# Closed-Form Last Layer Optimization

**Alexandre Galashov**[*]
Google DeepMind
Gatsby Unit, University College London
agalashov@google.com

**Nathaël Da Costa**[*]
Tübingen AI Center, University of Tübingen
nathael.da-costa@uni-tuebingen.de

**Liyuan Xu**
Secondmind[†]
liyuan9988@gmail.com

**Philipp Hennig**
Tübingen AI Center, University of Tübingen
philipp.hennig@uni-tuebingen.de

**Arthur Gretton**
Google DeepMind
Gatsby Unit, University College London
gretton@google.com

## Abstract

Neural networks are typically optimized with variants of stochastic gradient descent. Under a squared loss, however, the optimal solution to the linear last layer weights is known in closed-form. We propose to leverage this during optimization, treating the last layer as a function of the backbone parameters, and optimizing solely for these parameters. We show this is equivalent to alternating between gradient descent steps on the backbone and closed-form updates on the last layer. We adapt the method for the setting of stochastic gradient descent, by trading off the loss on the current batch against the accumulated information from previous batches. We provide theoretical analyses showing convergence of the method to an optimal solution in the neural tangent kernel regime, as well as quantifying the gains compared to standard SGD in a one-step analysis. Finally, we demonstrate the effectiveness of our approach compared with SGD and Adam on a squared loss in several regression tasks, including neural operators and causal inference.

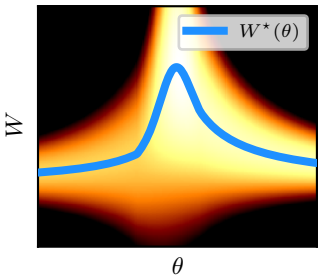


Figure 1: The squared loss landscape of a two-parameter neural network

$$f(x) = W\,\mathrm{ReLU}(\theta x)$$

with three random training data points. Dark / light regions correspond to values of high / low loss respectively. We plot in blue the optimal last layer parameter $W^\star(\theta)$ as a function of the backbone parameter $\theta$. We propose to optimize along the blue curve, rather than in two-dimensional space.

[*]Equal contribution.
[†]Work done while at University College London

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

# 1 Introduction

Training deep neural networks is almost always done with variants of stochastic gradient descent (SGD). Despite their empirical success, these iterative methods treat every layer of the network in the same way. However, the linear last layer often admits a much simpler – and in the case of squared loss, closed-form – optimal solution. This mismatch suggests an opportunity: if the optimal last layer weights can be computed directly given the current features produced by the backbone, we can regard the last layer as an implicit function of the backbone parameters. This could simplify the optimization problem by constraining the last layer to be optimal throughout (see Fig. 1).

In SGD, gradients at each step are computed in minibatches. Because of computational constraints, the closed-form solution of the last layer should also use minibatches. This risks overfitting the last layer to each batch at each optimization step. To correct for this issue, there is the need to account for previous last layer solutions.

In this paper, we develop a training procedure that can perform optimization with a closed-form optimal last layer through SGD on the backbone parameters. Our contributions are as follows:

1. We propose leveraging the closed-form last layer solution for squared loss, and optimizing the backbone parameters while treating the last layer as a deterministic function of those parameters. We show that this requires no backpropagation through the closed-form solution (Section 3).

2. We adapt the approach to stochastic mini-batch training by regularizing for previous last layer solutions, producing a practical algorithm that integrates cleanly with standard training pipelines and that admits an approximate Kalman filter interpretation (Section 4).

3. We provide theoretical analyses. In the infinite width neural tangent kernel (NTK) limit we show that, under continuous time dynamics, despite non-convexity of the loss in function space, convergence of the method to an optimal solution is guaranteed. We further quantify the superiority of our method compared to SGD in the discrete time case with a one-step analysis (Section 5).

4. We validate the approach empirically, demonstrating improvements compared to standard training under squared losses, including applications in deep feature instrumental variable regression and Fourier neural operators (Section 6).

## 1.1 Related Work

We outline several strands of related work.

**Two-timescale regime.** Optimizing under a closed-form last layer can be seen as performing bilevel optimization, where an optimization problem is nested into another [35, 23]. In recent years, this last layer bilevel optimization approach has been considered in several works as a simplifying assumption for demonstrating convergence of gradient descent in neural networks. This framework was coined the two-timescale regime [19, 2, 3, 1].

[19] noted that, by the envelope theorem, optimizing with an optimal last layer as a function of the backbone parameters is equivalent to optimizing only the backbone parameters while keeping the last layer optimal. In the present work, we bring this theoretical argument to a practical method that can accelerate optimization, by observing that the envelope result can be leveraged computationally in backpropagation. Indeed, unlike these works, our aim is to propose novel methodology, and demonstrate its practicality in a number of scenarios. From the theoretical side, our work is the first to analyze the critical points of the resulting loss in function space, and the convergence to a global minimum in the NTK regime. Other works have investigated the mean-field regime instead [31, 28].

In an experiment, [1] propose to update the last layer by an exponential moving average of closed-form solutions to account for the stochasticity in SGD. However, this approach decouples the last layer from the backbone, as they no longer optimize the same loss, which leads to instabilities. In contrast, our approach for stochasticity allows the last layer and the backbone to continue optimizing for the same loss.

**Layer-wise learning.** [26, 34] propose to tune the learning rates of SGD layer-wise. [34] further show that, while shallower layers tend to have smaller gradients, this is not a reason for such layers needing larger learning rates, as the layer weights also appear to be smaller themselves. Indeed, [7] analyze the convergence speeds of different layers during optimization, and show that, despite smaller

gradients, shallower layers learn faster than deeper layers. Our method thus remediates this layer convergence bias for the last layer.

**Bayesian last layers.** These works leverage closed-form or quasi-closed-form solutions for last layer Bayesian posteriors to train them with variational inference [12, 5, 11]. In contrast, the present work does not construct a Bayesian posterior, and instead leverages closed-form solutions for the last layer to accelerate optimization.

**Feature learning in instrumental variables regression.** Training with a closed-form last layer has found applications in causal inference, particularly instrumental variables regression [32, 33]. In these applications, the closed-form last layer solution is required to solve the bilevel optimization problem. However, these works backpropagated through the closed-form solution, making them computationally expensive. Moreover, they did not include a proximal regularizer, and thus required large batch sizes for the networks to train. We address both limitations in the present work, and demonstrate superior performance of our method in these applications. [9] addresses the same limitations by removing the closed-form stage altogether, using a first-order Lagrangian reformulation of the bilevel problem with global convergence guarantees for two-layer networks. We instead keep the closed-form last layer but avoid differentiating through it.

**Randomized neural networks.** Randomized neural networks fix randomly generated hidden weights and solve for the output weights in closed form. Examples include the random vector functional link (RVFL) network [22, 21], its deep and ensemble deep extensions dRVFL and edRVFL [25], the broad learning system (BLS) [6], which expands the network in width, and stochastic configuration networks (SCNs) [30], which sample hidden weights under a supervisory mechanism. We refer to [18] for a survey of recent variants of these models. We view these as a degenerate case of our method, where hidden weights are random and only the last layer is solved in closed form. In contrast, our work is optimization-centric: the last layer and hidden weights are updated in a coupled way, and closed-form solves on mini-batches let the method scale to large datasets. A systematic comparison with state-of-the-art randomized networks is left for future work.

**Neural tangent kernel (NTK) regime.** NTK analysis typically assumes convexity of the loss in function space to show convergence to an optimum [15]. The present work goes beyond the standard analysis to show convergence to an optimum for our loss, which is shown to be non-convex in function space.

## 2 Background

**Non-linear multi-dimensional regression.** We consider the regression problem with a squared loss in which we aim to predict $y \in \mathbb{R}^o$ from the input $x \in \mathcal{X}$ (typically $\mathcal{X} \subset \mathbb{R}^m$), where $\mathcal{X}$ is some input space. We employ a model $f(x; W, \theta) = W\phi_\theta(x)$, where $\phi_\theta \colon \mathcal{X} \to \mathbb{R}^d$ is the neural network feature map, which we call the *backbone*, parametrised by $\theta \in \Theta$, $\Theta \subset \mathbb{R}^N$ is the feature parameter space, and $W \in \mathbb{R}^{o \times d}$ is the weight matrix for the last linear layer. Given $n$ observations $\{(x_i, y_i)\}_{i=1}^n$, we want to find the parameters $(W, \theta)$ that minimize the regularized squared loss

$$\mathcal{L}(W, \theta) = \sum_{i=1}^{n} \|y_i - W\phi_\theta(x_i)\|_2^2 + \beta \|W\|_F^2 \tag{1}$$

for some hyperparameter $\beta > 0$, where $\|\cdot\|_F$ is the Frobenius norm. For a fixed $\theta$, the objective (1) is a ridge regression problem for $W$ which enjoys the closed-form solution (derivation in Appendix A.1)

$$W^\star(\theta) := \underset{W \in \mathbb{R}^{o \times d}}{\arg\min}\, \mathcal{L}(W, \theta) = Y\phi_\theta(X)^\top \left(\phi_\theta(X)\phi_\theta(X)^\top + \beta I\right)^{-1} \tag{2}$$

where $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_n) \in \mathbb{R}^{o \times n}$ and $\phi_\theta(X) \in \mathbb{R}^{d \times n}$ is a matrix where we apply $\phi_\theta$ to each $x_i$.

**Gradient descent.** An approach to optimize (1) for $W$ and $\theta$ is to use gradient descent, an iterative optimization algorithm which starts from $(W_0, \theta_0)$ and at each iteration $t$ solves the following linearized optimization problem

$$W_{t+1}, \theta_{t+1} = \underset{W, \theta}{\arg\min}\, \nabla\mathcal{L}(W_t, \theta_t)^\top [W, \theta] + \frac{1}{2\eta}||W - W_t||_F^2 + \frac{1}{2\alpha}||\theta - \theta_t||_F^2, \tag{3}$$

where $(\alpha, \eta)$ are learning rates and $[W, \theta]$ denotes the concatenation operation. The solution to (3) can be expressed with the familiar gradient descent updates

$$W_{t+1} = W_t - \eta \nabla_W \mathcal{L}(W_t, \theta_t), \qquad \theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}(W_t, \theta_t). \tag{4}$$

## 3 Optimizing with a Closed-Form Last Layer

During optimization, we would like to leverage the fact that, for each $\theta$, the optimal last layer $W^\star(\theta)$ is available in closed-form (Eq. (2)). The idea is that there is no need to update $W_t$ through gradient steps as in Eq. (4), as we may treat it directly as a function of $\theta$ through Eq. (2). This leads to the loss

$$\mathcal{L}^\star(\theta) := \mathcal{L}(W^\star(\theta), \theta) = \sum_{i=1}^{n} \|y_i - W^\star(\theta)\phi_\theta(x_i)\|_2^2 + \beta \|W^\star(\theta)\|_F^2 \tag{5}$$

We now propose to optimize this loss instead of Eq. (1). Computationally, this involves alternating between solving linear regressions to obtain $W^\star(\theta)$ and gradient steps on $\theta$ through $\mathcal{L}$ (backpropagating through the closed-form solution to the regression). Explicitly, we start at some $\theta_0$ and iterate

$$W_{t+1} = W^\star(\theta_t), \qquad \theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}^\star(\theta_t). \tag{6}$$

Note that $\nabla_\theta \mathcal{L}^\star(\theta_t)$ involves backpropagating through $W^\star(\theta)$ given by Eq. (2), and hence through an inverse. This is computationally demanding. Fortunately, this operation is not needed, as the following result demonstrates (see also the *envelope theorem* and [19, Remark 1]):

**Proposition 3.1.** *For fixed $\theta$, letting $W^\star := W^\star(\theta)$ with Eq. (2), we have*

$$\nabla_\theta \mathcal{L}^\star(\theta) = \nabla_\theta \mathcal{L}(W^\star, \theta) \tag{7}$$

*Proof.* By the chain rule,

$$\nabla_\theta \mathcal{L}^\star(\theta) = \underbrace{\nabla_W \mathcal{L}(W^\star, \theta)}_{=0} \mathrm{D}W^\star(\theta)^\top + \nabla_\theta \mathcal{L}(W^\star, \theta) \underbrace{\mathrm{D}\,\mathrm{id}(\theta)^\top}_{=I} = \nabla_\theta \mathcal{L}(W^\star, \theta) \tag{8}$$

where $\nabla_W \mathcal{L}(W^\star, \theta) = 0$ since $W^\star = \arg\min_W \mathcal{L}(W, \theta)$ and D denotes the differential operator. □

Unlike $\nabla_\theta \mathcal{L}^\star(\theta)$ which requires complex backpropagation, $\nabla_\theta \mathcal{L}(W^\star, \theta)$ uses standard backpropagation through $\phi_\theta$ with the fixed last layer $W^\star$. From Proposition 3.1, Eq. (6) is equivalent to

$$W_{t+1} = W^\star(\theta_t), \qquad \theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}(W_{t+1}, \theta_t), \tag{9}$$

i.e. it suffices to replace the gradient step on $W$ in Eq. (4) by a closed-form update of the form Eq. (2).

## 4 The Practical Method

In practice, neural networks are not trained with gradient descent, but with *stochastic* gradient descent, or variants thereof. At time $t$ we observe a batch of data $\mathcal{B}_t \subset \{(x_i, y_i)\}_{i=1}^n$. Then the squared loss on the batch is given by

$$\mathcal{L}_{\mathcal{B}_t}(W, \theta) := \sum_{(x_i, y_i) \in \mathcal{B}_t} \|y_i - W\phi_\theta(x_i)\|_2^2 + \beta \|W\|_F^2. \tag{10}$$

As before, we may write

$$W^\star_{\mathcal{B}_t}(\theta) := \underset{W \in \mathbb{R}^{o \times d}}{\arg\min}\, \mathcal{L}_{\mathcal{B}_t}(W, \theta) \tag{11}$$

and update

$$W_{t+1} = W^\star_{\mathcal{B}_t}(\theta_t), \qquad \theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}_{\mathcal{B}_t}(W_{t+1}, \theta_t). \tag{12}$$

While such an approach will be valid for large batch sizes, it will be ineffective for small batches as the last layer $W^\star_{\mathcal{B}_t}(\theta)$ will overfit to each batch $\mathcal{B}_t$ at each $t$. The last layer estimates $W_{t+1}$ will then vary drastically at each iteration. As a consequence, the features $\phi_{\theta_t}(X)$ might not be able to adapt

to an unstable last layer. The smaller the batch size relative to the complete dataset, the more severe the issue (see the experiments in Section 6).

Instead, we propose to optimize a different loss. Motivated by how gradient descent regularizes to the previous estimates of the parameters (see Eq. (3)) we propose to regularize the objective function against the distance from $W$ to the previous estimate $W_t$, yielding the *proximal loss*

$$\mathcal{L}^{\text{prox}}_{\mathcal{B}_t,W_t}(W,\theta) := \sum_{(x_i,y_i)\in\mathcal{B}_t} \|y_i - W\phi_\theta(x_i)\|_2^2 + \lambda\|W - W_t\|_F^2 \tag{13}$$

where $\|\cdot\|_F$ is the Frobenius norm and $\lambda > 0$ is some hyperparameter. This ensures that closed-form solutions to Eq. (13) are close to the previous estimate $W_t$.

