(i-1,k)} = h_b^{(i-1)}$ for all $k = 1, \dots, K_i$ due to the forward pass. The first inequality is Jensen's inequality. The second inequality follows from (4). The last equality is due to $h_b^{(i)} = (h_{b1}^{(i)}, \dots, h_{bK_i}^{(i)})$. Summing over b and taking square roots yields $\delta^{(i-1)} \le \delta^{(i)}$, as desired.

For the stronger bound, fix b, k and $\tau \in (0, 1]$, and let $h_\tau := (1-\tau)\tilde{h}_b^{(i-1,k)} + \tau h_b^{(i-1,k)}$. Since $(h_\tau, \tilde{\theta}_b^{(i,k)}, f_{ik}(h_\tau, \tilde{\theta}_b^{(i,k)})) \in A_{bik}$, we have

$$\begin{aligned}&\|\tilde{h}_b^{(i-1,k)} - h_b^{(i-1,k)}\|_2^2 + \|\tilde{\theta}_b^{(i,k)} - \theta_b^{(i,k)}\|_2^2 + \|\tilde{h}_{bk}^{(i)} - \bar{h}_{bk}^{(i)}\|_2^2 \\ &\qquad \le \|h_\tau - h_b^{(i-1,k)}\|_2^2 + \|\tilde{\theta}_b^{(i,k)} - \theta_b^{(i,k)}\|_2^2 + \|f_{ik}(h_\tau, \tilde{\theta}_b^{(i,k)}) - \bar{h}_{bk}^{(i)}\|_2^2.\end{aligned}$$

Since $\|h_\tau - h_b^{(i-1,k)}\|_2 = (1-\tau)\|\tilde{h}_b^{(i-1,k)} - h_b^{(i-1,k)}\|_2$ and $\tilde{h}_{bk}^{(i)} = f_{ik}(\tilde{h}_b^{(i-1,k)}, \tilde{\theta}_b^{(i,k)})$, the triangle inequality and Lipschitz continuity yield

$$\begin{aligned}&\|\tilde{h}_b^{(i-1,k)} - h_b^{(i-1,k)}\|_2^2 + \|\tilde{h}_{bk}^{(i)} - \bar{h}_{bk}^{(i)}\|_2^2 \\ &\qquad \le (1-\tau)^2 \|\tilde{h}_b^{(i-1,k)} - h_b^{(i-1,k)}\|_2^2 + \left(\|\tilde{h}_{bk}^{(i)} - \bar{h}_{bk}^{(i)}\|_2 + \tau L_i \|\tilde{h}_b^{(i-1,k)} - h_b^{(i-1,k)}\|_2\right)^2.\end{aligned}$$

Expanding, dividing by τ and letting $\tau \downarrow 0$ yields $\|\tilde{h}_b^{(i-1,k)} - h_b^{(i-1,k)}\|_2 \le L_i \|\tilde{h}_{bk}^{(i)} - \bar{h}_{bk}^{(i)}\|_2$. Invoking (4), we then obtain

$$(1 + L_i^2)\|\tilde{h}_b^{(i-1,k)} - h_b^{(i-1,k)}\|_2^2 \le L_i^2 \|\bar{h}_{bk}^{(i)} - h_{bk}^{(i)}\|_2^2,$$

and the same arguments as before lead us to $\delta^{(i-1)} \le \frac{L_i}{\sqrt{1+L_i^2}} \delta^{(i)} \le c\delta^{(i)}$. The desired result follows. □

Remark 4.2. The Lipschitz continuity assumption is mild. Common activation functions such as ReLU are Lipschitz continuous, while linear layers are Lipschitz in their input for fixed weights. More generally, the theorem only requires a finite Lipschitz bound over the parameter values encountered during the projection step.

We experimented with several heuristics to mitigate vanishing targets, including residual connections (a related heuristic has also been tried in [9]) and explicit amplification of the target signal. While some of these approaches improved performance in specific settings, none consistently closed the gap on deeper networks. For residual connections, this is consistent with our theoretical analysis: adding a skip connection increases the relevant Lipschitz constant by at most one, but we have contraction nonetheless. We leave the development of more effective techniques for mitigating vanishing targets to future work.

5 Experiments

We evaluate $\mathcal{P}$Torch against Torch [47] and PJAX [9] to have a gradient-based and projection-based baseline. Unless explicitly stated otherwise, all benchmarks share the configuration in Table 1.

5.1 Shallow MLP: MNIST and CIFAR10

We first compare an MLP with a single hidden layer of 1024 ReLU units on MNIST [19] ($784 \to 1024 \to 10$) and CIFAR10 [36] ($3072 \to 1024 \to 10$). For this experiment, we used a batch size of 32 and trained the model for 5000 steps. The validation accuracy is shown in Figure 1 and representative timings can be found in Table 5. $\mathcal{P}$Torch utilizes Algorithm 2 with ℓ_2 projection geometry whereas $\mathcal{P}$Torch (ℓ_∞) uses ℓ_∞ projections for the bilinear layers.

We observe that the $\mathcal{P}$Torch variants match or exceed the performance of the gradient-based baseline (Torch with Adam) while maintaining comparable speed to Torch relative to PJAX, thanks to the system integration.

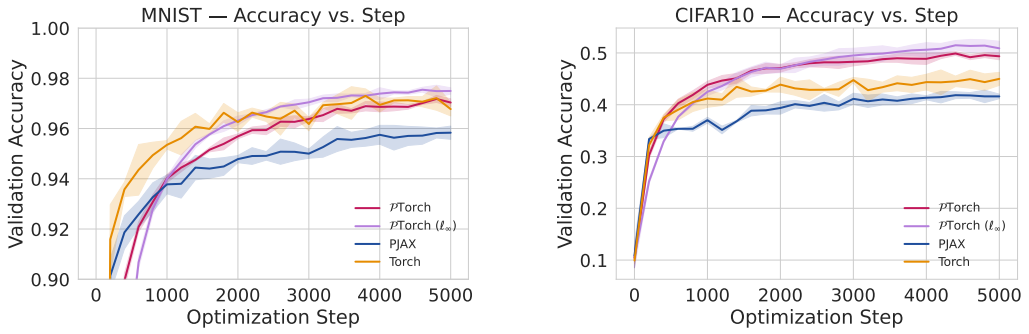

Figure 1: Validation accuracy vs. optimization step for training an MLP with a single hidden layer. Curves show the mean over three seeds $\pm$ 1 standard deviation.

We further train the same MLP in a hybrid geometry. It optimizes the hidden layer in the ℓ_2 norm and the $1024 - 10$ part using the ℓ_∞ projections, for which the result can be found in Table 5. Combining ℓ_2 and ℓ_∞ layers results in lower computational costs, while maintaining or exceeding the accuracy achieved by ℓ_∞ layers. Nevertheless, the interaction between these two layer types is highly experiment-dependent.

5.2 Vanishing target signal and depth

We now present two complementary experiments that highlight the depth barrier predicted by Theorem 4.1. (a) *Exponential attenuation*: We train linear MLPs of depth $N \in \{2, 4, 8\}$ and width 32 to recover the identity map $f(x) = x$ on i.i.d. Gaussian inputs $x \sim \mathcal{N}(0, I_{32})$. Each batch contains 16 samples. We then compute the target signal magnitude $\delta^{(i)}$ at each layer. (b) *Deep MLP:* We train an 8-layer MLP on CIFAR10 using the same hyperparameters as in Section 5.1, but with eight hidden layers of 1024 units instead of one.