As before, we define (derivation in Appendix A.2)

$$W^\star_{\mathcal{B}_t,W_t}(\theta) := \operatorname*{arg\,min}_{W\in\mathbb{R}^{o\times d}} \mathcal{L}^{\text{prox}}_{\mathcal{B}_t,W_t}(W,\theta) = \left(Y_t\phi_\theta(X_t)^\top + \lambda W_t\right)\left(\phi_\theta(X_t)\phi_\theta(X_t)^\top + \lambda I\right)^{-1} \tag{14}$$

where $X_t$ and $Y_t$ correspond to the matrix of inputs and outputs of batch $\mathcal{B}_t$ respectively, and

$$\mathcal{L}^{\text{prox}\,\star}_{\mathcal{B}_t,W_t}(\theta) := \mathcal{L}^{\text{prox}}_{\mathcal{B}_t,W_t}(W^\star_{\mathcal{B}_t,W_t}(\theta),\theta). \tag{15}$$

Thus we propose to start at some $(W_0,\theta_0)$ and iterate

$$W_{t+1} = W^\star_{\mathcal{B}_t,W_t}(\theta_t), \qquad \theta_{t+1} = \theta_t - \alpha\nabla_\theta\mathcal{L}^{\text{prox}\,\star}_{\mathcal{B}_t,W_t}(\theta_t). \tag{16}$$

This approach addresses the stochasticity issues and ensures that $W$ and $\theta$ optimize the same loss, namely $\mathcal{L}^{\text{prox}}$. Moreover, we can obtain a result analogous to Proposition 3.1 for the proximal loss:

**Proposition 4.1.** *For fixed $\theta$, letting $W^\star_{\mathcal{B}_t,W_t} := W^\star_{\mathcal{B}_t,W_t}(\theta)$, we have*

$$\nabla_\theta\mathcal{L}^{\text{prox}\,\star}_{\mathcal{B}_t,W_t}(\theta) = \nabla_\theta\mathcal{L}_{\mathcal{B}_t}(W^\star_{\mathcal{B}_t,W_t},\theta). \tag{17}$$

*Proof.* As for Proposition 3.1, we have $\nabla_\theta\mathcal{L}^{\text{prox}\,\star}_{\mathcal{B}_t,W_t}(\theta) = \nabla_\theta\mathcal{L}^{\text{prox}}_{\mathcal{B}_t,W_t}(W^\star_{\mathcal{B}_t,W_t},\theta)$. Since the regularizer $\|W^\star_{\mathcal{B}_t,W_t} - W_t\|$ does not depend on $\theta$, we have $\nabla_\theta\mathcal{L}^{\text{prox}}_{\mathcal{B}_t,W_t}(W^\star_{\mathcal{B}_t,W_t},\theta) = \nabla_\theta\mathcal{L}_{\mathcal{B}_t}(W^\star_{\mathcal{B}_t,W_t},\theta)$. □

Like Proposition 3.1, Proposition 4.1 replaces the demanding backpropagation of $\nabla_\theta\mathcal{L}^{\text{prox}\,\star}_{\mathcal{B}_t,W_t}(\theta)$ by a standard backpropagation step for $\nabla_\theta\mathcal{L}_{\mathcal{B}_t}(W^\star_{\mathcal{B}_t},\theta)$. So Eq. (16) is equivalent to

$$W_{t+1} = W^\star_{\mathcal{B}_t,W_t}(\theta_t), \qquad \theta_{t+1} = \theta_t - \alpha\nabla_\theta\mathcal{L}_{\mathcal{B}_t}(W_{t+1},\theta_t). \tag{18}$$

In Appendix B we give another interpretation for Eq. (18) as performing approximate Kalman filtering on the last layer throughout SGD on the backbone parameters.

### 4.1 Numerical considerations

We now discuss practical choices required to implement the updates (18) in a practical algorithm.

**Use of a bias term.** We could add an additional bias dimension to feature vector $\phi_\theta$ and a learnable bias $b$ to the last layer $W$, which lead to extended vectors $\tilde{\phi}_\theta = [\phi_\theta, 1]$ and $\tilde{W} = [W, b]$. Overall, we found it was always beneficial for empirical performance to add bias.

**Last layer initialization.** First, we consider *zeros*, i.e. $W_0^{ij} = 0, \forall i,j$, which worked the best in practice. We then consider classical weight initializations – *LeCun normal* $W_0^{ij} \sim \mathcal{N}\left(0, \frac{1}{d}\right)$, *Xavier normal* $W_0^{ij} \sim \mathcal{N}\left(0, \frac{2}{d+o}\right)$ and *He normal* $W_0^{ij} \sim \mathcal{N}\left(0, \frac{2}{d}\right)$. Bias is always initialized as $b_0^i = 0$. See Appendix G.2 for detailed ablations.

**Full algorithm.** The complete method is given in Algorithm 1. The loop is structured so the last layer is re-fit after each gradient update, ensuring that the last layer is up-to-date at the end of each iteration.

**Algorithm 1** Proximal closed-form SGD

**Given:** Batch size $B$, proximal coefficient $\lambda > 0$, neural network $\phi_\theta$ with initial parameters $\theta_0$, learning rate $\alpha > 0$, initial last layer parameters $W_0$.
$t \leftarrow 0$
**Fit last layer on the first batch**
$W_1 \leftarrow W^\star_{\mathcal{B}_1, W_0}(\theta_t)$
**while** $\theta_t$ has not converged **do**
    $t \leftarrow t + 1$
    **Update backbone on the current batch** $\mathcal{B}_t$
    $\theta_t \leftarrow \theta_{t-1} - \alpha \nabla_\theta \mathcal{L}_{\mathcal{B}_t}(W_t, \theta_{t-1})$
    **Update last layer on the next batch** $\mathcal{B}_{t+1}$
    $W_{t+1} \leftarrow W^\star_{\mathcal{B}_{t+1}, W_t}(\theta_t)$
**Output:** Optimized $(W^\star, \theta^\star)$

We present an alternative method in Appendix D as Algorithm 2. In this alternative method, the gradient and closed-form solutions are computed on the same batch within each iteration, making it simpler to implement. However it departs from our theoretical framework laid out in Eq. (18) and Proposition 4.1.

### 4.2 Application to Classification

While the above methodology is mainly introduced for regression problems with $\ell_2$ loss, here we discuss how it could be extended to classification settings. Please see Appendix G.1 for the experimental results on classification.

We treat the output $y_i$ as one-hot vectors, i.e., $y_i \in \{0, 1\}^C$ such that $\sum_{c=1}^C [y_i]_c = 1$, where $C = o$ is the number of classes. We then can directly optimize the a squared loss on one-hot vectors (see for instance [14] for the use of the squared loss in classification) using (18) to optimize $W$ and $\theta$. This, however does not guarantee that the model $f(x; W, \theta) = W\phi_\theta(x)$ outputs probability vectors, i.e. $\sum_{c=1}^C W_{c\cdot}\phi_\theta(x) \neq 1$, where $W_{c\cdot}$ is the $c^{\text{th}}$ row of $W$. Therefore, for prediction, we take the arg max over output vectors, i.e. $c_{\text{out}}(x) = \arg\max_c W_{c\cdot}\phi_\theta(x)$. While this strategy is a simple heuristic, we found that using it together with Eq. (18) led to reasonable performance.

[14] proposes changes in the parametrization of the squared loss that empirically improve the performance on classification; in Appendix A.3 we derive a formula for closed-form solutions for these squared loss generalizations. These can be expensive to compute for large number of classes, so we do not make use of them in this work.

## 5 Theoretical Analyses

### 5.1 Analysis of the Modified Loss

In the section we uncover theoretical insights for the loss $\mathcal{L}^\star(\theta)$ from Eq. (5). For a tractable analysis, we will start by considering this loss as a function of the backbone, instead of the parameters $\theta$.

Let $\mathcal{F}$ be the space of functions $\phi\colon \mathcal{X} \to \mathbb{R}^d$. We define the loss (1) with a backbone $\phi$ as second argument:

$$\mathcal{L}_{\mathcal{F}}(W, \phi) = \sum_{i=1}^n \|y_i - W\phi(x_i)\|_2^2 + \beta\|W\|_F^2 \tag{19}$$

where $W \in \mathbb{R}^{o\times d}$, $\phi \in \mathcal{F}$ and $\beta \geq 0$. Then, as before we define $W^\star_{\mathcal{F}}(\phi) := \arg\min_{W\in\mathbb{R}^{o\times d}} \mathcal{L}_{\mathcal{F}}(W, \phi)$ and $\mathcal{L}^\star_{\mathcal{F}}(\phi) := \mathcal{L}^\star_{\mathcal{F}}(W^\star_{\mathcal{F}}(\phi), \phi)$.

The following theorem shows that $\mathcal{L}^\star_{\mathcal{F}}$ possesses unexpected characteristics.

**Theorem 5.1.** *If $Y \neq 0$ then $\mathcal{L}^\star_\mathcal{F}$ is not convex and admits critical points $\phi^\star$ that are not global minimizers.*

The proof for this theorem can be found in Appendix C.1.

In contrast, the usual squared (or ridge) loss $\sum_{i=1}^n \|y_i - f(x_i)\|_2^2$, where $f\colon \mathcal{X} \to \mathbb{R}^o$ which does not use a closed-form solution on the last layer $W^\star_\mathcal{F}(\phi)$, is convex in $f$ . The critical points of this loss function are exactly the functions $f^\star$ such that $f^\star(x_i) = y_i$ for all $i$ (or $f^\star(x_i) \approx y_i$ in the presence of a ridge regularizer). The non-trivial critical points of $\mathcal{L}^\star_\mathcal{F}$ occur because, when the features $\phi(X)_{j\cdot}$ are orthogonal to the outputs $Y_{k\cdot}$ for all $j$ and $k$, there is no gradient information for the features. For example $\phi^\star = 0$ is always a critical point of $\mathcal{L}^\star_\mathcal{F}$.

Is this an issue when $\phi$ is a neural network? Intuitively, these 'bad' critical points form a 'lower dimensional manifold' in function space, so should be avoided with high probability. We formalize this intuition by analyzing the NTK infinite width neural network regime [15]. In this regime, the initial neural network function $\phi$ can be shown to be a Gaussian process with respect to the random initialization, controlled by the neural Gaussian process kernel (NGPK). The training dynamics of $\phi$ are given by kernel gradient descent in function space $\mathcal{F}$ with respect to the NTK. If the NTK is positive definite, we know that $\phi$ will converge to a critical point of $\mathcal{L}^\star_\mathcal{F}$ in $\mathcal{F}$. The following result shows that if we make the slightly stronger assumption that the NGPK is positive definite (see for example [10, Theorem 4.5]), then $\phi$ will converge to a global minimizer.

**Theorem 5.2.** *In the NTK regime with positive definite NGPK, assuming $\beta = 0$, $d \geq \operatorname{rank} Y$ and that the $x_i$ are distinct, we have that $\phi$ converges almost surely to a global minimizer $\phi^\star$ of $\mathcal{L}^\star_\mathcal{F}$.*

The assumptions $\beta = 0$, $d \geq \operatorname{rank} Y$ and that the $x_i$ are distinct simplify the proof, the latter two ensuring that the outputs $Y$ are expressible through the features $\phi(X)$. This theorem provides a sanity check that the non-convexity of the function space loss (Theorem 5.1) causes no problems during optimization. See Appendix C.2 for a formal description of the NTK regime and a proof.

### 5.2 Analysis of One-Step Performance

Section 5.1 uncovers theoretical insights on the modified loss $\mathcal{L}^\star$. It shows non-convexity in function space, but which does not appear to be an issue, using the NTK regime as a case study. However, it does not say anything about the gains of optimizing $\mathcal{L}^\star$ as opposed to $\mathcal{L}$.

To investigate this, we compare the decrease in loss $\mathcal{L}(W, \theta) - \mathcal{L}(W_{GD}, \theta)$ after a gradient step on the last layer with the decrease in loss $\mathcal{L}(W, \theta) - \mathcal{L}(W_{CF}, \theta)$ after a closed-form step. The following result quantifies the superiority of closed-form steps.

**Theorem 5.3.** *Fix $W$ and $\theta$, and assume $\beta = 0$ and that $W \notin \arg\min_{\tilde{W}} \mathcal{L}(\tilde{W}, \theta)$. Then for any learning rate $\eta > 0$,*

$$\frac{\mathcal{L}(W, \theta) - \mathcal{L}(W_{GD}, \theta)}{\mathcal{L}(W, \theta) - \mathcal{L}(W_{CF}, \theta)} \leq \left(1 + \frac{\operatorname{Var}_\mathbb{P}(\sigma^2)}{\mathbb{E}_\mathbb{P}[\sigma^2]^2}\right)^{-1} \leq 1 \tag{20}$$

*where $W_{GD} := W - \eta \nabla \mathcal{L}(W, \theta)$, $W_{CF} := W^\star(\theta)$ and $\mathbb{P}$ is a categorical distribution over the singular values $\sigma$ of $\phi_\theta(X)$, independent of $\eta$. If we write $\phi_\theta(X) = U \Sigma V^\top$ for its SVD and if all singular values are distinct, $\mathbb{P}$ is given by*

$$\mathbb{P}(\sigma = \Sigma_{ii}) = \frac{\|((W\phi_\theta(X) - Y)V)_{\cdot i}\|^2}{\|(W\phi_\theta(X) - Y)V\|_F^2}. \tag{21}$$

The proof for Theorem 5.3 can be found in Appendix C.3. Intuitively, the more spread out the singular values of $\phi_\theta(X)$, the worse conditioned is the quadratic problem for $W$, the larger is the coefficient of variation $\frac{\sqrt{\operatorname{Var}_\mathbb{P}(\sigma^2)}}{\mathbb{E}_\mathbb{P}[\sigma^2]}$, and the worse is a gradient step compared to a closed-form step. This bound is valid for optimal learning rates in GD, so in practice the discrepancy may be much greater.

## 6 Experiments

**Experimental setup.** We evaluate the performance of our proximal closed-form approach (Eq. (18)), denoted as "$\ell_2$ *c.f. proximal* ($\lambda$)". We compare this against a closed-form ridge regression baseline

(Eq. (12)), "$\ell_2$ *c.f. ridge* ($\beta$)", and a standard "$\ell_2$ *loss*" baseline optimized via Eq. (4). We expect "$\ell_2$ *c.f. proximal* ($\lambda$)" to be effective across all batch sizes while "$\ell_2$ *c.f. ridge* ($\beta$)" to only work well with large batch sizes. We sweep over all the hyperparameters such as $\lambda$ and $\beta$ as well as the learning rate $\alpha$ for every method. We train models on a training set and we select best hyper-parameters using a validation set. The performance metrics are reported on the hold-out *test* set, averaged over 3 random seeds (with 95% confidence intervals). See Appendix F for experimental details. We refer the reader to Appendix G for additional experiments and ablations, and Appendix H for timings.

**Quantum Chemistry regression.** The QM9 dataset [24] consists of approximately 133,000 small organic molecules, each represented by 435-dimensional Coulomb Matrix features. The task is to predict 12 quantum chemical properties. We train all methods for 50 epochs with various batch sizes and report test-set mean squared error (MSE), using either SGD or Adam to optimize the backbone. See Appendix F for further details.

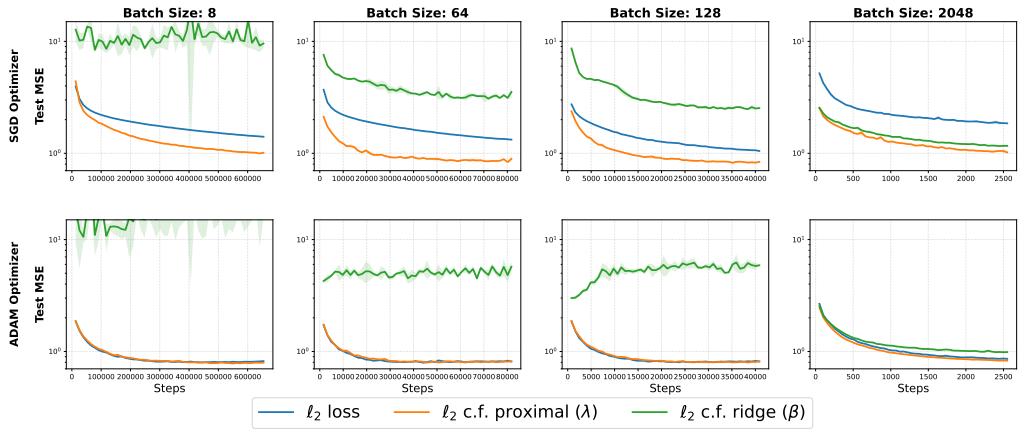


Figure 2: **QM9 results**. X-axis is the number of iterations, Y-axis is a test set mean squared error (MSE), columns represents different batch sizes, rows represent different backbone optimizers. Different colors indicate different methods.

Results are illustrated in Figure G.4. We observe that "$\ell_2$ *c.f. ridge* ($\beta$)" exhibits significant instability and fails to converge to a reasonable MSE at small batch sizes. Only at a batch size of 2048 does it attain performance comparable to "$\ell_2$ *c.f. proximal* ($\lambda$)". This is expected: a large batch already approximates the full dataset well, narrowing the gap between the ridge and proximal formulations. In contrast, our proximal method "$\ell_2$ *c.f. proximal* ($\lambda$)", consistently outperforms the standard "$\ell_2$ *loss*" when paired with SGD and performs comparably under Adam, demonstrating robustness to the choice of base optimizer.

**Impact of $\lambda$.** We study sensitivity of "$\ell_2$ *c.f. proximal* ($\lambda$)" to the $\lambda$ parameter on QM9. Smaller batch sizes require larger values of $\lambda$ for stable training: intuitively, a stronger proximal penalty prevents the closed-form solution from overfitting to noisy small-batch statistics. For larger batch sizes, where the batch better approximates the full dataset, smaller values of $\lambda$ suffice. Overall, we find that moderately large $\lambda$ (around $10^3$) is beneficial across settings.

**Fourier Neural Operator (FNO) regression.** We apply our approach to operator learning using a 1D Fourier Neural Operator (FNO) [17] on the viscous Burgers equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2} \tag{22}$$

on the periodic domain $\Omega = (0, 2\pi)$ and viscosity $\nu = 0.1$. Our dataset consists of 2048 initial conditions $u(t = 0, x)$ on a $N = 8192$ resolution grid together with their solution at time one $u(t = 1, x)$. The data is split into training (1448 points), validation (200 points) and test (400 points) sets. We train for 100 epochs on the 32-fold downsampled resolution grid (256 DoFs instead of 8192). The last layer $W \in \mathbb{R}^{H\times 1}$ is shared across spatial dimensions; consequently, a batch of size

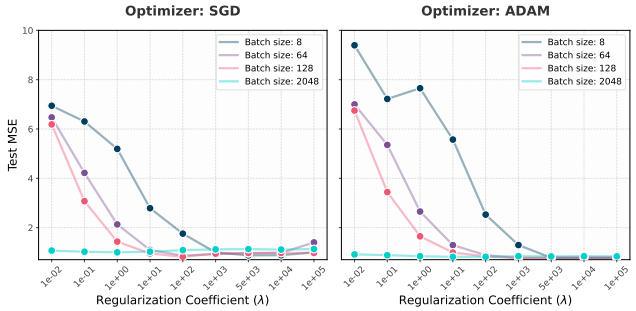


Figure 3: **QM9, impact of** $\lambda$. X-axis is $\lambda$, Y-axis is the final test mean squared error. First column shows results with the SGD optimizer, while second column shows results with the Adam optimizer.

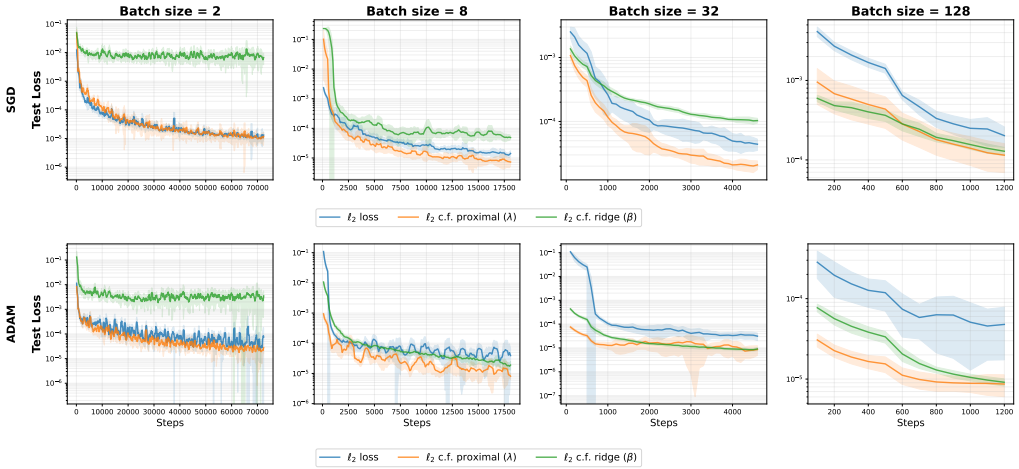


Figure 4: **FNO results**. X-axis is the number of iterations, Y-axis is a test set mean squared error (MSE), columns represents different batch sizes. Different colors indicate different methods. We use a rolling average with window size 5 to smooth the curves.

$B$ on a grid of 256 points yields an effective batch of $256B$ for the closed-form update. We report the mean squared error (MSE) on the whole $N = 8192$ grid (super-resolution evaluation).

Results are shown in Figure 4. Our approach "$\ell_2$ *c.f. proximal* ($\lambda$)" remains effective across batch sizes and optimizers. The ridge baseline "$\ell_2$ *c.f. ridge* ($\beta$)" fails to converge to a reasonable test loss at the smallest batch size (2) and is consistently outperformed by our method "$\ell_2$ *c.f. proximal* ($\lambda$)" at small batch sizes (8, 32) under SGD, while having similar performance only at larger batch sizes and with the Adam optimizer (on batch sizes 8, 32, 128). We conjecture that Adam's per-coordinate step sizes partly compensate for a suboptimal last layer, which is why the gap to the baseline is smaller under Adam than under SGD.

Note that, as discussed above, the effective batch seen by the last layer is the nominal batch size multiplied by the number of spatial degrees of freedom (256), thus even a nominal batch size of 2 corresponds to an effective batch size of 512. This mirrors our findings on QM9: as the batch size grows, the mini-batch objective (10) increasingly approximates the full-dataset setting (1), diminishing the advantage of the proximal term.

**Deep Feature Instrumental Variable (DFIV) regression.** We evaluate our method in a causal inference setting using the two-stage DFIV framework [32]; see Appendix E for background and details. We adapt DFIV to the minibatch setting and apply our proximal closed-form variant, which we call "*DFIV Proximal*". For evaluation, we use two strategies. The first follows [32] and re-estimates first-stage and second-stage last layers with ridge regression (we use 0.01 coefficient for this) on the whole training set. The second strategy uses the current estimates of the last layers.

The results are given in Fig. 5. For small batch sizes, our method outperforms standard DFIV, while achieving comparable performance in the large-batch regime. Notably, the online and re-estimated evaluations yield nearly identical results for our method, eliminating the need for a costly re-estimation pass over the full training set.

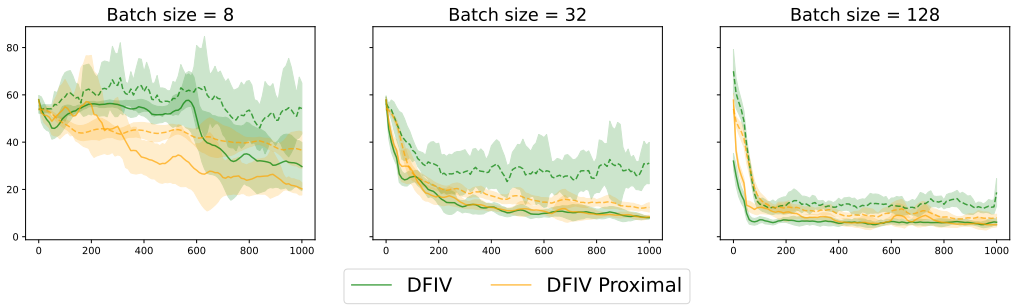


Figure 5: **DFIV results**. X-axis is the number of iterations, Y-axis is a test set MSE. Each column corresponds to a different batch size. Different colors indicate different methods. Solid lines use the last layer re-estimated on the entire training set, while dashed lines use current last layer estimates. We use a rolling average with window size 5 to smooth the curves.

**Further results.** In Appendices G to I we conduct additional experiments, notably we record timings (Appendices H and I), memory footprint (Appendix I) and classification tasks (Appendix G.1). For such tasks, "$\ell_2$ *loss*" usually underperforms "*Cross Entropy*" but, surprisingly, "$\ell_2$ *c.f. proximal* ($\lambda$)" achieves comparable performance to "*Cross Entropy*" on datasets with small to moderate number of classes. The closed-form update adds $O(Bd^2 + d^3)$ compute and $O(d^2)$ memory per step, where $B$ is the batch size and $d$ the last-layer feature dimension, independent of the backbone size. In practice, this amounts to a roughly 10–15% slowdown for large batches, but the overhead grows when the batch size is small and the last-layer dimension is large (up to $2.4\times$ for $B = 32$, $d = 4096$ on ResNet-18; see Appendix H).

## 7 Conclusion

We proposed leveraging closed-form optimal solutions for the last layer of neural networks under squared loss as an integral part of optimization.

Theoretically, we showed that the closed-form approach is well-founded in the neural tangent kernel regime, and bounded its superiority over gradient steps in a one-step analysis.

On regression tasks—including quantum chemistry (QM9), PDE operator learning (FNO/Burgers), and causal inference (DFIV)—our proximal closed-form approach consistently accelerates training and matches or outperforms standard gradient-based optimization of the squared loss. Compared to naive closed-form ridge regression, our proximal formulation is substantially more robust, particularly in the small-batch regime. The DFIV results are especially compelling: the closed-form solution is naturally motivated by the two-stage structure, and our method eliminates the need for costly re-estimation on the full training set.

When naively applied to classification, our method based on the squared loss achieves performance comparable to SGD on the cross entropy loss for tasks with a small to moderate number of classes. We observe it underperforms for tasks with a large number of classes. Indeed, a limitation of our work is that it only applies to the squared loss. Adapting it to cross entropy presents a promising area of future work, so that it may be successfully applied to more scenarios.

In addition, in future work we aim to apply our method to larger-scale two-stage problems such as offline reinforcement learning [8] and proxy variable regression [33]. We also aim to use a dimension-dependent regularization vector $\lambda$, and schedules for it that adapt over the course of training, which could mirror the strategies from adaptive optimizers such as Adam.

## 8 Acknowledgments

NDC and PH gratefully acknowledge co-funding by the European Union (ERC, ANUBIS, 101123955). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Unionor the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. NDC is supported by the Fonds National de la Recherche, Luxembourg, Project 17917615. PH is supported by the DFG through Project HE 7114/6-1 in SPP2298/2. PH is a member of the Machine Learning Cluster of Excellence, funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645. NDC and PH also gratefully acknowledge the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ:01IS18039A); and funds from the Ministry of Science, Research and Arts of the State of Baden-Württemberg.

## References

[1] Raphaël Barboni, Gabriel Peyré, and François-Xavier Vialard. Ultra-fast feature learning for the training of two-layer neural networks in the two-timescale regime, July 2025. arXiv:2504.18208 [cs].

[2] Raphaël Berthier, Andrea Montanari, and Kangjie Zhou. Learning Time-Scales in Two-Layers Neural Networks. *Foundations of Computational Mathematics*, August 2024.

[3] Alberto Bietti, Joan Bruna, and Loucas Pillaud-Vivien. On learning Gaussian multi-index models with gradient flow part I: General properties and two-timescale learning. *Communications on Pure and Applied Mathematics*, n/a, 2025.

[4] Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization, 2021.

[5] Paul Brunzema, Mikkel Jordahn, John Willes, Sebastian Trimpe, Jasper Snoek, and James Harrison. Bayesian Optimization via Continual Variational Last Layer Training. October 2024.

[6] C. L. Philip Chen and Zhulin Liu. Broad Learning System: An effective and efficient incremental learning system without the need for deep architecture. *IEEE Transactions on Neural Networks and Learning Systems*, 29(1):10–24, 2018.

[7] Yixiong Chen, Alan Yuille, and Zongwei Zhou. Which Layer is Learning Faster? A Systematic Exploration of Layer-wise Convergence Rate for Deep Neural Networks. September 2022.

[8] Yutian Chen, Liyuan Xu, Caglar Gulcehre, Tom Le Paine, Arthur Gretton, Nando de Freitas, and Arnaud Doucet. On Instrumental Variable Regression for Deep Offline Policy Evaluation. *Journal of Machine Learning Research*, 23(302):1–40, 2022.

[9] Zonghao Chen, Atsushi Nitanda, Arthur Gretton, and Taiji Suzuki. Towards a unified analysis of neural networks in nonparametric instrumental variable regression: Optimization and generalization. *arXiv preprint arXiv:2511.14710*, 2025.

[10] Tianxiang Gao, Xiaokai Huo, Hailiang Liu, and Hongyang Gao. Wide Neural Networks as Gaussian Processes: Lessons from Deep Equilibrium Models. *Advances in Neural Information Processing Systems*, 36:54918–54951, December 2023.

[11] James Harrison, John Willes, Paul Brunzema, and Jasper Snoek. Heteroscedastic Variational Last Layers. March 2025.

[12] James Harrison, John Willes, and Jasper Snoek. Variational Bayesian Last Layers. October 2023.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[14] Like Hui and Mikhail Belkin. Evaluation of neural architectures trained with square loss vs cross-entropy in classifications tasks. In *International Conference on Learning Representations*, 2021.