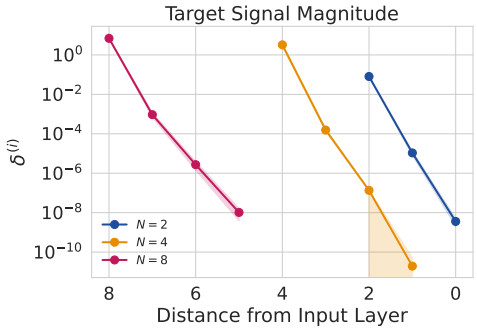

Figure 2: Target signal magnitude $\delta^{(i)}$ for varying depths on the identity recovery task.

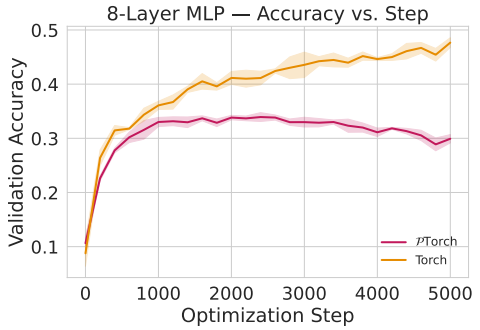

Figure 3: Validation accuracy vs. step for an 8-layer MLP on CIFAR10.

The results are shown in Figure 2 and Figure 3 respectively. For the identity recovery task, it is clear that the target signal magnitude $\delta^{(i)}$ decays exponentially as layer i moves away from the output, with layers three steps from the output already reaching machine precision. Similarly, we have that training an eight-layer MLP with $\mathcal{P}$Torch yields worse results compared to both its gradient-based counterpart and the single-layer results. Both of these observations are consistent with the theory in Section 4.

5.3 Convolutional neural networks in $\mathcal{P}$Torch

To demonstrate the memory benefit of the fused projections we ran 2 CNN experiments. First, we compare the memory usage of PJAX to $\mathcal{P}$Torch as we scale the conv kernel and number of filters in Figure 4. This shows how the fused projections allow us to train vastly larger CNNs as we aren't memory bound as quickly compared to PJAX.

In Figure 5, we train an 8-layer convolutional network on CIFAR10 using an architecture inspired by ResNet-8 [30]. We adapted the network to better suit PBL, namely by flattening the final feature map rather than applying global average pooling and increasing the filter count. Unlike the other benchmarks, we use the Adam optimizer instead of Muon.

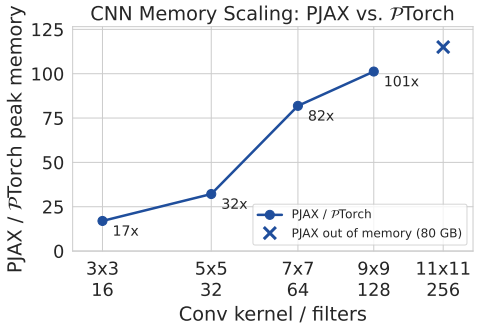

Figure 4: Peak memory usage for CNNs, comparing PJAX to $\mathcal{P}$Torch

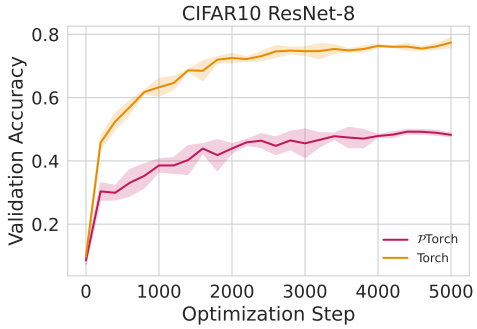

Figure 5: Validation accuracy vs. step for an 8-layer residual CNN on CIFAR10

Figure 5 illustrates the problem with training CNNs in PBL. CNN performance typically improves with increased depth, whereas PBL performance tends to worsen as the network becomes deeper (Section 4). This mismatch makes it very difficult to train a CNN that will outperform a well-tuned MLP.

6 Conclusion and future outlook

In this paper, we developed $\mathcal{P}$Torch, a Torch-native framework for PBL based on cyclic projections. We showed that PBL can be competitive with gradient-based optimization on shallow networks, while reducing memory overhead caused by explicit variable duplication. We also explored the use of standard first-order optimizers as nonlinear relaxation schemes, derived an efficient ℓ_∞ projection for bilinear layers, and identified a fundamental limitation with depth through the vanishing target theorem.

There are still several limitations and open problems. Most importantly, the target signal becomes weaker as it is propagated through deeper networks. Finding a way to mitigate this issue is therefore an important next step. Further, we have deliberately focused on a sequential approach for the best integration with Torch, though parallel extensions are also interesting future work. Our fused projections also do not directly carry over to Douglas–Rachford methods or to the bilinear ℓ_∞ projection, and supporting a wider range of architectures will require new projection operators. It is also interesting to see whether the nonlinear relaxation can be made to work well in the complementary framework PJAX. Some preliminary experiments suggest that this is not trivial and that its effect depends strongly on the projection algorithm with which it is combined.

Acknowledgments

This work was supported by the Research Foundation Flanders (FWO) PhD grant 11A8T26N; Research Council KUL grant C14/24/103. The resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government. LLMs were used to improve the writing and to assist with code development and running experiments. The authors take full responsibility for all content and results. We would also like to thank Andreas Bergmeister for his insightful discussions and valuable perspective on PBL.

References

[1] A. Askari, G. Negiar, R. Sambharya, and L. El Ghaoui. "Lifted neural networks". In: *arXiv preprint arXiv:1805.01532* (2018).

[2] H. Attouch, J. Bolte, and B. F. Svaiter. "Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized Gauss–Seidel methods". In: *Mathematical Programming* 137.1 (2013), pp. 91–129.

[3] A. Auslender. "Méthodes numériques pour la résolution des problèmes d'optimisation avec contraintes". PhD thesis. Université de Grenoble, 1969.

[4] S. Bartunov, A. Santoro, B. Richards, L. Marris, G. E. Hinton, and T. Lillicrap. "Assessing the Scalability of Biologically-Motivated Deep Learning Algorithms and Architectures". In: *Advances in Neural Information Processing Systems*. 2018.

[5] H. H. Bauschke and J. M. Borwein. "On projection algorithms for solving convex feasibility problems". In: *SIAM Review* 38.3 (1996), pp. 367–426.

[6] H. H. Bauschke, P. L. Combettes, and D. R. Luke. "Finding best approximation pairs relative to two closed convex sets in Hilbert spaces". In: *Journal of Approximation Theory* 127.2 (2004), pp. 178–192.

[7] H. H. Bauschke and W. M. Moursi. "On the Douglas–Rachford algorithm for solving possibly inconsistent optimization problems". In: *Mathematics of Operations Research* 49.1 (2024), pp. 58–77.

[8] H. H. Bauschke, M. K. Lal, and X. Wang. "Projecting onto Rectangular Hyperbolic Paraboloids in Hilbert Space". In: *Applied Set-Valued Analysis and Optimization* 5.2 (2023), pp. 163–180.

[9] A. Bergmeister, M. K. Lal, S. Jegelka, and S. Sra. "A Projection-Based Framework for Gradient-Free and Parallel Learning". In: *International Conference on Artificial Intelligence and Statistics*. 2026, pp. 2746–2754.

[10] D. P. Bertsekas. *Nonlinear programming*. 3rd ed. Athena Scientific, 2016.

[11] J. M. Borwein and M. K. Tam. "A cyclic Douglas–Rachford iteration scheme". In: *Journal of Optimization Theory and Applications* 160.1 (2014), pp. 1–29.

[12] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.

[13] J. Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/jax-ml/jax.

[14] L. M. Bregman. "The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming". In: *USSR computational mathematics and mathematical physics* 7.3 (1967), pp. 200–217.

[15] M. Carreira-Perpiñán and W. Wang. "Distributed optimization of deeply nested systems". In: *International Conference on Artificial Intelligence and Statistics*. 2014, pp. 10–19.

[16] A. Cegielski and Y. Censor. "Extrapolation and local acceleration of an iterative process for common fixed point problems". In: *Journal of Mathematical Analysis and Applications* 394.2 (2012), pp. 809–818.

[17] A. Choromanska et al. "Beyond Backprop: Online Alternating Minimization with Auxiliary Variables". In: *International Conference on Machine Learning*. 2019, pp. 1193–1202.

[18] P. L. Combettes. "Inconsistent signal feasibility problems: Least-squares solutions in a product space". In: *IEEE Transactions on Signal Processing* 42.11 (1994), pp. 2955–2966.

[19] L. Deng. "The MNIST database of handwritten digit images for machine learning research [best of the web]". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[20] J. Douglas and H. H. Rachford. "On the numerical solution of heat conduction problems in two and three space variables". In: *Transactions of the American mathematical Society* 82.2 (1956), pp. 421–439.

[21] R. L. Dykstra. "An algorithm for restricted least squares regression". In: *Journal of the American Statistical Association* 78.384 (1983), pp. 837–842.

[22] J. Eckstein and D. P. Bertsekas. "On the Douglas–Rachford splitting method and the proximal point algorithm for maximal monotone operators". In: *Mathematical Programming* 55.1 (1992), pp. 293–318.

[23] V. Elser. "Learning without loss". In: *Fixed Point Theory and Algorithms for Sciences and Engineering* 2021.1 (2021), p. 12.

[24] V. Elser and M. K. Lal. "Learning with Boolean threshold functions". In: *arXiv preprint arXiv:2602.17493* (2026).

[25] A. Fu, J. Zhang, and S. Boyd. "Anderson Accelerated Douglas–Rachford Splitting". In: *SIAM Journal on Scientific Computing* 42.6 (2020), A3560–A3583.

[26] P. Giselsson, M. Fält, and S. Boyd. "Line search for averaged operator iteration". In: *Conference on Decision and Control*. 2016, pp. 1015–1022.

[27] T. Grossman, R. Meir, and E. Domany. "Learning by choice of internal representations". In: *Advances in neural information processing systems*. 1988.

[28] F. Gu, A. Askari, and L. El Ghaoui. "Fenchel Lifted Networks: A Lagrange Relaxation of Neural Network Training". In: *International Conference on Artificial Intelligence and Statistics*. 2020, pp. 3362–3371.

[29] L. Gubin, B. T. Polyak, and E. Raik. "The method of projections for finding the common point of convex sets". In: *USSR Computational Mathematics and Mathematical Physics* 7.6 (1967), pp. 1–24.

[30] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[31] K. He, X. Zhang, S. Ren, and J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *International Conference on Computer Vision*. 2015, pp. 1026–1034.

[32] G. Hinton. "The forward-forward algorithm: Some preliminary investigations". In: *arXiv preprint arXiv:2212.13345* (2022).

[33] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *International Conference on Machine Learning*. 2015, pp. 448–456.

[34] K. Jordan, Y. Jin, V. Boza, J. You, F. Cesista, L. Newhouse, and J. Bernstein. *Muon: An optimizer for hidden layers in neural networks*. 2024. URL: https://kellerjordan.github.io/posts/muon/.

[35] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[36] A. Krizhevsky, G. Hinton, et al. *Learning multiple layers of features from tiny images*. 2009.

[37] M. K. Lal. "Backpropagation from KL Projections: Differential and Exact I-Projection Correspondences". In: *arXiv preprint arXiv:2512.24335* (2025).

[38] M. K. Lal. "Nonconvex Projections Arising in Bilinear Mathematical Models". PhD thesis. University of British Columbia, 2023.

[39] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio. "Difference target propagation". In: *Machine Learning and Knowledge Discovery in Databases*. 2015, pp. 498–515.

[40] G. Li and T. K. Pong. "Douglas–Rachford splitting for nonconvex optimization with application to nonconvex feasibility problems". In: *Mathematical Programming* 159.1 (2016), pp. 371–401.

[41] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. "Random synaptic feedback weights support error backpropagation for deep learning". In: *Nature Communications* 7.1 (2016), p. 13276.

[42] S. B. Lindstrom and B. Sims. "Survey: sixty years of Douglas–Rachford". In: *Journal of the Australian Mathematical Society* 110.3 (2021), pp. 333–370.

[43] S. Malladi, T. Gao, E. Nichani, A. Damian, J. D. Lee, D. Chen, and S. Arora. "Fine-Tuning Language Models with Just Forward Passes". In: *Advances in Neural Information Processing Systems*. 2023, pp. 53038–53075.

[44] R. J. Marks, S. Oh, and L. Atlas. "Alternating projection neural networks". In: *IEEE Transactions on Circuits and Systems* 36.6 (1989), pp. 846–857.

[45] A. Meulemans, F. Carzaniga, J. Suykens, J. Sacramento, and B. F. Grewe. "A Theoretical Framework for Target Propagation". In: *Advances in Neural Information Processing Systems*. 2020, pp. 20024–20036.

[46] N. Parikh and S. Boyd. "Proximal algorithms". In: *Foundations and Trends in Optimization* 1.3 (2014), pp. 127–239.

[47] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. 2019.

[48] G. Pierra. "Decomposition through formalization in a product space". In: *Mathematical Programming* 28.1 (1984), pp. 96–115.

[49] H. Robbins and S. Monro. "A stochastic approximation method". In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407.

[50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[51] B. Scellier and Y. Bengio. "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation". In: *Frontiers in Computational Neuroscience* 11 (2017), p. 24.

[52] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning". In: *arXiv preprint arXiv:1712.06567* (2017).

[53] Y. Tang, Y. Tian, and D. Ha. "EvoJAX: Hardware-accelerated neuroevolution". In: *Genetic and Evolutionary Computation Conference Companion*. 2022, pp. 308–311.

[54] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein. "Training neural networks without gradients: A scalable ADMM approach". In: *International Conference on Machine Learning*. 2016, pp. 2722–2731.

[55] A. Themelis and P. Patrinos. "Douglas–Rachford splitting and ADMM for nonconvex optimization: Tight convergence results". In: *SIAM Journal on Optimization* 30.1 (2020), pp. 149–181.

[56] A. Themelis, L. Stella, and P. Patrinos. "Douglas–Rachford splitting and ADMM for nonconvex optimization: accelerated and Newton-type linesearch algorithms". In: *Computational Optimization and Applications* 82.2 (2022), pp. 395–440.

[57] J. Wang, F. Yu, X. Chen, and L. Zhao. "ADMM for efficient deep learning with global convergence". In: *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2019, pp. 111–119.

[58] X. Wang and M. Benning. "Lifted Bregman training of neural networks". In: *Journal of Machine Learning Research* 24.232 (2023), pp. 1–51.

[59] S. Xie and Z. Li. "Implicit Bias of AdamW: ℓ_∞-Norm Constrained Optimization". In: *Proceedings of the 41st International Conference on Machine Learning*. 2024, pp. 54488–54510.

[60] S.-j. Yeh and H. Stark. "Learning in neural nets using projection methods". In: *Optical Society of America Annual Meeting*. 1990, MN5.