[15] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural Tangent Kernel: Convergence and Generalization in Neural Networks. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[16] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

[17] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[18] A. K. Malik, Ruobin Gao, M. A. Ganaie, M. Tanveer, and Ponnuthurai Nagaratnam Suganthan. Random vector functional link network: recent developments, applications, and future directions. *Applied Soft Computing*, 143:110377, 2023.

[19] Pierre Marion and Raphaël Berthier. Leveraging the two-timescale regime to demonstrate convergence of neural networks. *Advances in Neural Information Processing Systems*, 36:64996–65029, December 2023.

[20] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

[21] Yoh-Han Pao, Gwang-Hoon Park, and Dejan J. Sobajic. Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing*, 6(2):163–180, 1994.

[22] Yoh-Han Pao and Yoshiyasu Takefuji. Functional-link net computing: theory, system architecture, and functionalities. *Computer*, 25(5):76–79, 1992.

[23] Ieva Petrulionyte, Julien Mairal, and Michael Arbel. Functional Bilevel Optimization for Machine Learning. *Advances in Neural Information Processing Systems*, 37:14016–14065, December 2024.

[24] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1:140022, 2014.

[25] Qiushi Shi, Rakesh Katuwal, Ponnuthurai N. Suganthan, and M. Tanveer. Random vector functional link neural network based ensemble deep learning. *Pattern Recognition*, 117:107978, 2021.

[26] Bharat Singh, Soham De, Yangmuzi Zhang, Thomas Goldstein, and Gavin Taylor. Layer-Specific Adaptive Learning Rates for Deep Networks. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 364–368, December 2015.

[27] Simo Särkkä. *Bayesian Filtering and Smoothing*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, Cambridge, 2013.

[28] Shokichi Takakura and Taiji Suzuki. Mean-field Analysis on Two-layer Neural Networks from a Kernel Perspective, April 2024. arXiv:2403.14917 [cs].

[29] Michalis Titsias, Alexandre Galashov, Amal Rannen-Triki, Razvan Pascanu, Yee Whye Teh, and Jorg Bornschein. Kalman filter for online classification of non-stationary data. In *The Twelfth International Conference on Learning Representations*, 2024.

[30] Dianhui Wang and Ming Li. Stochastic configuration networks: Fundamentals and algorithms. *IEEE Transactions on Cybernetics*, 47(10):3466–3479, 2017.

[31] Guillaume Wang, Alireza Mousavi-Hosseini, and Lénaïc Chizat. Mean-Field Langevin Dynamics for Signed Measures via a Bilevel Approach. *Advances in Neural Information Processing Systems*, 37:35165–35224, December 2024.

[32] Liyuan Xu, Yutian Chen, Siddarth Srinivasan, Nando de Freitas, Arnaud Doucet, and Arthur Gretton. Learning Deep Features in Instrumental Variable Regression. October 2020.

[33] Liyuan Xu, Heishiro Kanagawa, and Arthur Gretton. Deep Proxy Causal Learning and its Application to Confounded Bandit Policy Evaluation. In *Advances in Neural Information Processing Systems*, volume 34, pages 26264–26275. Curran Associates, Inc., 2021.

[34] Yang You, Igor Gitman, and Boris Ginsburg. Large Batch Training of Convolutional Networks, September 2017. arXiv:1708.03888 [cs].

[35] Yihua Zhang, Prashant Khanduri, Ioannis Tsaknakis, Yuguang Yao, Mingyi Hong, and Sijia Liu. An Introduction to Bilevel Optimization: Foundations and applications in signal processing and machine learning. *IEEE Signal Processing Magazine*, 41(1):38–59, April 2024.

**A Derivations for the Closed-Form Solutions** **15**
A.1 Ridge Loss . . . 15
A.2 Proximal Loss . . . 15
A.3 Generalized Ridge Loss . . . 15

**B Kalman Filter Interpretation** **16**

**C Detailed Theoretical Analyses** **17**
C.1 Analysis of the Modified Loss . . . 17
C.2 Neural Tangent Kernel Infinite Width Limit . . . 18
C.3 Analysis of One-Step Performance . . . 20

**D Alternative Algorithm** **21**

**E Deep Feature Instrumental Variable Regression** **21**

**F Experimental Details** **23**

**G Additional Results** **24**
G.1 Application to Classification . . . 24
G.2 Impact of Initialization . . . 27
G.3 Use of a Bias Term . . . 28
G.4 Comparison of Algorithm 1 and Algorithm 2 . . . 28
G.5 Optimization Dynamics . . . 28
G.6 Adaptive Proximal Coefficient . . . 30
G.7 Additional Ablations on QM9 . . . 30

**H Wall-Clock Time Analysis on CIFAR-100** **31**

**I Wall-Clock Time, Memory Footprint, and Scalability Analysis on QM9** **32**
I.1 Computational Cost: Algorithm 1 vs. Algorithm 2 . . . 33
I.2 Empirical Scaling Results on A100 GPUs . . . 33
I.3 Shallow Backbones ($L = 3$) and End-to-End QM9 Training . . . 33

# A Derivations for the Closed-Form Solutions

In this appendix we derive the closed-form formulas for $W^\star(\theta)$ (Eq. (2)) and $W^\star_{\mathcal{B}_t,W_t}(\theta)$ (Eq. (14)).

## A.1 Ridge Loss

Recall Eq. (1):

$$\begin{aligned}\mathcal{L}(W,\theta) &= \sum_{i=1}^{n} \|y_i - W\phi_\theta(x_i)\|_2^2 + \beta\|W\|_F^2 \\ &= \|Y - W\phi_\theta(X)\|_F^2 + \beta\|W\|_F^2\end{aligned} \tag{23}$$

which we rewrote in matrix form, $\phi_\theta(X) \in \mathbb{R}^{d\times n}$, $Y \in \mathbb{R}^{o\times n}$, $W \in \mathbb{R}^{o\times d}$. Differentiating with respect to $W$ we get

$$\begin{aligned}\nabla_W \mathcal{L}(W,\theta) &= -2(Y - W\phi_\theta(X))\phi_\theta(X)^\top + 2\beta W \\ &= 2W\left(\phi_\theta(X)\phi_\theta(X)^\top + \beta I\right) - 2Y\phi_\theta(X)^\top.\end{aligned} \tag{24}$$

Solving for $W^\star$ such that $\nabla_W \mathcal{L}(W^\star,\theta) = 0$ we get the unique solution

$$W^\star = Y\phi_\theta(X)^\top\left(\phi_\theta(X)\phi_\theta(X)^\top + \beta I\right)^{-1}. \tag{25}$$

## A.2 Proximal Loss

Recall Eq. (13):

$$\begin{aligned}\mathcal{L}^{\text{prox}}_{\mathcal{B}_t,W_t}(W,\theta) &= \sum_{(x_i,y_i)\in\mathcal{B}_t} \|y_i - W\phi_\theta(x_i)\|_2^2 + \lambda\|W - W_t\|_F^2 \\ &= \|Y_t - W\phi_\theta(X_t)\|_F^2 + \lambda\|W - W_t\|_F^2.\end{aligned} \tag{26}$$

in matrix form, where $X_t$ and $Y_t$ correspond to the matrix of inputs and outputs respectively of batch $\mathcal{B}_t$. As in Appendix A.1, we compute the gradient with respect to $W$:

$$\begin{aligned}\nabla_W \mathcal{L}^{\text{prox}}_{\mathcal{B}_t,W_t}(W,\theta) &= -2(Y_t - W\phi_\theta(X_t))\phi_\theta(X_t)^\top + 2\lambda(W - W_t) \\ &= 2W\left(\phi_\theta(X_t)\phi_\theta(X_t)^\top + \lambda I\right) - 2Y_t\phi_\theta(X_t)^\top - 2\lambda W_t.\end{aligned} \tag{27}$$

Solving for $W^\star$ such that $\nabla_W \mathcal{L}^{\text{prox}}_{\mathcal{B}_t,W_t}(W^\star,\theta) = 0$ we get the unique solution

$$W^\star = \left(Y_t\phi_\theta(X_t)^\top + \lambda W_t\right)\left(\phi_\theta(X_t)\phi_\theta(X_t)^\top + \lambda I\right)^{-1}. \tag{28}$$

## A.3 Generalized Ridge Loss

[14] proposes to include additional hyperparameters within the squared loss objective in order to improve the performance on classification tasks. Namely, if $c$ is the correct class for datapoint $x$, consider the loss

$$\ell(W,\theta) = k\cdot([W\phi_\theta(x)]_c - m)^2 + \sum_{j=1,j\neq c}^{C} [W\phi_\theta(x)]_j^2 \tag{29}$$

for some hyperparameters $k, m > 0$. $\mathcal{L}^{\text{gen}}(W,\theta)$ is then given by the sum of the $\ell(W,\theta)$ for all training datapoints $x_i$ plus a regularizer. In this appendix we use a ridge regularizer $\beta\|W\|_F^2$, though we may use a proximal regularizer instead.

Let $W_{c\cdot}$, $Y_{c\cdot}$ be the $c^{\text{th}}$ row of the last layer, labels respectively, which we view both as column vectors. Let $z^{(c)} \in \mathbb{R}^n$ be the vector with entry $z_i^{(c)} = \sqrt{k}$ if $Y_{ci} = 1$, $z_i^{(c)} = 1$ otherwise. That is, $z^{(c)} = \mathbf{1} + (\sqrt{k} - 1)Y_{c\cdot}$, where $\mathbf{1} \in \mathbb{R}^n$ is the vector with ones in all entries. Finally, let $\odot$ denote the Hadamard product. We define the loss 'restricted' to the $c^{\text{th}}$ class as:

$$\mathcal{L}^{\text{gen}}_c(W_{c\cdot},\theta) = \left[z^{(c)}\odot\left(mY_{c\cdot} - \phi_\theta(X)^\top W_{c\cdot}\right)\right]^\top\left[z^{(c)}\odot\left(mY_{c\cdot} - \phi_\theta(X)^\top W_{c\cdot}\right)\right] + \beta\, W_{c\cdot}^\top W_{c\cdot}. \tag{30}$$

$\mathcal{L}^{\text{gen}}(W, \theta)$ is the sum over the $\mathcal{L}^{\text{gen}}_c(W_{c\cdot}, \theta)$. Differentiating only with respect to the $c^{\text{th}}$ row of $W$,

$$\begin{aligned}
\nabla_{W_{c\cdot}} \mathcal{L}^{\text{gen}}(W, \theta) &= \nabla_{W_{c\cdot}} \mathcal{L}^{\text{gen}}_c(W_{c\cdot}, \theta) \\
&= -2\phi_\theta(X) \left( (z^{(c)})^2 \odot \left( mY_{c\cdot} - \phi_\theta(X)^\top W_{c\cdot} \right) \right) + 2\beta W_{c\cdot} \\
&= 2 \left( \phi_\theta(X) \left( (z^{(c)})^2 \odot \phi_\theta(X) \right)^\top + \beta I \right) W_{c\cdot} - 2\phi_\theta(X) \left( (z^{(c)})^2 \odot mY_{c\cdot} \right) \\
&= 2 \left( \left( z^{(c)} \odot \phi_\theta(X) \right) \left( z^{(c)} \odot \phi_\theta(X) \right)^\top + \beta I \right) W_{c\cdot} \\
&\quad - 2 \left( z^{(c)} \odot \phi_\theta(X) \right) \left( z^{(c)} \odot mY_{c\cdot} \right) \\
&= 2 \left( \phi^{(c)}_\theta(X) \phi^{(c)}_\theta(X)^\top + \beta I \right) W_{c\cdot} - 2\phi^{(c)}_\theta(X) Y^{(c)}_{c\cdot}
\end{aligned} \tag{31}$$

where the square over $z^{(c)}$ is taken component-wise, $\phi^{(c)}_\theta(X) := z^{(c)} \odot \phi_\theta(X)$ and $Y^{(c)}_{c\cdot} := z^{(c)} \odot mY_{c\cdot}$.

Solving for $W^\star_{c\cdot}$ such that $\nabla_{W_{c\cdot}} \mathcal{L}^{\text{gen}}_c(W, \theta) = 0$ we get

$$W_{c\cdot} = \left( \phi^{(c)}_\theta(X) \phi^{(c)}_\theta(X)^\top + \beta I \right)^{-1} \phi^{(c)}_\theta(X) Y^{(c)}_{c\cdot}. \tag{32}$$

Equation (32) presents a computational bottleneck for large number of classes, since it requires a different matrix inversion per class.

# B Kalman Filter Interpretation

We can interpret the updates on the last layer in Eq. (18) as a Kalman filter under several simplifying assumptions. We take the Bayesian point of view, where we treat the last layer $W_t$ at time $t$ given the feature parameters $\theta_t$ as a random variable. See also [29] for a similar discussion.

First, we assume that the model fits the data perfectly during optimization, so we have the likelihood

$$p(y_i \mid x_i, W_t, \theta_t) = \mathcal{N}(y_i \mid W_t \phi_{\theta_t}(x_i), \sigma^2_Y I) \tag{33}$$

where $I$ is the identity matrix and $\sigma^2_Y$ is some hyperparameter controlling the variance of the outputs and $(x_i, y_i) \in \mathcal{B}_t$.

Next, we assume $W_t$ evolves like a random walk with Gaussian steps, as the parameters evolve through time $\theta_t$, so

$$p(W_{t+1} \mid W_t) = \mathcal{N}(W_{t+1} \mid W_t, \sigma^2_W I) \tag{34}$$

where $\sigma^2_W$ is some hyperparameter controlling the variance of the steps.

Equations (33) and (34) provide us a way to update our belief about $W_t$ in closed-form through time. Namely our belief about $W_t$ given our observations $\mathcal{B}_s$ for $s < t$ will be a Gaussian distribution $\mathcal{N}(W_t, \Sigma_t)$, obtained by *Kalman filtering* [27]. Here, note that $\Sigma_t$ will be a $od \times od$ matrix, i.e. will be squared times the number of parameters in the last layer. For large last layers, this can be intensive to store and manipulate.

Instead, we propose an additional simplifying approximation: we ignore the covariance $\Sigma_t$ at each step, and collapse our belief over $W_{t+1}$ to its maximum-a-posteriori estimate. The resulting update on our point estimate of the last layer is given by:

$$\begin{aligned}
W_{t+1} &= \operatorname*{arg\,min}_{W \in \mathbb{R}^{o \times d}} - \sum_{(x_i, y_i) \in \mathcal{B}_t} \log p(y_i \mid x_i, W, \theta_t) - \log p(W \mid W_t) \\
&= \operatorname*{arg\,min}_{W \in \mathbb{R}^{o \times d}} - \sum_{(x_i, y_i) \in \mathcal{B}_t} \log \mathcal{N}(y_i \mid W \phi_{\theta_t}(x_i), \sigma^2_Y I) - \log \mathcal{N}(W \mid W_t, \sigma^2_W I) \\
&= \operatorname*{arg\,min}_{W \in \mathbb{R}^{o \times d}} \sum_{(x_i, y_i) \in \mathcal{B}_t} \frac{1}{2\sigma^2_Y} \| y_i - W \phi_{\theta_t}(x_i) \|^2_2 + \frac{1}{2\sigma^2_W} \| W - W_t \|^2_F \\
&= W^\star_{\mathcal{B}_t, W_t}(\theta)
\end{aligned} \tag{35}$$

with $\lambda = \frac{\sigma_Y^2}{\sigma_W^2}$ in Eq. (14). That is, such approximate Bayesian updates recovers precisely the minimum of the proximal loss Eq. (14), leading to the updates on $W_t$ as in Eqs. (16) and (18).