A Additional related work

Auxiliary-variable and proximal methods. The method of auxiliary coordinates [15] introduces auxiliary activations and performs training through penalties and alternating minimization. ADMM-based training [12, 54] similarly separates the layers and enforces consistency through augmented-Lagrangian updates. PBL instead applies local projections onto architectural constraints. The dlADMM algorithm [57] uses a backward-then-forward sweep, quadratic approximations, and backtracking to reduce the complexity of solving the subproblem from $O(n^3)$ to $O(n^2)$, and it is shown that every limit point is a critical point. Related work extends alternating minimization to mini-batches [17] and encodes activation relations through lifted formulations [1, 28, 58].

Target propagation and local learning. Target propagation constructs layer-wise targets using approximate inverses of the forward map [39]. This differs from our approach, where targets arise from corrections to architectural constraints. Target propagation has been related to approximate Gauss–Newton updates [45], while its scalability has been examined empirically in [4]. Feedback alignment uses random feedback weights [41], whereas equilibrium propagation compares free and nudged equilibria [51], and Forward-Forward trains local goodness objectives on positive and negative examples [32]. A KL-projection construction relaxes message passing on a lifted computation graph to backpropagation at a fixed forward evaluation [37].

Gradient-free learning. There are many works utilizing gradient-free methods for learning. We describe a few of them. Neuroevolution searches directly over network parameters through population-based optimization [52]. EvoJAX [53] provides a JAX-based library for running evolutionary algorithms. MeZO [43] instead uses perturbed loss evaluations for language-model fine-tuning, with memory requirements comparable to inference. These methods require no gradients, but use objective evaluations only. Similarly, PBL requires no global gradients but only local projection operators.

Nonlinear relaxation and acceleration. Extrapolation and residual line searches adapt stepsizes using projection displacements or fixed-point residuals [16, 26]. Anderson acceleration combines iterate histories with safeguards in Douglas–Rachford splitting [25], while nonconvex Douglas–Rachford and ADMM variants incorporate quasi-Newton directions through envelope-based criteria [56].

B Optimization perspectives on $\mathcal{P}$Torch

B.1 Connection to incremental gradient descent

To motivate the nonlinear relaxations in Section 3.3, we make explicit the connection between cyclic projections in the Euclidean geometry and incremental gradient descent [10]. Recall from Section 2.2 that training is tightly connected to minimizing the finite-sum problem $F(\omega)$ defined in (2). This objective has the form $F(\omega) = \sum_{j=1}^{M} f_j(\omega)$, with each component $f_j(\omega) = \frac{1}{2}\operatorname{dist}^2(S_j, \omega)$ a squared distance to some general constraint set S_j. At every point where f_j is differentiable, the Euclidean projection is unique and $\nabla f_j(\omega) = \omega - \Pi_{S_j}(\omega)$. Since the squared distance to a closed set is locally Lipschitz, it is differentiable almost everywhere. Thus, although the neural-network constraint sets are generally nonconvex, this relation holds almost everywhere. This is analogous to commonly used nonsmooth activation functions such as ReLU, which are also differentiable almost everywhere.

With this gradient, a cyclic projection step can be interpreted directly as an incremental gradient update. Consider a relaxed projection onto S_{j_k},

$$\omega^{k+1} = (1 - \alpha^k)\omega^k + \alpha^k \Pi_{S_{j_k}}(\omega^k),$$

where $\alpha^k > 0$ is the relaxation parameter. Rearranging and using the relation above gives

$$\omega^{k+1} = \omega^k - \alpha^k\big(\omega^k - \Pi_{S_{j_k}}(\omega^k)\big) = \omega^k - \alpha^k \nabla f_{j_k}(\omega^k).$$

This is precisely an incremental gradient step on F [10, Eq. (2.78)], where the index j_k follows the cyclic ordering of the constraint sets from the backward traversal. Ordinary cyclic projections correspond to the special case $\alpha^k = 1$.

This interpretation motivates the nonlinear relaxation introduced in Section 3.3. In the Euclidean setting, the projection residual $\omega^k - \Pi_{S_{j_k}}(\omega^k)$ is the gradient of the corresponding local squared-distance term wherever that term is differentiable. $\mathcal{P}$Torch passes this residual through a first-order optimizer such as Adam or Muon. This no longer yields an exact incremental gradient method on F, but can be viewed as an optimizer-based nonlinear relaxation of the underlying projection step.

ℓ_∞ and hybrid geometries. Regarding the ℓ_∞ and hybrid projection geometries, the identity between the projection residual and the gradient of a squared-distance (in the other geometry) does not hold in general. Of course, $\Pi_{S_j}(\omega) - \omega$ is still a descent direction for the distance to S_j in the corresponding geometry. Thus, cyclic projections could be viewed as an incremental descent method. This interpretation does not extend to residuals that are transformed by Adam or Muon, which we see as heuristics.

B.2 Connection to the Moreau envelope

Using a proximal loss results in better empirical performance, as shown by our experiments and [9]. The output proximal step in Algorithm 2 replaces the hard assignment $h^{(N)} \leftarrow y$ by a softer update that is a compromise between reducing the loss while remaining close to the current output. Following [9], for $\lambda > 0$ we define the proximal target for some loss $\mathcal{L}(\cdot, \cdot)$ by

$$\bar{h}^{(N)} = \operatorname{prox}_{\lambda \mathcal{L}(\cdot, y)}(h^{(N)}) \coloneqq \arg\min_u \left\{ \mathcal{L}(u, y) + \frac{1}{2\lambda} \|u - h^{(N)}\|_2^2 \right\}. \tag{5}$$

Throughout this subsection, $\mathcal{L}(\cdot, y)$ is proper, lower semicontinuous and convex, so the prox exists and is unique. If $\mathcal{L}(\cdot, y)$ is the indicator function of $\{y\}$, i.e., 0 at y and $+\infty$ elsewhere, then (5) recovers the hard assignment on line 3 of Algorithm 1.

The minimum value in (5) determines the Moreau envelope

$$\mathcal{L}_\lambda(h^{(N)}, y) \coloneqq \min_u \left\{ \mathcal{L}(u, y) + \frac{1}{2\lambda} \|u - h^{(N)}\|_2^2 \right\}.$$

It is continuously differentiable even when the loss is not, with gradient [46, Eq. (3.2)]

$$\nabla_{h^{(N)}} \mathcal{L}_\lambda(h^{(N)}, y) = \frac{h^{(N)} - \bar{h}^{(N)}}{\lambda} \implies \bar{h}^{(N)} = h^{(N)} - \lambda \nabla_{h^{(N)}} \mathcal{L}_\lambda(h^{(N)}, y).$$

Thus, the proximal update is a gradient step with unit stepsize on $\lambda \mathcal{L}_\lambda$. This furnishes the connection to incremental gradient descent. Indeed, let $\tilde{F}$ denote F from (2) without its B_{out} term, and consider the surrogate

$$\widetilde{F}_\lambda(\omega) \coloneqq \tilde{F}(\omega) + \lambda \sum_{b=1}^{B} \mathcal{L}_\lambda(h_b^{(N)}, y_b).$$

Then, the updates underlying Algorithm 2 without the nonlinear relaxation correspond exactly to incremental gradient descent on $\widetilde{F}_\lambda$ with the cyclic order corresponding to the computational graph.

Further, we recall that the minimizers of $\mathcal{L}_\lambda(\cdot, y)$ equal those of $\mathcal{L}(\cdot, y)$, which also equal the fixed points of the proximal operator. This agrees with the fixed point motivation in [9]. If each loss is minimized exactly at y_b and the original feasibility problem has a solution, then F and $\widetilde{F}_\lambda$ also share their global minimizers. However, this need not hold when the constraints are inconsistent. Moreover, it is not guaranteed a priori that a minimizer exists. For example, one-hot cross-entropy on unconstrained logits has no finite minimizer.