# C Detailed Theoretical Analyses

This appendix contains the analysis and proofs for Section 5.

## C.1 Analysis of the Modified Loss

In this appendix we prove Theorem 5.1.

To begin, consider the case $\beta > 0$. Spelling out the expression for $\mathcal{L}^\star_{\mathcal{F}}$ in matrix form,

$$\begin{aligned}\mathcal{L}^\star_{\mathcal{F}}(\phi) &:= \sum_{i=1}^n \|y_i - W^\star_{\mathcal{F}}(\phi)\phi(x_i)\|_2^2 + \beta\|W^\star_{\mathcal{F}}(\phi)\|_F^2 \\ &= \|Y - W^\star_{\mathcal{F}}(\phi)\phi(X)\|_F^2 + \beta\|W^\star_{\mathcal{F}}(\phi)\|_F^2 \\ &= \operatorname{tr}\left((Y - W^\star_{\mathcal{F}}(\phi)\phi(X))^\top (Y - W^\star_{\mathcal{F}}(\phi)\phi(X))\right) + \beta \operatorname{tr}\left(W^\star_{\mathcal{F}}(\phi)^\top W^\star_{\mathcal{F}}(\phi)\right)\end{aligned} \tag{36}$$

where recall, $\phi(X) \in \mathbb{R}^{d\times n}$, $Y \in \mathbb{R}^{o\times n}$ and $W^\star_{\mathcal{F}}(\phi) := Y\phi(X)^\top \left(\phi(X)\phi(X)^\top + \beta I\right)^{-1} \in \mathbb{R}^{o\times d}$.

The derivative of $\mathcal{L}^\star_{\mathcal{F}}$ at $\phi$ is a linear map $\mathrm{D}\mathcal{L}^\star_{\mathcal{F}}(\phi)\colon \mathcal{F} \to \mathbb{R}$. Just as in Proposition 3.1, to calculate $\mathrm{D}\mathcal{L}^\star_{\mathcal{F}}(\phi)$ we do not need to differentiate $W^\star_{\mathcal{F}}(\phi)$ with respect to $\phi$, and may treat it as constant instead. So for $\psi \in \mathcal{F}$,

$$\mathrm{D}\mathcal{L}^\star_{\mathcal{F}}(\phi)[\psi] = \operatorname{tr}\big(\underbrace{-2(Y - W^\star_{\mathcal{F}}(\phi)\phi(X))^\top W^\star_{\mathcal{F}}(\phi)}_{=:\nabla\mathcal{L}^\star_{\mathcal{F}}(\phi)^\top}\psi(X)\big). \tag{37}$$

The definition of the gradient $\nabla\mathcal{L}^\star_{\mathcal{F}}(\phi) \in \mathbb{R}^{d\times n}$ is just a stand-alone notation used for convenience as – without an inner product on $\mathcal{F}$ – we do not have a well defined notion of gradients for the functional $\mathcal{L}^\star_{\mathcal{F}}$. Importantly, note that $\phi^\star$ is a critical point of $\mathcal{L}^\star_{\mathcal{F}}$ if and only if $\mathrm{D}\mathcal{L}^\star_{\mathcal{F}}(\phi^\star) = 0$, i.e. if and only if $\mathrm{D}\mathcal{L}^\star_{\mathcal{F}}(\phi^\star)[\psi] = 0$ for all $\psi \in \mathcal{F}$, i.e. if and only if $\nabla\mathcal{L}^\star_{\mathcal{F}}(\phi^\star) = 0$.

Plugging in the expression for $W^\star_{\mathcal{F}}(\phi)$ in the definition of $\nabla\mathcal{L}^\star_{\mathcal{F}}(\phi)$ and writing $\Phi := \phi(X) \in \mathbb{R}^{d\times n}$, $\Phi^+_\beta := \Phi^\top(\Phi\Phi^\top + \beta I)^{-1} \in \mathbb{R}^{n\times d}$, we get

$$\begin{aligned}\nabla\mathcal{L}^\star_{\mathcal{F}}(\phi) &= 2W^\star_{\mathcal{F}}(\phi)^\top(W^\star_{\mathcal{F}}(\phi)\phi(X) - Y) \\ &= 2(\Phi\Phi^\top + \beta I)^{-1}\Phi Y^\top Y(\Phi^\top(\Phi\Phi^\top + \beta I)^{-1}\Phi - I) \\ &= 2(\Phi^+_\beta)^\top Y^\top Y(\Phi^+_\beta\Phi - I).\end{aligned} \tag{38}$$

Define

$$Y_\star = Y\Phi^+\Phi, \quad Y_\perp = Y(I - \Phi^+\Phi). \tag{39}$$

where $\Phi^+ \in \mathbb{R}^{n\times d}$ is the Moore-Penrose pseudo-inverse of $\Phi$. In particular

$$Y = Y_\star + Y_\perp. \tag{40}$$

$Y_\star$ should be thought of the part of the outputs 'attainable' by the features $\Phi$, and $Y_\perp$ the part of the outputs 'unattainable'. To see this, take a singular value decomposition (SVD) of $\Phi$ of the form $\Phi = U\Sigma V^\top$ where $U \in \mathbb{R}^{d\times d}$ has orthonormal columns, $\Sigma \in \mathbb{R}^{d\times d}$ is diagonal with the first $r := \operatorname{rank}\Phi$ diagonal entries being non-zero and $V \in \mathbb{R}^{n\times d}$ has orthonormal columns. Moreover write $\mathbb{V}_\star \le \mathbb{R}^n$ for the subspace spanned by the rows of $\Phi$, or equivalently the first $r$ columns of $V$, the space of 'attainable' outputs. Further let $\mathbb{V}_\perp \le \mathbb{R}^n$ its orthogonal complement, the space of 'unattainable' outputs. We have $\Phi^+\Phi = V\Lambda V^\top$ the orthogonal projection onto $\mathbb{V}_\star$, where $\Lambda_{ij} = 1$ for $i = j \le r$, $\Lambda_{ij} = 0$ otherwise. Therefore the rows of $Y_\star$ are the orthogonal projection of those of $Y$ onto $\mathbb{V}_\star$, and those of $Y_\perp$ are the orthogonal projection onto $\mathbb{V}_\perp$.

Now note that

$$\begin{aligned}(\Phi^+_\beta)^\top Y^\top &= (\Phi\Phi^\top + \beta I)^{-1}\Phi Y^\top \\ &= U(\Sigma^2 + \beta I)^{-1}\Sigma V^\top Y^\top \\ &= U(\Sigma^2 + \beta I)^{-1}\Sigma V^\top Y_\star^\top \\ &= (\Phi^+_\beta)^\top Y_\star^\top.\end{aligned} \tag{41}$$

So from Eq. (38),

$$\nabla \mathcal{L}^\star_\mathcal{F}(\phi) = 2(\Phi^+_\beta)^\top Y_\star^\top Y(\Phi^+_\beta \Phi - I). \tag{42}$$

We see for example that $\phi = 0$, or more generally whenever the rows of $\Phi = \phi(X)$ are orthogonal to the rows of $Y$, we have $Y_\star = 0$ and $Y_\perp = Y$, so is a critical point of $\mathcal{L}^\star_\mathcal{F}$, even though it is not a global minimizer, assuming $Y \neq 0$.

When $Y \neq 0$, $\mathcal{L}^\star_\mathcal{F}$ admits critical points which are not global minima thus it is not convex, so this concludes the proof of Theorem 5.1 for $\beta > 0$.

*Remark* 1. To minimize $\mathcal{L}^\star_\mathcal{F}(\phi)$ in the case $\beta > 0$, one must express the rows of $Y$ (all, or as many as possible) with the span of the rows of $\Phi$, while at the same time taking $\Sigma_{ii} \to 0$ for all $i$ or $\Sigma_{ii} \to \infty$ for some $i$, to ensure that $\|W^\star_\mathcal{F}(\phi)\|^2_F = \|YV\Sigma(\Sigma^2 + \beta I)^{-1}U^\top\|^2_F \to 0$. The loss does not attain a global minimum.

For $\beta = 0$, the minimizer $W^\star_\mathcal{F}(\phi)$ of Eq. (19) is not unique, so we make the natural choice $W^\star_\mathcal{F}(\phi) = Y\Phi^+$. The Moore-Penrose pseudo-inverse operation is not continuous and hence not differentiable. But it is differentiable on any manifold of constant rank $\Phi$, and hence so is $\mathcal{L}^\star_\mathcal{F}$. Specifically (c.f. Eqs. (38) and (42))

$$\nabla \mathcal{L}^\star_\mathcal{F}(\phi) = 2(\Phi^+)^\top Y^\top Y(\Phi^+\Phi - I) = -2(\Phi^+)^\top Y_\star^\top Y_\perp. \tag{43}$$

As for $\beta > 0$, any $\phi$ where the rows of $\Phi$ are orthogonal to the rows of $Y$ is a critical point of $\mathcal{L}^\star_\mathcal{F}$. When $Y \neq 0$, this is not a global minimizer. Theorem 5.1 thus also follows in the case $\beta = 0$.

*Remark* 2. When $d \geq \operatorname{rank} Y$ and the $x_i$ are distinct, that is when the outputs $Y$ are expressible through the features $\phi(X)$, the minimizers correspond exactly to the points $\phi$ such that $Y_\perp = 0$ and $Y_\star = Y$.

*Remark* 3. A result such as Proposition 3.1 cannot be extended to second derivatives, otherwise we could differentiate $\mathcal{L}^\star_\mathcal{F}$ twice by keeping $W^\star_\mathcal{F}(\phi)$ constant, and would obtain that the Hessian of $\mathcal{L}^\star_\mathcal{F}$ is positive semi-definite since so is the one of the squared loss. But this is impossible since we showed that $\mathcal{L}^\star_\mathcal{F}$ is not convex.

### C.2 Neural Tangent Kernel Infinite Width Limit

In this appendix we provide a self-contained overview of the neural tangent kernel (NTK) limit, based on [15], and then prove Theorem 5.2.

$\phi_\theta$ is assumed to be a neural network, $\theta$ are its parameters consisting of weights $W^{(\ell)}$ and biases $b^{(\ell)}$, such that $\phi_\theta(x) = \alpha^{(L)}_\theta(x)$, the pre-activations $\tilde{\alpha}^{(\ell)}_\theta(x) \colon \mathbb{R}^{d_0} \to \mathbb{R}^{d_\ell}$ the activations $\alpha^{(\ell)}_\theta(x) \colon \mathbb{R}^{d_0} \to \mathbb{R}^{d_\ell}$, $d_L = d$ and

$$\begin{aligned} \alpha^{(0)}_\theta(x) &= x \\ \tilde{\alpha}^{(\ell+1)}_\theta(x) &= \frac{1}{\sqrt{d_\ell}} W^{(\ell)} \alpha^{(\ell)}_\theta(x) + b^{(\ell)} \\ \alpha^{(\ell)}_\theta(x) &= \sigma\left(\tilde{\alpha}^{(\ell)}_\theta(x)\right), \end{aligned} \tag{44}$$

where $\sigma \colon \mathbb{R} \to \mathbb{R}$ is a twice differentiable non-linearity function with bounded second derivative, applied element-wise. The parameters $\theta$ are initialized at time $t = 0$ with $W^{(\ell)}_{ij} \sim \mathcal{N}(0,1)$, $b^{(\ell)}_j \sim \mathcal{N}(0,1)$ which, combined with the pre-multiplicative factors $1/\sqrt{d_\ell}$ in Eq. (44), corresponds to the LeCun initialization (see Section 4.1).

We then consider gradient flow on the loss $\mathcal{L}^\star_\mathcal{F}$:

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = -\nabla_\theta \mathcal{L}^\star_\mathcal{F}(\theta). \tag{45}$$

Consider the limit $d_1, \dots, d_{L-1} \to \infty$ sequentially, that is we first take $d_1 \to \infty$, then $d_2 \to \infty$, etc. up to $d_{L-1} \to \infty$. In this limit, under some integrability assumption (see Remark 4), the dynamics of the function $\phi = \phi_\theta$ under Eq. (45) are given by kernel gradient descent: for $x \in \mathbb{R}^{d_0}$, $\beta = 0$,

$$\begin{aligned} \frac{\mathrm{d}\phi(x)}{\mathrm{d}t} &= -\sum_{i=1}^n k(x, x_i) \nabla \mathcal{L}^\star_\mathcal{F}(\phi)_i \\ &= 2(\Phi^+)^\top Y_\star^\top Y_\perp k(X, x) \end{aligned} \tag{46}$$

where we used Eq. (42), and $k$ is a positive semi-definite kernel $k\colon \mathbb{R}^{d_0} \times \mathbb{R}^{d_0} \to \mathbb{R}$, the *neural tangent kernel* [15, Theorem 1 & 2], $k(X, x) \in \mathbb{R}^n$. Whenever this kernel is positive definite (that is for all $x_1, \dots, x_N \in \mathbb{R}^{d_0}$, the matrix $(k(x_i, x_j))_{i,j}$ is positive definite; see for example [15, Proposition 2]) we know that, as $t \to \infty$, $\phi$ will converge pointwise to a critical point $\phi^\star$ of $\mathcal{L}^\star_{\mathcal{F}}$. The goal of Theorem 5.2 is to show that, almost surely in the initialization, $\phi^\star$ will be a global minimum of $\mathcal{L}^\star_{\mathcal{F}}$. In other words, under the assumption that $d \geq \operatorname{rank} Y$ and that the $x_i$ are distinct, we want to show $Y_\star \to Y$ and $Y_\perp \to 0$ as $t \to \infty$ (see Remark 2).

From Eq. (46), we see that

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = 2(\Phi^+)^\top Y_\star^\top Y_\perp K \tag{47}$$

where $K := k(X, X) \in \mathbb{R}^{n \times n}$.

First, consider the case where $d \leq n$. At initialization, the infinite width neural network is a Gaussian process [15, Proposition 1]. If the corresponding *neural Gaussian process kernel* (NGPK) is positive definite (see for example [10, Theorem 4.5]) then the distribution of the columns of $\Phi$ corresponding to distinct $x_i$ follow a non-degenerate Gaussian at initialization ($t = 0$). So, since $d \leq n$, $\Phi$ has full row rank $d$ at $t = 0$ almost surely. Thus, in that case, $\Phi^+ = \Phi^\top(\Phi\Phi^\top)^{-1}$. So

$$\begin{aligned}
\frac{\mathrm{d}Y_\star}{\mathrm{d}t} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(Y\Phi^\top(\Phi\Phi^\top)^{-1}\Phi\right) \\
&= Y\frac{\mathrm{d}\Phi^\top}{\mathrm{d}t}(\Phi\Phi^\top)^{-1}\Phi - Y\Phi^\top(\Phi\Phi^\top)^{-1}\frac{\mathrm{d}\Phi}{\mathrm{d}t}\Phi^\top(\Phi\Phi^\top)^{-1}\Phi \\
&\quad - Y\Phi^\top(\Phi\Phi^\top)^{-1}\Phi\frac{\mathrm{d}\Phi^\top}{\mathrm{d}t}(\Phi\Phi^\top)^{-1}\Phi + Y\Phi^\top(\Phi\Phi^\top)^{-1}\frac{\mathrm{d}\Phi}{\mathrm{d}t} \\
&= 2YKY_\perp^\top Y_\star\Phi^\top(\Phi\Phi^\top)^{-2}\Phi - 2Y\Phi^\top(\Phi\Phi^\top)^{-2}\Phi Y_\star^\top Y_\perp K\Phi^\top(\Phi\Phi^\top)^{-1}\Phi \\
&\quad - 2Y\Phi^\top(\Phi\Phi^\top)^{-1}\Phi KY_\perp^\top Y_\star\Phi^\top(\Phi\Phi^\top)^{-2}\Phi + 2Y\Phi^\top(\Phi\Phi^\top)^{-2}\Phi Y_\star^\top Y_\perp K.
\end{aligned} \tag{48}$$

Hence

$$\begin{aligned}
\frac{\mathrm{d}Y_\star}{\mathrm{d}t}Y_\star^\top &= 2YKY_\perp^\top Y_\star\Phi^\top(\Phi\Phi^\top)^{-2}\Phi Y_\star^\top - 2Y\Phi^\top(\Phi\Phi^\top)^{-2}\Phi Y_\star^\top Y_\perp K\Phi^\top(\Phi\Phi^\top)^{-1}\Phi Y_\star^\top \\
&\quad - 2Y\Phi^\top(\Phi\Phi^\top)^{-1}\Phi KY_\perp^\top Y_\star\Phi^\top(\Phi\Phi^\top)^{-2}\Phi Y_\star^\top + 2Y\Phi^\top(\Phi\Phi^\top)^{-2}\Phi Y_\star^\top Y_\perp KY_\star^\top \\
&= 2Y(I - \Phi^\top(\Phi\Phi^\top)^{-1}\Phi)KY_\perp^\top Y_\star\Phi^\top(\Phi\Phi^\top)^{-2}\Phi Y_\star^\top \\
&= 2\underbrace{\left(Y_\perp KY_\perp^\top\right)}_{=:A}\underbrace{\left(Y_\star\Phi^\top(\Phi\Phi^\top)^{-2}\Phi Y_\star^\top\right)}_{=:B}.
\end{aligned} \tag{49}$$

So

$$\frac{\mathrm{d}}{\mathrm{d}t}\|Y_\star\|_F^2 = 2\operatorname{tr}\left(\frac{\mathrm{d}Y_\star}{\mathrm{d}t}Y_\star^\top\right) = 2\operatorname{tr}(AB) \tag{50}$$

At $t = 0$, $\dim \mathbb{V}_\star = \operatorname{rank}\Phi = d$ almost surely. Since by assumption $d \geq \operatorname{rank} Y$, projecting the rows of $Y$ onto $\mathbb{V}_\star$ we get that $\operatorname{rank} Y_\star = \operatorname{rank} Y$ almost surely at $t = 0$. Moreover $B = Y_\star\Phi^\top(\Phi\Phi^\top)^{-2}\Phi Y_\star^\top = Y_\star V\Sigma^{-2}V^\top Y_\star^\top$ in the SVD notation from before, and $\Sigma_{ii} > 0$ for all $i$ almost surely. So $B$ is positive definite almost surely. $A$ is positive semi-definite so $\operatorname{tr}(AB) \geq 0$, and hence by Eq. (50), $\|Y_\star\|_F^2$ is non-decreasing through time. It converges when $A = 0$, i.e. $Y_\perp = 0$, which corresponds to global minima of $\mathcal{L}^\star_{\mathcal{F}}$. $Y_\star$ is guaranteed to converge when the NTK is positive definite.

We see in addition that, since the NTK is the sum of the NGPK with some other positive semi-definite kernel [15, Theorem 1], the NGPK being positive definite implies that the NTK is too.

For the case $n < d$, the optimization becomes trivial almost surely. Indeed, at $t = 0$, $\dim \mathbb{V}_\star = \operatorname{rank}\Phi = n$ almost surely. In other words $\mathbb{V}_\star = \mathbb{R}^n$ and $\mathbb{V}_\perp = \{0\}$, and thus $Y_\star = Y$ and $Y_\perp = 0$ almost surely at $t = 0$.

*Remark* 4. To show that the infinite width limit leads to kernel gradient descent as in Eq. (46), one needs to assume that, for any $T > 0$, the random variable (over the random initialization on $\theta$) given

by the integral

$$\int_0^T \|\nabla \mathcal{L}^\star_{\mathcal{F}}(\phi_\theta)\|_F \, \mathrm{d}t \tag{51}$$

stays stochastically bounded as $d_1, \ldots, d_{L-1} \to \infty$ sequentially [15, Theorem 2].

### C.3 Analysis of One-Step Performance

In this appendix we prove Theorem 5.3.

Taking a second order Taylor expansion of the loss around $W$:

$$\mathcal{L}(W_{GD}, \theta) = \mathcal{L}(W, \theta) - \eta \|\nabla_W \mathcal{L}(W, \theta)\|_F^2 + \frac{\eta^2}{2} H_W \mathcal{L}(W, \theta) \left[\nabla_W \mathcal{L}(W, \theta), \nabla_W \mathcal{L}(W, \theta)\right] \tag{52}$$

which is an exact equality since $\mathcal{L}$ is quadratic in $W$. Equation (52) is minimized at the learning rate

$$\eta^* = \frac{\|\nabla_W \mathcal{L}(W, \theta)\|_F^2}{H_W \mathcal{L}(W, \theta) \left[\nabla_W \mathcal{L}(W, \theta), \nabla_W \mathcal{L}(W, \theta)\right]}. \tag{53}$$

Note that the denominator of Eq. (53) is non-zero since $W \notin \arg\min_{\tilde{W}} \mathcal{L}(\tilde{W}, \theta)$. Let $\phi_\theta(X) = U \Sigma V^\top$ be an SVD, with $U \in \mathbb{R}^{d \times r}$, $\Sigma \in \mathbb{R}^{r \times r}$, $V \in \mathbb{R}^{n \times r}$ where $r$ is the rank of $\phi_\theta(X)$. Plugging Eq. (53) into Eq. (52) we get that for any $\eta > 0$

$$\begin{aligned}
\mathcal{L}(W, \theta) - \mathcal{L}(W_{GD}, \theta) &\leq \frac{\|\nabla_W \mathcal{L}(W, \theta)\|_F^4}{2 H_W \mathcal{L}(W, \theta) \left[\nabla_W \mathcal{L}(W, \theta), \nabla_W \mathcal{L}(W, \theta)\right]} \\
&= \frac{\|(W\phi_\theta(X) - Y)\phi_\theta(X)^\top\|_F^4}{\|(W\phi_\theta(X) - Y)\phi_\theta(X)^\top \phi_\theta(X)\|_F^2} \\
&= \frac{\|(W\phi_\theta(X) - Y)V\Sigma U^\top\|_F^4}{\|(W\phi_\theta(X) - Y)V\Sigma^2 V^\top\|_F^2} \\
&= \frac{\|(W\phi_\theta(X) - Y)V\Sigma\|_F^4}{\|(W\phi_\theta(X) - Y)V\Sigma^2\|_F^2}.
\end{aligned} \tag{54}$$

Now let $\mathbb{P}$ be the categorical distribution over the singular values $\sigma$ of $\phi_\theta(X)$ given by

$$\mathbb{P}(\sigma = \Sigma_{ii}) = \sum_{j : \Sigma_{jj} = \Sigma_{ii}} \frac{\|((W\phi_\theta(X) - Y)V)_{\cdot j}\|^2}{\|(W\phi_\theta(X) - Y)V\|_F^2}. \tag{55}$$

The sum in Eq. (55) is over repeated singular values. We get

$$\begin{aligned}
\|(W\phi_\theta(X) - Y)V\Sigma\|_F^2 &= \|(W\phi_\theta(X) - Y)V\|_F^2 \cdot \sum_{i=1}^r \frac{\|(W\phi_\theta(X) - Y)V)_{\cdot i}\|^2}{\|(W\phi_\theta(X) - Y)V\|_F^2} \Sigma_{ii}^2 \\
&= \|(W\phi_\theta(X) - Y)V\|_F^2 \cdot \mathbb{E}_{\mathbb{P}}[\sigma^2]
\end{aligned} \tag{56}$$

and similarly

$$\begin{aligned}
\|(W\phi_\theta(X) - Y)V\Sigma^2\|_F^2 &= \|(W\phi_\theta(X) - Y)V\|_F^2 \cdot \sum_{i=1}^r \frac{\|(W\phi_\theta(X) - Y)V)_{\cdot i}\|^2}{\|(W\phi_\theta(X) - Y)V\|_F^2} \Sigma_{ii}^4 \\
&= \|(W\phi_\theta(X) - Y)V\|_F^2 \cdot \mathbb{E}_{\mathbb{P}}[\sigma^4].
\end{aligned} \tag{57}$$

Plugging Eqs. (56) and (57) into Eq. (54) we get

$$\begin{aligned}
\mathcal{L}(W, \theta) - \mathcal{L}(W_{GD}, \theta) &\leq \|(W\phi_\theta(X) - Y)V\|_F^2 \cdot \frac{\mathbb{E}_{\mathbb{P}}[\sigma^2]^2}{\mathbb{E}_{\mathbb{P}}[\sigma^4]} \\
&= \|(W\phi_\theta(X) - Y)V\|_F^2 \cdot \left(1 + \frac{\mathrm{Var}_{\mathbb{P}}(\sigma^2)}{\mathbb{E}_{\mathbb{P}}[\sigma^2]^2}\right)^{-1}.
\end{aligned} \tag{58}$$

As discussed in Appendix C.1, the minimizer $W_{CF} = W^\star(\theta)$ is generally non-unique for $\beta = 0$. By definition of the minimizer, any choice will give the same value $\mathcal{L}(W_{CF}, \theta)$, so without loss

of generality we may assume $W_{CF} = Y\phi_\theta(X)^+$, where $\phi_\theta(X)^+ \in \mathbb{R}^{n\times d}$ is the Moore-Penrose pseudo-inverse of $\phi_\theta(X)$.

$$\mathcal{L}(W_{CF}, \theta) = \|Y\phi_\theta(X)^+\phi_\theta(X) - Y\|_F^2 = \|Y(I - VV^\top)\|_F^2 = \|(W\phi_\theta(X) - Y)(I - VV^\top)\|_F^2. \tag{59}$$

Then since $VV^\top$ is an orthogonal projection when acting on the right,

$$\begin{aligned}
\mathcal{L}(W, \theta) &= \|W\phi_\theta(X) - Y\|_F^2 \\
&= \|(W\phi_\theta(X) - Y)VV^\top + (W\phi_\theta(X) - Y)(I - VV^\top)\|_F^2 \\
&= \|(W\phi_\theta(X) - Y)VV^\top\|_F^2 + \|(W\phi_\theta(X) - Y)(I - VV^\top)\|_F^2 \\
&= \|(W\phi_\theta(X) - Y)V\|_F^2 + \mathcal{L}(W_{CF}, \theta).
\end{aligned} \tag{60}$$

Theorem 5.3 then follows by combining Eq. (60) with Eq. (58).

# D Alternative Algorithm

Algorithm 1 presented in Section 3 structures the updates in such a way that the backbone parameters are updated on the current batch and the last layer is updated on the future batch. This allows us to keep the last layer parameters up-to date. Moreover, this is theoretically justified since it directly follows the structure of (16) and Proposition 4.1.

Here, we present an alternative approach which we describe in Algorithm 2. In this algorithm, we update the backbone and the last layer on the same batch, but in a reversed order. Overall, we found this approach to lead to a similar performance as Algorithm 1, see Appendix G.4. The minor advantage of this method is that it is conceptually easier to implement (since we do not need to track two consecutive batches), but this approach departs from (16) and from Proposition 4.1, and therefore is less theoretically justified.

**Algorithm 2** Simple proximal closed-form SGD

**Given:** Batch size $B$, proximal coefficient $\lambda > 0$, neural network $\phi_\theta$ with initial parameters $\theta_0$, learning rate $\alpha > 0$, initial last layer parameters $W_0$.
$t \leftarrow 0$
**while** $\theta_t$ has not converged **do**
  $t \leftarrow t + 1$
  **Update backbone on the current batch** $\mathcal{B}_t$
  $\theta_t \leftarrow \theta_{t-1} - \alpha\nabla_\theta\mathcal{L}_{\mathcal{B}_t}(W_{t-1}, \theta_{t-1})$
  **Update last layer on the current batch** $\mathcal{B}_t$
  $W_t \leftarrow W^\star_{\mathcal{B}_t, W_{t-1}}(\theta_t)$
**Output:** Optimized $(W^\star, \theta^\star)$

# E Deep Feature Instrumental Variable Regression

In Instrumental Variable Regression, we observe a treatment $X$ and an outcome $Y$. But we have an unobserved confounder that affects both $X$ and $Y$, specifically we have the relation

$$Y = f_{\text{struct}}(X) + \epsilon, \quad \mathbb{E}[\epsilon] = 0, \quad \mathbb{E}[\epsilon \mid X] \neq 0 \tag{61}$$

where $f_{\text{struct}}$ is called the structural function which we aim to infer, and $\epsilon$ is an additive noise term. Because $\mathbb{E}[\epsilon \mid X] \neq 0$, we cannot use ordinary supervised learning techniques. Instead we assume we have access to an instrumental variable $Z$ which satisfies $\mathbb{E}[\epsilon \mid Z] = 0$. Then we have that $\mathbb{E}[Y \mid Z] = \mathbb{E}[f_{\text{struct}}(X) \mid Z]$, so we solve this equation for $f_{\text{struct}}$.

Deep Feature Instrumental Variable Regression (DFIV) [32] solves this by using two neural networks. The first neural network models $w^\top\psi_{\theta_X}(x) = f_{\text{struct}}(x)$, and the second neural network models $W\phi_{\theta_Z}(z) = \mathbb{E}[\psi_{\theta_X}(X) \mid Z = z]$. Training alternates between two stages. In the first stage, $W$ and $\theta_Z$ are fitted so that

$$W\phi_{\theta_Z}(z) = \mathbb{E}[\psi_{\theta_X}(X) \mid Z = z], \tag{62}$$

and in the second stage, $w$ and $\theta_X$ are fitted so that

$$w^\top W \phi_{\theta_Z}(z) = \mathbb{E}[Y \mid Z = z]. \tag{63}$$

When both Eqs. (62) and (63) are simultaneously satisfied we see that $\mathbb{E}[w^\top \psi_{\theta_X}(X) \mid Z] = \mathbb{E}[Y \mid Z] = \mathbb{E}[f_{\text{struct}}(X) \mid Z]$, as required. Since the first-stage solution $W$ depends on $\theta_X$, this is a bilevel optimization problem, and we alternate between the two stages.