For the squared loss $\mathcal{L}(u, y) = \frac{1}{2}\|u - y\|_2^2$, direct minimization gives

$$\bar{h}^{(N)} = \frac{h^{(N)} + \lambda y}{1 + \lambda}, \qquad h^{(N)} - \bar{h}^{(N)} = \frac{\lambda}{1 + \lambda}(h^{(N)} - y).$$

The displacement then becomes a scaled version of the hard assignment displacement: smaller λ gives a smaller correction, while $\lambda \to \infty$ recovers the hard assignment. More generally, if $\mathcal{L}(\cdot, y)$ is G-Lipschitz, proximal optimality gives $\|h^{(N)} - \bar{h}^{(N)}\|_2 \le \lambda G$. Thus, the parameter λ can control the magnitude of the signal initiating the backward pass.

C A backward pass through a small network

This section illustrates one backward pass of Algorithm 2 for an MLP with two linear layers and an elementwise activation, as shown in Figure 6. For each sample $b = 1, \dots, B$, the forward pass is

$$h_b^{(0)} = x_b, \qquad h_b^{(1)} = W^{(1)} h_b^{(0)}, \qquad h_b^{(2)} = \Sigma(h_b^{(1)}), \qquad h_b^{(3)} = W^{(3)} h_b^{(2)}.$$

In the notation of (1), we have $N = 3$, $\theta^{(1)} = W^{(1)}$ and $\theta^{(3)} = W^{(3)}$, while the activation has no learnable parameters, cf. Example 2.1. To distinguish the forward and backward quantities, we write $h_b^{(i)}$ for the forward-pass value, $\bar{h}_b^{(i)}$ for the target propagated from the following layer, and $\tilde{h}_b^{(i)}$ for the value left after projecting layer i. Algorithm 2 performs these updates in place.

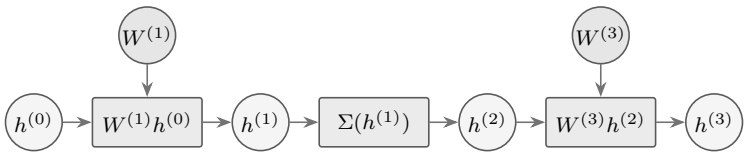

Figure 6: Two-layer MLP.

Output target. The backward pass starts with the proximal output step

$$\bar{h}_b^{(3)} = \mathrm{prox}_{\lambda \mathcal{L}(\cdot, y_b)}(h_b^{(3)}), \qquad b = 1, \dots, B.$$

This is visualized in Figure 7.

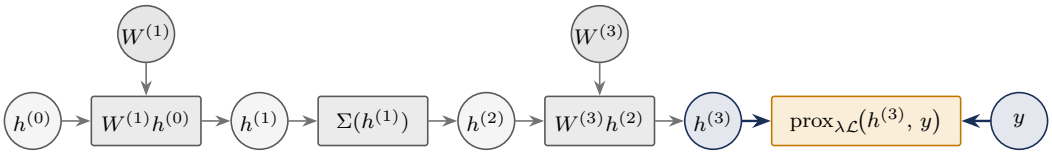

Figure 7: Proximal update at the output node.

Output linear layer. Let $(w_k^{(3)})^\top$ denote the kth row of $W^{(3)}$, and let K_3 be its number of rows. As in Example 2.2, the associated constraints are

$$A_{b3k} = \{\omega : h_{bk}^{(3)} = (w_b^{(3,k)})^\top h_b^{(2,k)}\},$$

where $h_b^{(2,k)}$ and $w_b^{(3,k)}$ are copies of the input and weight row, respectively. Starting from their common values from the forward pass, the local projections give

$$(\tilde{h}_b^{(2,k)}, \tilde{w}_b^{(3,k)}, \tilde{h}_{bk}^{(3)}) \in \Pi_{A_{b3k}}(h_b^{(2)}, w_k^{(3)}, \bar{h}_{bk}^{(3)}), \qquad k = 1, \dots, K_3.$$

The consensus step averages input copies across output rows and weight copies across the minibatch:

$$\bar{h}_b^{(2)} = \frac{1}{K_3} \sum_{k=1}^{K_3} \tilde{h}_b^{(2,k)}, \qquad \bar{w}_k^{(3)} = \frac{1}{B} \sum_{b=1}^{B} \tilde{w}_b^{(3,k)}.$$

Stacking the rows $(\bar{w}_k^{(3)})^\top$ yields $\overline{W}^{(3)}$. As in Section 3.2, the fused projection combines these two steps without explicitly materializing the copies. The variables entering this step are shown in Figure 8.

The hidden states are updated directly to $\bar{h}^{(2)}$ and $\tilde{h}^{(3)}$, where $\tilde{h}_b^{(3)} = (\tilde{h}_{bk}^{(3)})_{k=1}^{K_3}$. In contrast, nonlinear relaxation is applied to the weights:

$$W^{(3)} \leftarrow \mathrm{Optimizer}(W^{(3)}, W^{(3)} - \overline{W}^{(3)}).$$

The target for the activation layer becomes $\bar{h}^{(2)}$.

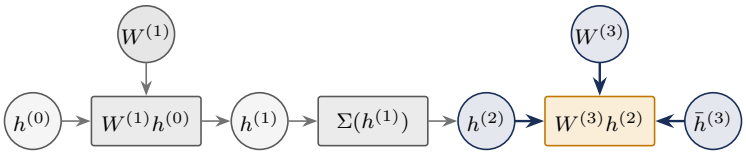

Figure 8: Output-layer projection. The fused projection applies the architectural projection followed by the corresponding consensus projection.

Activation layer. The next projection acts on the graph of the elementwise activation, that is, $A_{b2} = \{(u, v) : v = \Sigma(u)\}$ for each sample. The involved variables are shown in Figure 9. Using the target from the output linear layer gives

$$(\bar{h}_b^{(1)}, \tilde{h}_b^{(2)}) \in \Pi_{A_{b2}}(h_b^{(1)}, \bar{h}_b^{(2)}).$$

Since the activation has no parameters and acts elementwise, the projection separates across coordinates and requires neither variable duplication nor consensus. Both hidden states are updated directly, and $\bar{h}_b^{(1)}$ is passed to the first linear layer.

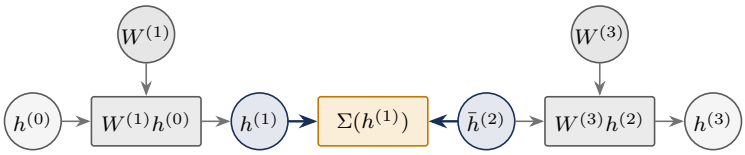

Figure 9: Activation-layer projection. The incoming target $\bar{h}^{(2)}$ and the forward-pass state $h^{(1)}$ are projected onto the activation constraint, yielding the target $\bar{h}^{(1)}$ for the first linear layer.

Input linear layer and input constraint. The first linear layer uses the same fused projection and consensus operations as the output layer, now with the target $\bar{h}^{(1)}$:

$$\big(\bar{h}^{(0)}, \overline{W}^{(1)}, \tilde{h}^{(1)}\big) \leftarrow \mathrm{FusedProj}_1\big(h^{(0)}, W^{(1)}, \bar{h}^{(1)}\big). \tag{6}$$

The hidden states are again updated directly, while the parameter update is

$$W^{(1)} \leftarrow \mathrm{Optimizer}(W^{(1)},\, W^{(1)} - \overline{W}^{(1)}).$$

Finally, the input batch constraint is reimposed via a hard assignment, $h_b^{(0)} \leftarrow x_b$ for $b = 1, \ldots, B$, completing one backward sweep.