**Algorithm 3** DFIV proximal

**Given:** Stage 1 data $\{(x_i^{(1)}, z_i^{(1)})\}$, stage 2 data $\{(y_i^{(2)}, z_i^{(2)})\}$, batch sizes $B_1, B_2$, proximal coefficients $\lambda_1, \lambda_2, \lambda_{1,2} > 0$, neural networks $\psi_{\theta_X}, \phi_{\theta_Z}$ with initial parameters $\theta_{X_0}, \theta_{Z_0}$ respectively, learning rates $\alpha_1, \alpha_2 > 0$, initial last layer parameters $w_0, W_0$, number of updates in each stage $T_1, T_2$.
$t_1 \leftarrow 0$
$t_2 \leftarrow 0$
**while** $\theta_{Z_{t_1}}$ and $\theta_{X_{t_2}}$ have not converged **do**
  Sample $B_1$ stage 1 data $\mathcal{B}^{(1)} \subset \{(x_i^{(1)}, z_i^{(1)})\}$, and $B_2$ stage 2 data $\mathcal{B}^{(2)} \subset \{(y_i^{(2)}, z_i^{(2)})\}$
  **for** $t = 1$ to $T_1$ **do**
    $t_1 \leftarrow t_1 + 1$
    $\theta_{Z_{t_1}} \leftarrow \theta_{Z_{t_1-1}} - \alpha_1 \nabla_{\theta_Z} \mathcal{L}^{(1)}_{\mathcal{B}^{(1)}, W_{t_1-1}}(W_{t_1-1}, \theta_{Z_{t_1-1}}, \theta_{X_{t_2}}; \lambda_1)$
    $W_{t_1} \leftarrow W^\star_{\mathcal{B}^{(1)}, W_{t_1-1}}(\theta_{Z_{t_1}}, \theta_{X_{t_2}}; \lambda_1)$ using (65) with proximal coefficient $\lambda_1$
  **for** $t = 1$ to $T_2$ **do**
    $t_2 \leftarrow t_2 + 1$
    Form $W^\star_{\mathcal{B}^{(1)}, W_{t_1}}(\theta_{Z_{t_1}}, \theta_X; \lambda_{1,2})$ as a function of $\theta_X$ via (65)
    $\theta_{X_{t_2}} \leftarrow \theta_{X_{t_2-1}} - \alpha_2 \nabla_{\theta_X} \mathcal{L}^{(2)}_{\mathcal{B}^{(2)}, w_{t_2-1}, W_{t_1}}(w_{t_2-1}, \theta_{X_{t_2-1}}, \theta_{Z_{t_1}}; \lambda_{1,2}, \lambda_2)$
    $w_{t_2} \leftarrow w^\star_{\mathcal{B}^{(2)}, w_{t_2-1}, W_{t_1}}(\theta_{X_{t_2}}, \theta_{Z_{t_1}}; \lambda_{1,2}, \lambda_2)$ using (67) with proximal coefficient $\lambda_2$
**Output:** Optimized $(w^\star, W^\star, \theta^\star_X, \theta^\star_Z)$

Similarly to the methodology introduced in Section 4, we fit both stages with stochastic proximal losses, using batches of data $\mathcal{B}^{(1)} \subset \{(x_i^{(1)}, z_i^{(1)})\}$ for the first stage and $\mathcal{B}^{(2)} \subset \{(y_i^{(2)}, z_i^{(2)})\}$ for the second stage. Given the previous estimate $W_t$, the first-stage loss with proximal coefficient $\lambda_1 \geq 0$ is

$$\mathcal{L}^{(1)}_{\mathcal{B}^{(1)}, W_t}(W, \theta_Z, \theta_X; \lambda_1) := \sum_{(x_i^{(1)}, z_i^{(1)}) \in \mathcal{B}^{(1)}} \|W \phi_{\theta_Z}(z_i^{(1)}) - \psi_{\theta_X}(x_i^{(1)})\|_2^2 + \lambda_1 \|W - W_t\|_F^2, \tag{64}$$

and we denote its minimizer with respect to the last layer by

$$W^\star_{\mathcal{B}^{(1)}, W_t}(\theta_Z, \theta_X; \lambda_1) = \arg\min_W \mathcal{L}^{(1)}_{\mathcal{B}^{(1)}, W_t}(W, \theta_Z, \theta_X; \lambda_1). \tag{65}$$

Solving for $W$ in closed form makes it depend on $\theta_X$, and DFIV leverages this dependence in the second stage. We make the proximal coefficient explicit in Eqs. (64) and (65) because the second stage uses the solution (65) with a different coefficient $\lambda_{1,2}$. Given the previous estimate $w_t$, the second-stage loss is

$$\begin{aligned}\mathcal{L}^{(2)}_{\mathcal{B}^{(2)}, w_t, W_t}(w, \theta_X, \theta_Z; \lambda_{1,2}, \lambda_2) := &\sum_{(y_i^{(2)}, z_i^{(2)}) \in \mathcal{B}^{(2)}} \|w^\top W^\star_{\mathcal{B}^{(1)}, W_t}(\theta_Z, \theta_X; \lambda_{1,2}) \phi_{\theta_Z}(z_i^{(2)}) - y_i^{(2)}\|_2^2 \\ &+ \lambda_2 \|w - w_t\|_2^2,\end{aligned} \tag{66}$$

and we denote its minimizer with respect to the last layer by

$$w^\star_{\mathcal{B}^{(2)}, w_t, W_t}(\theta_X, \theta_Z; \lambda_{1,2}, \lambda_2) = \arg\min_w \mathcal{L}^{(2)}_{\mathcal{B}^{(2)}, w_t, W_t}(w, \theta_X, \theta_Z; \lambda_{1,2}, \lambda_2). \tag{67}$$

For our method "$\ell_2$ *c.f. proximal* ($\lambda$)", we therefore use three distinct proximal hyperparameters: $\lambda_1$ for the closed-form solution $W^\star_{\mathcal{B}^{(1)}, W_t}(\theta_Z, \theta_X; \lambda_1)$ in the first stage (65), $\lambda_2$ for the closed-form

solution $w^{\star}_{\mathcal{B}^{(2)},w_t,W_t}(\theta_X, \theta_Z; \lambda_{1,2}, \lambda_2)$ in the second stage (67), and a separate $\lambda_{1,2}$ for the closed-form solution $W^{\star}_{\mathcal{B}^{(1)},W_t}(\theta_Z, \theta_X; \lambda_{1,2})$ used inside the second stage (66). In practice we found that it was beneficial to take $\lambda_{1,2} < \lambda_1$ See Algorithm 3 for full algorithmic details.

For the baseline "$\ell_2$ *c.f. ridge* ($\beta$)", we essentially replace all the proximal penalties in Eqs. (64) and (66) by ridge penalties, i.e. $\beta_1\|W\|_F^2$ and $\beta_2\|w\|_2^2$.

# F Experimental Details

**Hardware.** For every experiment we used $A100$ GPU. For all the regression experiments, every hyperparameter run was using 1 $A100$ GPU. For ImageNet experiments, every hyperparameter run used 16 $A100$ GPUs.

**Quantum Chemistry regression** . QM9 dataset [24] consists of approximately 133,000 small organic molecules and involves predicting 12 distinct quantum chemical properties from 435-dimensional Coulomb Matrix features to represent the molecular structures. We utilized a 3-layer MLP backbone (256 hidden units per layer, GELU activation) to regress 12 molecular properties from 435-dimensional Coulomb Matrix features. We train all the methods for $50$ epochs and we report test-set mean squared error (MSE). We use either SGD or Adam optimizers for the backbone. For SGD, we use Nesterov momentum with $0.9$ coefficient. For SGD, we initialize the last layer with zeros, while for Adam, we use LeCun initialization.

Following a standard 80/10/10 split, we performed a hyperparameter sweep over learning rates $\alpha \in \{0.1, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$, $\lambda \in \{0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, 5000.0, 10000.0, 100000.0\}$ and $\beta \in \{0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, 5000.0, 10000.0, 100000.0\}$. We selected the best hyperparameters using validation set MSE at the last $20\%$ of the training iterations.

**FNO regression.** We use a Fourier Neural Operator (FNO) [17] backbone with 4 FNO blocks (width 128, 16 Fourier modes, ReLU activation) as the feature extractor, with a separate linear last layer of dimension 1×128 (plus bias). Training is performed at a subsampled resolution (stride 32), and we additionally report test MSE at the full resolution. We train all methods for 100 epochs and report test-set mean squared error (MSE). We use either SGD optimizer (wiht momentum $0.9$, Nesterov) or Adam optimizers for the backbone. For SGD, we initialize the last layer with zeros, while for Adam, we use LeCun initialization.

Using a 1448/200/400 train/validation/test split, we performed a hyperparameter sweep over learning rates $\alpha \in \{10, 5, 2, 1, 0.5, 0.2, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005\}$ for SGD and $\alpha \in \{0.1, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$ for Adam, $\lambda \in \{0.0001, 0.001, 0.01, 0.1, 1.0, 10, 100, 1000, 10000\}$ and $\beta \in \{0.0001, 0.001, 0.01, 0.1, 1.0, 10, 100, 1000, 10000\}$. We used batch sizes $8, 32, 128$. All experiments are averaged over 3 random seeds. We selected the best hyperparameters using validation set MSE at the end of the training.

**DFIV regression.** For the experiments, we follow closely [32] and we consider a slightly modified version of `d-spirtes` task [20]. This is an image dataset described by five latent parameters $(\texttt{shape}, \texttt{scale}, \texttt{rotation}, \texttt{posX}, \texttt{posY})$. The images are $64 \times 64 = 4096$ dimensional. In this experiment, the authors fix the `shape` parameter to `heart`, i.e., they only used heart-shaped images. The authors generated data for IV regression in which they use each figure as a treatment variable $X$. Hence, the treatment variable is $4096$-dimensional in this experiment. To make the task more challenging, they used `posY` as the hidden confounder, which is not revealed to the model. They used three latent varaibles as the instrument variables $Z$. The outcome $Y$ is defined as

$$Y = f_{\text{struct}}(X) + 32(\texttt{posY} - 0.5) + \epsilon, \tag{68}$$

where $\epsilon \sim \mathcal{N}(0, 0.5)$. Here, we used $f_{\text{struct}}(X)$ from a different paper [33], which was defined as

$$f_{\text{struct}}(X) = \frac{(\texttt{vec}(B)^\top X)^2 - 3000}{500}, \tag{69}$$

where $B \in \mathbb{R}^{64\times 64}$, $B_{ij} = \frac{|32-j|}{32}$ and $\texttt{vec}(B)$ collapses the matrix $B$ to a vector of dimensionality $4096$. The choice of this structural function was motivated by [33], because the original choice described in [32] led to essentially a constant function (in expectation).

For our experiments, we use different batch sizes. The DFIV method [32] essentially corresponds to two-stage "$\ell_2$ *c.f. ridge* ($\beta$)" where we have $\beta_1$ and $\beta_2$ parameters for the first and second stage correspondingly. In our proximal method, DFIV proximal, as described in Appendix E, we have three parameters $\lambda_1$, $\lambda_2$ for the first and second stage proximal updates and $\lambda_{1,2}$ for first-stage update inside the second stage. In practice, we sweep over $\lambda_1$ and $\lambda_2$ and we use $\lambda_{1,2}$ to be very small, i.e. $\lambda_{1,2} = 0.0001$ as we found that using large $\lambda_{1,2}$ did not work well. We choose $T_1 = 20$ and $T_2 = 1$ as in [32].

The datasets are split into the training set with $10000$ points, validation set with $100$ points and holdout test set with $488$ points.

When we evaluate performance, we use two strategies. One is following [32] and whenever performance is reported, takes the first stage and second stage backbone parameters, and re-estimates the corresponding last layers on the whole $10000$ training set. In Figure 5 it is represented by the solid line. The second strategy just takes the current estimates of the last layers. In Figure 5 it is represented by the dashed line.

As optimizer, we use AdamW with weight decay parameter $0.1$. We use LeCun initialization for the last layer.

The sweep range for $\beta_1, \beta_2, \lambda_1, \lambda_2$ is $\{0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 10^3, 10^4\}$. On top of that, we also sweep over the learning rate (we use the same learning rate for both stages) in the range $\{0.001, 0.005, 0.01, 0.05, 0.1\}$. Each experiment is run with 3 seeds. The best hyperparameters are selected by minimizing the mean squred error (MSE) on the validation set at the end of the training, using the first evaluation strategy (re-estimating the last layers on the whole dataset). The performance is reported on the holdout test set.

**CIFAR-100.** We always use SGD optimizer with Nesterov momentum $\gamma = 0.9$. For SGD, we initialize the last layer with zeros, while for Adam, we use LeCun initialization. We train on the $80\%$ of the training set and we use the remaining $20\%$ for validation. For reporting performance, we use the corresponding test set. The sweep ranges are $\alpha \in \{10.0, 5.0, 2.0, 1.0, 0.5, 0.3, 0.2, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005\}$, learning rate and $\lambda \in \{0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]$, $\beta \in \{0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0\}$.

**ImageNet.** We follow closely the experimental setup described in [4], including learning rate schedule, label smoothing, data augmentations and Nesterov momentum in the SGD. The learning schedule is a warmup cosine decay with the peak learning rate $\alpha = 1.6$. We also swept over $\alpha \in [0.1, 1., 1.6, 2., 5.0]$ range. We use zeros initialization for the last layer. Overall, all the methods performed the best with $\alpha = 1.6$ except for "$\ell_2$ *loss*" which performed the best with $\alpha = 1$. For "$\ell_2$ *c.f. proximal* ($\lambda$)", we used $\lambda = 10000$ and for "$\ell_2$ *c.f. ridge* ($\beta$)", we used $\beta = 0.01$. To select these parameters, we ran a sweep over $\beta \in [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]$ and over $\lambda \in [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]$. We used validation set of ImageNet for the hyperparmaeters selection.

# G Additional Results

## G.1 Application to Classification

**CIFAR-100.** We perform experiments on the CIFAR-100 dataset [16], across batch sizes $B = [32, 128, 1024, 4096]$, where we use ResNet-18 [13] as a backbone $\phi_\theta$. Please refer to Appendix F for more details.

The results are presented in Figure G.1. Consistent with the regression setting, our method "$\ell_2$ *c.f. proximal* ($\lambda$)" outperforms the "$\ell_2$ *loss*" baseline under both optimizers, with the performance gap widening as batch size increases. The ridge method, "$\ell_2$ *c.f. ridge* ($\beta$)" fails to converge at small batch sizes and only reaches comparable performance at larger ones – underscoring the role of the

proximal term in preventing overfitting to individual mini-batches. Throughout, our method remains competitive and robust regardless of the choice between SGD and Adam. Notably, "$\ell_2$ *c.f. proximal* ($\lambda$)" even surpasses *Cross Entropy* on CIFAR-100, though as we show below, this advantage does not persist at the larger scale of ImageNet.