D Implementation details and additional experiments

To make PBL more accessible to future researchers, this section discusses implementation and tuning details that help ensure successful experiments. To make all experiments reproducible, the default hyperparameters are provided in Table 1.

Table 1: Default experimental protocol.

Component	Default value
Random seed	2026 (per-run seed $= 2026 + r$ for $r = 0, 1, 2$)
Batch size	32
Train/val/test split	Official train set split 90/10 into train/val; official test set for final evaluation
$\mathcal{P}$Torch norm, optimizer	ℓ_2; `ProjectionMuon`, lr $= 10^{-3}$
$\mathcal{P}$Torch loss	`CrossEntropy` (proximal, $\lambda = 10$)
PJAX optimizer	`DouglasRachford`, 50 inner projection steps per batch (matches [9])
Torch optimizer	`Adam`, lr $= 10^{-3}$, β_1=0.9, β_2=0.999
Torch loss	`CrossEntropyLoss`
Hardware	1× NVIDIA A100 (80GB)

D.1 Architecture and Initialization

When designing architectures in this framework we recommend shallow networks as they are both more stable and empirically superior to deeper ones. Make sure that every layer is implemented using a $\mathcal{P}$Torch module. The use of standard Torch layers (e.g., standard linear layers) will interfere with the autograd-based target propagation mechanism, causing targets to be interpreted as gradients, or vice versa.

Initialization is important. The framework uses the same methods as in gradient-based training. Our experiments have demonstrated that initialization has a massive effect on both convergence and the final state of the network. This is an under-researched area, and considerable benefit could be gained from deriving a projection-specific initialization scheme.

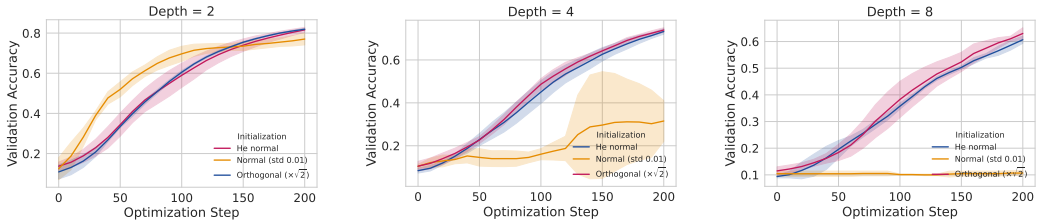

Figure 10: Impact of the weight initialization on convergence across depths. Validation accuracy over the first 200 steps of $\mathcal{P}$Torch (ℓ_2) on MNIST with the settings of Figure 1. We compare normal, He normal and orthogonal initialization. All three initializations converge at depth 2, the normal initialization diverges as depth increases and stays at random level for depth 8, whereas He normal and orthogonal remain stable.

D.2 Advanced Relaxations and Loss Formulation

Nonlinear relaxation can be applied to the projection residuals. Applying it only to the weight residuals usually performs better than applying it to both weight and hidden-state residuals. However, in some cases, such as the XOR benchmark (Figure 11), where a network is trained on the four 2-bit XOR points with a hard-margin output across 5 random initializations and 1000 backward passes, applying relaxation more broadly helps.

Finally, introducing a proximal projection (Appendix B.2) for the loss accelerates convergence in certain circumstances. This has been reported in prior projection-based literature like PJAX [9]. Figure 12 is a variant of Figure 1, showing the effect of the loss formulation.

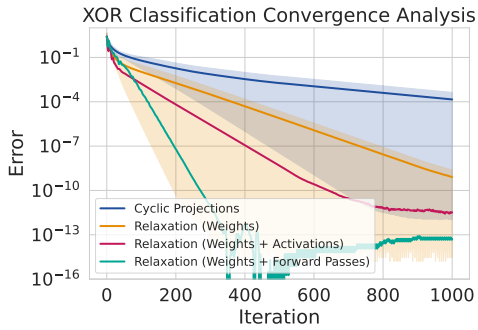

Figure 11: First-order relaxation applied to the XOR benchmark.

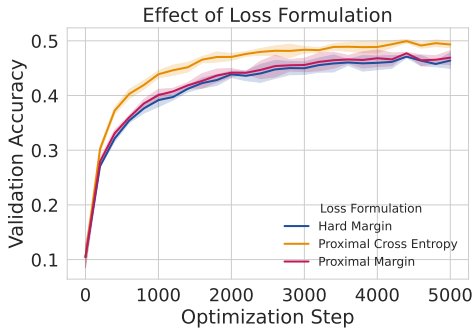

Figure 12: The effect of various loss assignments on the validation accuracy.

D.3 Projection Mechanics

The choice of projection norm alters the training dynamics. Table 3 indicates that the ℓ_∞ projection is computationally more expensive but achieves higher final accuracy for 1-hidden layer networks. The performance of the ℓ_∞ norm drops significantly faster than that of the ℓ_2 norm when increasing depth. Thus, when experimenting with deep architectures, the ℓ_2 norm is recommended.

The projection can be further tuned via weighted projection parameters. In practice, the framework is robust but careful tuning can yield modest gains.

In the cyclic projections algorithm, we can perform multiple backward passes per batch, as in [9]. In the current implementation, the number of backward passes per batch is fixed at 1. This allows more efficient and elegant code. Table 2 shows the performance of $\mathcal{P}$Torch under equal steps or time on the MNIST benchmark for multiple backward passes. We observe that 1 backward pass is a competitive choice for our specific implementation.

Table 2: Test accuracy of $\mathcal{P}$Torch as a function of the number of backward passes P.

P	Test Acc (equal-compute, 3000 steps)	Test Acc (equal-time 0.3 s)
1	0.9229 ± 0.0013	0.8435 ± 0.0050
2	0.9237 ± 0.0007	0.8411 ± 0.0063
10	0.9158 ± 0.0010	0.8273 ± 0.0096

D.4 Memory scaling and optimizer choice

Choosing the optimizer is one of the highest-leverage decisions. In most settings, the Muon optimizer yields the strongest results and is our recommended default. Adam is promising but substantially less stable and often requires learning-rate tuning to train reliably. When using SGD or Adadelta, higher learning rates (sometimes even larger than 10) are beneficial. By contrast, Muon and Adam generally perform best at their standard learning rates of 10^{-3}.

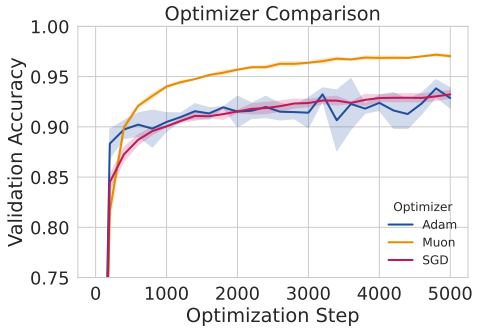

Figure 13: Validation accuracy on MNIST versus optimization step for Adam, Muon and SGD.

Empirically, both $\mathcal{P}$Torch ℓ_2 and Torch scale better and have lower memory consumption than PJAX, allowing larger networks to be trained before memory is exhausted. As shown in the MNIST benchmark in Figure 14, for a single linear-layer projection, PJAX and $\mathcal{P}$Torch ℓ_∞ scales as $O(N_{\text{in}}N_{\text{out}}B)$, whereas $\mathcal{P}$Torch ℓ_2 reduces this to $\mathcal{O}(N_{\text{in}}B + N_{\text{out}}B + N_{\text{in}}N_{\text{out}})$, matching Torch's asymptotic scaling. A detailed ablation of the memory usage during training can be found in Table 3.

D.5 Unified ablation

To properly attribute performance gains, we perform an ablation study covering both the engineering and the mathematical contributions. The setup follows Figure 1, except that we use a batch size of 256 and train for 3000 steps. $\mathcal{P}$Torch relies on precompiled versions of selected operations cached on disk. Without this cache, initializing a $\mathcal{P}$Torch layer incurs a one-time compilation cost of 20.8 ± 1.1 seconds for the Table 3 R2 setup. PJAX uses the XLA JIT compiler of JAX, which adds a startup time

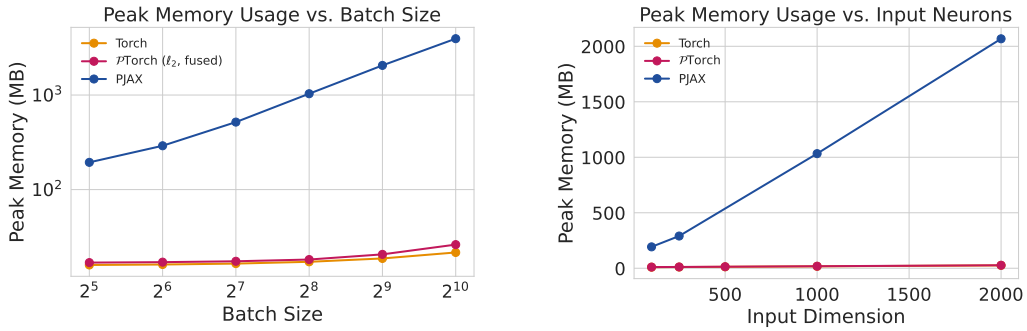

Figure 14: Peak memory of a single linear-layer projection. PJAX scale as $\mathcal{O}(N_{\text{in}}N_{\text{out}}B)$. The fused ℓ_2 projection scales as $\mathcal{O}(N_{\text{in}}B + N_{\text{out}}B + N_{\text{in}}N_{\text{out}})$ and closely tracks Torch.

Table 3: Ablation study of $\mathcal{P}$Torch on CIFAR10.

Method	Test acc.	ms/step	Peak MB	Wall (s)
Torch (Adam)	0.5172 ± 0.0041	0.97 ± 0.01	75.3	2.9
PJAX (DR, 50 passes/step)	0.4659 ± 0.0030	824.81 ± 3.94	6,260.2	2,474.4
R1 ($\mathcal{P}$Torch ℓ_2)	0.2934 ± 0.0035	1.47 ± 0.02	91.3	4.4
R1 + proximal CE loss	0.4408 ± 0.0024	1.32 ± 0.04	91.3	4.0
R2 + Muon	0.5044 ± 0.0031	2.28 ± 0.06	103.3	6.9
R3 + ℓ_∞ geometry	0.5174 ± 0.0022	14.44 ± 0.07	4,966.9	43.3
R4 - Muon	0.1611 ± 0.0046	13.70 ± 0.03	4,954.8	41.1
$\mathcal{P}$Torch-hybrid	0.5372 ± 0.0041	2.69 ± 0.02	254.2	8.1

of 3.71±0.19 seconds to every run. It should be noted that this linear ablation does not accurately predict the performance gains achievable by combining arbitrary elements. For example, employing the ℓ_∞ geometry without the nonlinear relaxation yields an accuracy of only 16.11±0.46%.

D.6 Robustness

The robustness to label noise was tested by replacing $x\%$ of the training labels with random labels. We used the ℓ_2 norm and the hybrid variant, averaged over 10 seeds for robustness, while keeping the rest of the setup identical to Figure 1

Table 4: Robustness to label noise for the studied methods

Dataset	Algorithm	0%	10%	20%
MNIST	Torch	0.976 ± 0.001	0.969 ± 0.002	0.963 ± 0.002
MNIST	$\mathcal{P}$Torch ℓ_2 (Muon)	0.973 ± 0.001	0.966 ± 0.002	0.961 ± 0.002
MNIST	$\mathcal{P}$Torch hybrid (Muon)	0.980 ± 0.001	0.966 ± 0.002	0.955 ± 0.003
CIFAR10	Torch	0.461 ± 0.006	0.439 ± 0.009	0.420 ± 0.007
CIFAR10	$\mathcal{P}$Torch ℓ_2 (Muon)	0.490 ± 0.003	0.476 ± 0.004	0.460 ± 0.005
CIFAR10	$\mathcal{P}$Torch hybrid (Muon)	0.526 ± 0.004	0.496 ± 0.003	0.465 ± 0.004

On MNIST, $\mathcal{P}$Torch ℓ_2 is most robust as noise increases, but Torch retains the best overall performance. On CIFAR10, the ℓ_2 and hybrid methods consistently achieve higher accuracies at every noise level.

D.7 Test set results

The following section contains the test set results of the experiments conducted in the main text. Each test accuracy is computed with the checkpoint that reached the highest validation accuracy during training. Bold marks the best test accuracy within each architecture and dataset.

Table 5: Test set results for the shallow MLP (Figure 1), the deep MLP (Figure 3) and the CNN (Figure 5).

Dataset	Method	Test acc (%)	ms/step	Train time (s)
Shallow MLP (Figure 1)				
MNIST	$\mathcal{P}$Torch (ℓ_2)	97.33 ± 0.05	2.10	10.5
	$\mathcal{P}$Torch (ℓ_∞)	97.76 ± 0.08	2.60	13.0
	$\mathcal{P}$Torch hybrid	$\mathbf{97.89 \pm 0.08}$	2.49	12.5
	PJAX	96.07 ± 0.21	35.31	176.5
	Torch	97.66 ± 0.10	0.89	4.5
CIFAR10	$\mathcal{P}$Torch (ℓ_2)	49.01 ± 0.33	2.17	10.8
	$\mathcal{P}$Torch (ℓ_∞)	50.65 ± 0.16	3.64	18.2
	$\mathcal{P}$Torch hybrid	$\mathbf{52.65 \pm 0.33}$	2.76	13.8
	PJAX	42.13 ± 0.85	112.98	564.9
	Torch	45.65 ± 0.31	0.92	4.6
Deep MLP (Figure 3)				
CIFAR10	$\mathcal{P}$Torch (ℓ_2)	34.10 ± 0.44	7.98	39.9
	Torch	$\mathbf{47.16 \pm 0.52}$	2.49	12.4
CNN (Figure 5)				
CIFAR10	$\mathcal{P}$Torch	48.83 ± 0.38	6.79	33.9
	Torch	$\mathbf{76.70 \pm 1.12}$	4.28	21.4

E Useful Projections

In this section, we describe some useful projections for common neural network layers. In particular, we develop a novel solver for projecting onto the bilinear constraint set in the ℓ_∞ geometry. For additional useful projections, see [23, 9].

E.1 ReLU Layer

The ReLU constraint is $C_{\mathrm{ReLU}} = \{(x, z) : z = \max(0, x)\}$, which is the union of the rays $\{(x, 0) : x \leq 0\}$ and $\{(x, x) : x \geq 0\}$. The projection is found by projecting onto both rays and returning the closer candidate. Writing $x_1 = \min(x, 0)$ and $x_2 = \max\{(x + z)/2, 0\}$, these candidates are $(x_1, 0)$ and (x_2, x_2). Hence

$$(\bar{x}, \bar{z}) = \begin{cases} (x_1, 0) & d_1 < d_2, \\ (x_2, x_2) & \text{otherwise}, \end{cases}$$

where for ℓ_2 we have $d_1^2 = (x - x_1)^2 + z^2$ and $d_2^2 = (x - x_2)^2 + (z - x_2)^2$ and for ℓ_∞ we have $d_1 = \max\{|x - x_1|, |z|\}$ and $d_2 = \max\{|x - x_2|, |z - x_2|\}$. For vector-valued ReLU, the projection is applied coordinatewise. For ℓ_2, this projection has also been considered in [23, 9].

E.2 Linear layer: ℓ_2 projection

The linear layer constraint is the set $C_{\mathrm{dot}} = \{(h, w, h^+) : w^\top h = h^+\}$. The Euclidean projection has been studied in [23, 8] and we refer to [8] for global optimality. The projection problem is

$$\underset{(\tilde{h}, \tilde{w}, \tilde{h}^+) \in C_{\mathrm{dot}}}{\text{minimize}} \ \frac{1}{2}\|\tilde{h} - h\|_2^2 + \frac{1}{2}\|\tilde{w} - w\|_2^2 + \frac{1}{2}(\tilde{h}^+ - h^+)^2.$$

Let t be a Lagrange multiplier for the equality constraint $\tilde{h}^+ - \tilde{w}^\top \tilde{h} = 0$. The first-order optimality conditions then give $\tilde{h} = h + t\tilde{w}$, $\tilde{w} = w + t\tilde{h}$, and $\tilde{h}^+ = h^+ - t$. For $|t| < 1$, solving the first two equations yields

$$\tilde{h} = \frac{h + tw}{1 - t^2}, \qquad \tilde{w} = \frac{w + th}{1 - t^2}, \qquad \tilde{h}^+ = h^+ - t.$$

It remains to choose t such that the constraint is satisfied. Defining $p = w^\top h$ and $q = \|h\|_2^2 + \|w\|_2^2$, substitution gives the scalar equation

$$h^+ - t = \frac{p(1 + t^2) + qt}{(1 - t^2)^2}.$$

In the generic case, the root corresponding to the Euclidean projection lies in $(-1, 1)$ and can be computed efficiently with Newton's method. Then, writing $u = (1 - t^2)^{-1}$ and $v = t(1 - t^2)^{-1}$, we obtain $\tilde{h} = uh + vw$ and $\tilde{w} = uw + vh$.

E.3 Linear layer: ℓ_∞ projection

We derive the projection in (3) by finding the smallest $\varepsilon \geq 0$ for which there exists a feasible point satisfying $\|\tilde{h} - h\|_\infty \leq \varepsilon$, $\|\tilde{w} - w\|_\infty \leq \varepsilon$, and $|\tilde{h}^+ - h^+| \leq \varepsilon$. Let $D = w^\top h$. If $D = h^+$, the point is already feasible. Now suppose that $D < h^+$. Our goal is to find the largest increase in the dot product possible for a given ε. The case $D > h^+$ follows from replacing (h, w, h^+) with $(h, -w, -h^+)$

Writing $\Delta h = \tilde{h} - h$ and $\Delta w = \tilde{w} - w$, the contribution of the kth coordinate to D is given by $\Delta D_k = h_k \Delta w_k + w_k \Delta h_k + \Delta h_k \Delta w_k$. This expression attains its maximum over $[-\varepsilon, \varepsilon] \times [-\varepsilon, \varepsilon]$ at a corner, as illustrated in Figure 15. Defining $a_k = |h_k + w_k|$ and $b_k = |h_k - w_k|$, this maximum is $M_k(\varepsilon) = \max\{a_k\varepsilon + \varepsilon^2, b_k\varepsilon - \varepsilon^2\}$. Hence, the largest dot product attainable within radius ε is $D + \sum_k M_k(\varepsilon)$, while the smallest allowed output is $h^+ - \varepsilon$. Therefore, the smallest $\varepsilon^\star$ is a root of

$$G(\varepsilon) = D + \sum_{k=1}^{N_{\text{in}}} M_k(\varepsilon) - (h^+ - \varepsilon) = 0.$$

Each M_k is continuous and nondecreasing. Hence, G is continuous and strictly increasing, with $G(0) = D - h^+ < 0$ and $G(h^+ - D) > 0$. It follows that G has a unique positive root $\varepsilon^\star$.

Δh_k	Δw_k	ΔD_k
$+\epsilon$	$+\epsilon$	$\epsilon(h_k + w_k) + \epsilon^2$
$-\epsilon$	$-\epsilon$	$-\epsilon(h_k + w_k) + \epsilon^2$
$+\epsilon$	$-\epsilon$	$\epsilon(w_k - h_k) - \epsilon^2$
$-\epsilon$	$+\epsilon$	$\epsilon(h_k - w_k) - \epsilon^2$

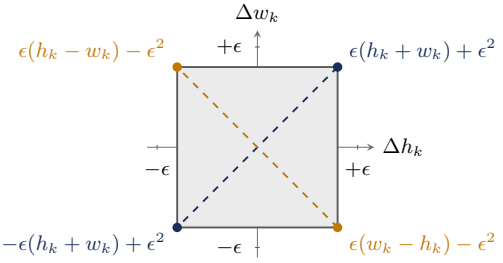

Figure 15: The coordinatewise change ΔD_k at the four corners of the perturbation box. Maximizing over the two diagonal pairs gives the two branches of M_k.

Each M_k switches branches at most once, at $\varepsilon = (b_k - a_k)/2$ when $b_k > a_k$. A direct approach sorts these switching points, and then runs through all the possibilities. This method has a computational cost of $\mathcal{O}(N_{\text{in}} \log N_{\text{in}})$ due to the sorting. We avoid sorting by selecting the active branch directly at each iterate. Define $m_k(\varepsilon)$ to be 1 if $2\varepsilon \geq b_k - a_k$ and 0 otherwise, so that

$$M_k(\varepsilon) = m_k(\varepsilon)(a_k\varepsilon + \varepsilon^2) + (1 - m_k(\varepsilon))(b_k\varepsilon - \varepsilon^2).$$

On any interval where the values of $m_k(\varepsilon)$ are fixed, G is differentiable, with

$$G'(\varepsilon) = 1 + \sum_{k=1}^{N_{\text{in}}} \left[m_k(\varepsilon)(a_k + 2\varepsilon) + (1 - m_k(\varepsilon))(b_k - 2\varepsilon) \right].$$

Thus, both G and G' can be evaluated in $\mathcal{O}(N_{\mathrm{in}})$ time. We solve $G(\varepsilon) = 0$ using safeguarded Newton iterations. The root is initially bracketed by $[0, h^+ - D]$. At each iteration, we update this bracket according to the sign of G and we take a Newton step if it remains inside the current bracket. Otherwise, or if the iterate coincides with a branch-switching point, we take a bisection step. Since G is continuous and strictly increasing, the bracket always contains the unique root, guaranteeing convergence. For T iterations, the total computational cost is $\mathcal{O}(TN_{\mathrm{in}})$.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction reflect the paper's contribution.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of the method in detail and highlight the areas for improvement.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We provide correct proofs for our theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide the information needed to reproduce the main experimental results of the paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide the code needed for the experiments.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide details on the experiments we performed, including the problem formulations, datasets and the algorithms we compared.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We provide details on the experiments we performed, including the problem formulations, datasets and the algorithms we compared.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification:We provide details on the compute resources required to perform our experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: Our research conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: : This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We provide the credits for the assets we used.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The code is documented and provided.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.