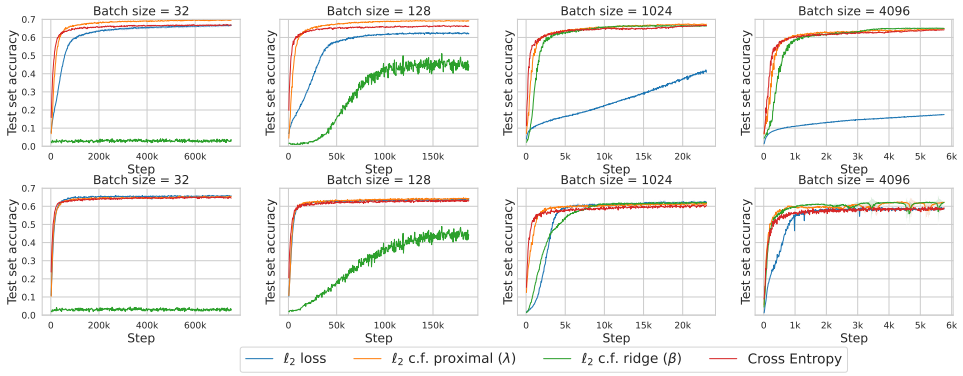


Figure G.1: **CIFAR-100 results.**. X-axis is the number of iterations, Y-axis is a test set accuracy. Each column corresponds to a different batch size. Different colors indicate different methods.

**Impact of $\lambda$ and $\beta$.** In Figure G.2, we report performance at the end of the training as a function $\lambda$ and $\beta$, as well as the best learning rate $\alpha$ for every batch size. For the first two plots we used the learning rates reported in the third plot. The method "$\ell_2$ *c.f. proximal* ($\lambda$)" is overall robust to $\lambda$ provided it is large enough. We only see some sensitivity for smaller batch sizes. The approach "$\ell_2$ *c.f. ridge* ($\beta$)" is more sensitive to the parameter $\beta$ and works better for larger batch sizes. Finally, both of the approaches benefit from large learning rates whenever batch size is increased, while "*Cross Entropy*" and "$\ell_2$ *loss*" require small learning rates.

**Class confidence.** We run a simple experiment on CIFAR-100 where we track the maximum probability of a correct class. for $\ell_2$ loss, we renormalize this probability to be positive (since the output of the model does not have to be positive nor normalized to 1), i.e., the probabilities are given by

$$p_c = \frac{\tilde{p}_c}{\sum_{c'} \tilde{p}_{c'}}, \tag{70}$$

where

$$\tilde{p}_c = f(x)_c - \min_{c'} f(x)_{c'}, \tag{71}$$

and $f(x) = (f(x)_1, \ldots, f(x)_C)$ is the output of the model. We present results in Table G.1. We see that our method has much lower maximum probabilities compared to cross entropy, which means that the results are less confident. This could be, in principle, problematic for datasets with large numbers of classes. In future work, we will investigate empirical strategies to improve this performance.

Table G.1: Comparison of maximum probabilities

| Batch size | Cross entropy max probs | Our method max probs |
|---|---|---|
| 32 | 0.99 | 0.63 |
| 128 | 0.99 | 0.62 |
| 1024 | 0.99 | 0.73 |
| 4096 | 0.99 | 0.76 |

**Additional ablations.** We ran ablations on the design choices for our method. We first verified that the inclusion of a bias term in the last layer did not lead to a difference in performance (see Figure G.6). Further, we found that the *zeros* initialization strategy led to the best results (see Figure G.5) when using SGD optimizer, consistent to the QM9 experiments.

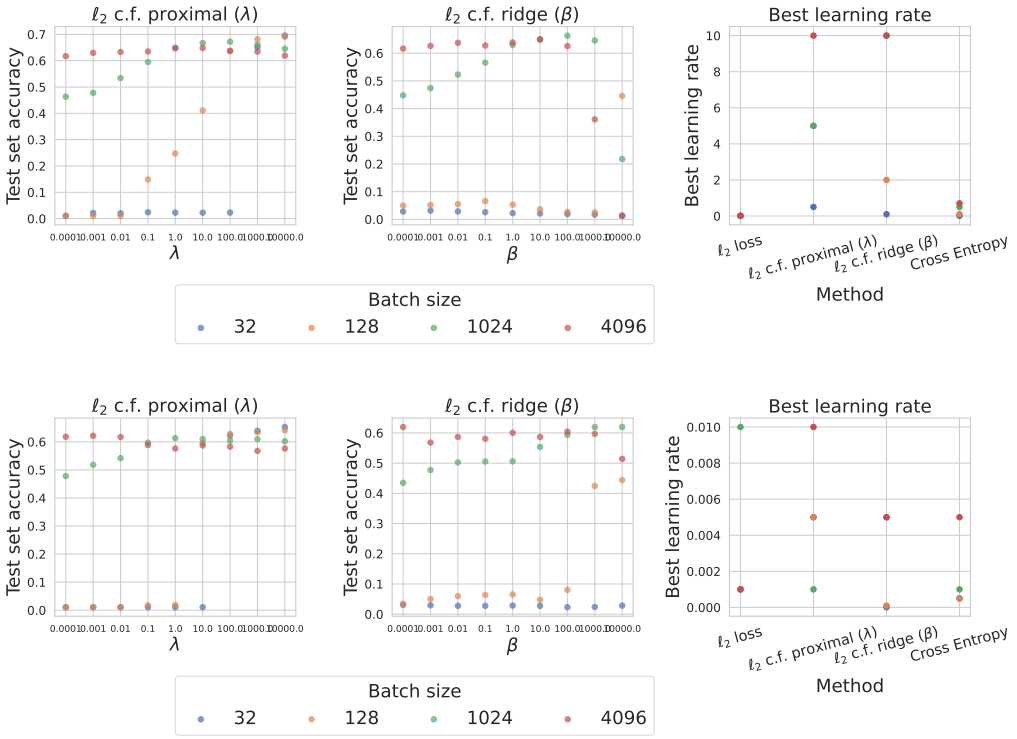


Figure G.2: **Dependence on hyperparameters on CIFAR-100**. Top: SGD performance. Bottom: Adam performance. X-axis is the number of iterations, Y-axis is a test set accuracy. **Left**, ablation over $\lambda$ for '$\ell_2$ *c.f. proximal* ($\lambda$)". **Center**, ablation over $\beta$ for "$\ell_2$ *c.f. ridge* ($\beta$)". **Right**, the best learning rate per method.

**Vision Transformer backbone.** We additionally run our method with a Vision Transformer backbone on CIFAR-100. We trained ViT-S with patch size $(4, 4)$ (images are $32 \times 32$), following the CIFAR-100 setup above, with an SGD backbone, at two batch sizes, 128 and 1024. In Table G.2, we report test accuracy at the end of training, averaged over 3 seeds. These results are in line with what we report for ResNets on CIFAR-100.

Table G.2: **ViT-S on CIFAR-100**. Test accuracy (%) at the end of training, averaged over 3 seeds, with an SGD backbone.

| Method | $B = 128$ | $B = 1024$ |
|---|---|---|
| Cross Entropy | 45.0 | 46.2 |
| $\ell_2$ c.f. proximal ($\lambda$) | 47.4 | 50.4 |

**Iterative last-layer solves for cross entropy.** There is no closed-form solution for the last layer under the cross-entropy loss. One could use iteratively reweighted least squares (IRLS), which is equivalent to Newton steps on the last layer. We implemented a variant replacing our closed form with $K$-step L-BFGS ($K = 1, 2, 5$) on the cross-entropy loss for fixed backbone features. The setup follows the CIFAR-100 experiments above. Results are reported in Table G.3. We see that using 5 L-BFGS steps outperforms both "*Cross Entropy*" and L-BFGS with 1 step. This suggests a viable practical strategy to approximate the closed-form solution for losses other than $\ell_2$. Interestingly, this variant is still less effective than our $\ell_2$ closed-form solution on CIFAR-100.

**Large scale classification on ImageNet.** We study the performance of our approach on ImageNet. We use NF-Nets-F0 architecture [4] with batch size 4096 and the same training regime as in [4]. We used 1 seed for these experiments. See Appendix F for more details. The results are given in Figure G.3. We see that our method achieves better performance than "$\ell_2$ *loss*" loss but under-

Table G.3: **Approximating the last layer for cross entropy on CIFAR-100**. The closed-form update is replaced by $K$ steps of L-BFGS on the cross-entropy loss for fixed backbone features.

| Method | Test accuracy |
|---|---|
| Cross Entropy | 0.662 |
| $\ell_2$ c.f. proximal ($\lambda$) | 0.686 |
| Cross Entropy, last layer via 1-step L-BFGS | 0.667 |
| Cross Entropy, last layer via 2-step L-BFGS | 0.678 |
| Cross Entropy, last layer via 5-step L-BFGS | 0.679 |

performs "*Cross Entropy*". While this is contrary to our finding on CIFAR-100, ImageNet has ten times the number of classes, so is a significantly different regime. The under-performance of the methods based on the squared loss could be due to advantageous properties of the cross entropy loss in classification, or simply because the training practices with cross entropy have been greatly perfected over the years.

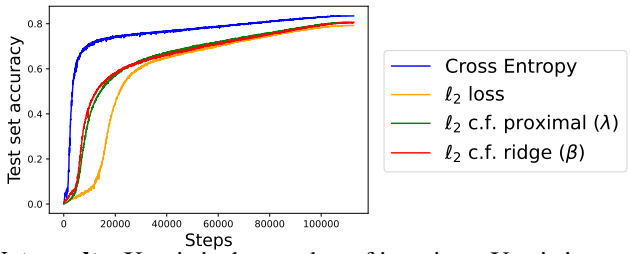


Figure G.3: **ImageNet results**. X-axis is the number of iterations, Y-axis is a test set accuracy. Each column corresponds to a different batch size. Different colors indicate different methods.

## G.2 Impact of Initialization

**Impact of initialization on QM9.** We study the impact of initialization of the last layer, discussed in Section 4.1. The results are given in Figure G.4. We see that when we use SGD optimizer, initializing with zeros seems to lead to the best performance (except for very small batch size). When using Adam optimizer, zeros initialization leads to worse results, while all other initializations lead to overall similar performance.

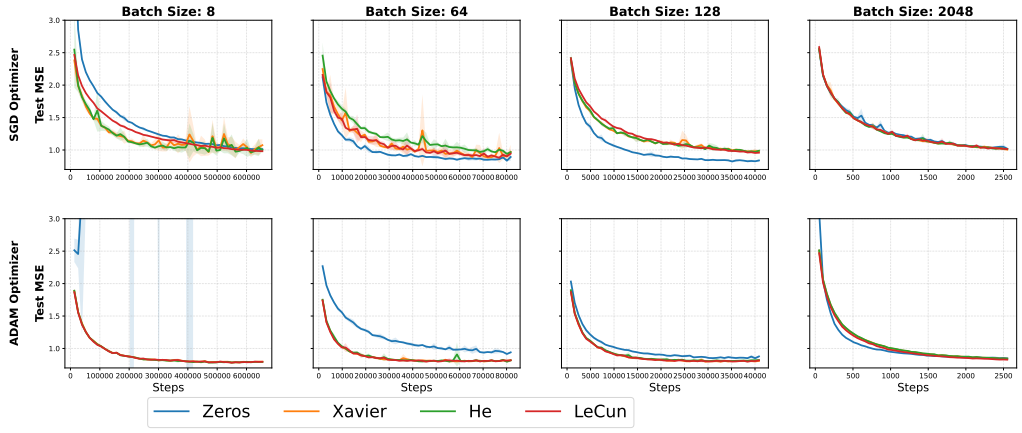


Figure G.4: **QM9, impact of initialization**. X-axis is the number of iterations, Y-axis is a test set mean squared error (MSE), columns represents different batch sizes, rows represent different backbone optimizers. Different colors indicate different ways to initialize the last layer.

**Impact of initialization on CIFAR-100.** We study impact of different initialization strategies (see Section 4.1 for more details) on CIFAR-100 when using SGD optimizer. The results are given in Figure G.5. We see that using *zero* initialization leads to overall better performance across batch sizes, which is consistent with our observations on QM9 above.

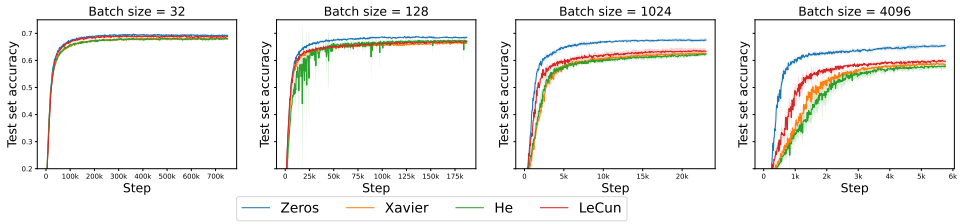


Figure G.5: **CIFAR-100, impact of initialization**. X-axis is the number of iterations, Y-axis is a test set accuracy. Each column corresponds to a different batch size. Different colors indicate different methods. We use SGD optimizer.

### G.3 Use of a Bias Term

**Use of a bias on CIFAR-100.** We study the impact of using bias on the performance of "$\ell_2$ *c.f. proximal* ($\lambda$)" on CIFAR-100. The results are given in Figure G.6. We observe similar performance for both strategies.

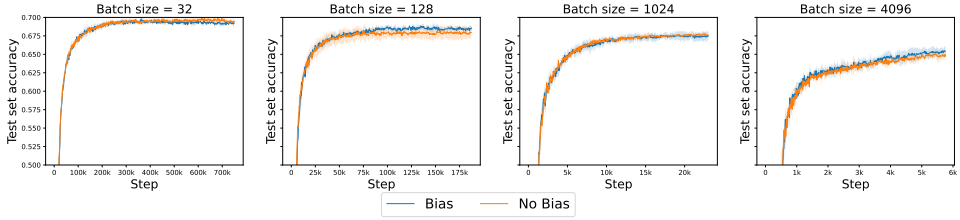


Figure G.6: **CIFAR-100, Whether to use a bias**. X-axis is the number of iterations, Y-axis is a test set accuracy. Each column corresponds to a different batch size. Different colors indicate different methods.

### G.4 Comparison of Algorithm 1 and Algorithm 2

**QM-9.** In Figure G.4, we study performance of our method when using Algorithm 1 or Algorithm 2. We see that there is not much difference in performance. The trade-off is that Algorithm 2 is conceptually easier to implement.

**CIFAR-100.** We compare the performance of Algorithm 2 and Algorithm 1 in Fig. G.8. We see that there is not much difference in performance. The trade-off is that Algorithm 2 is conceptually easier to implement.

### G.5 Optimization Dynamics

A natural concern is whether repeatedly recomputing the exact last layer introduces optimization oscillations, or slows backbone feature learning during the early stages of training. We investigate both concerns directly.

**Stability across learning rates.** Rather than introducing instability, maintaining the head near its optimum extends the stable learning-rate range: at the largest learning rate in our sweep (0.1), plain $\ell_2$ SGD training is unstable, while our method benefits from it (see Figure G.2, which reports the best learning rate per method on CIFAR-100).

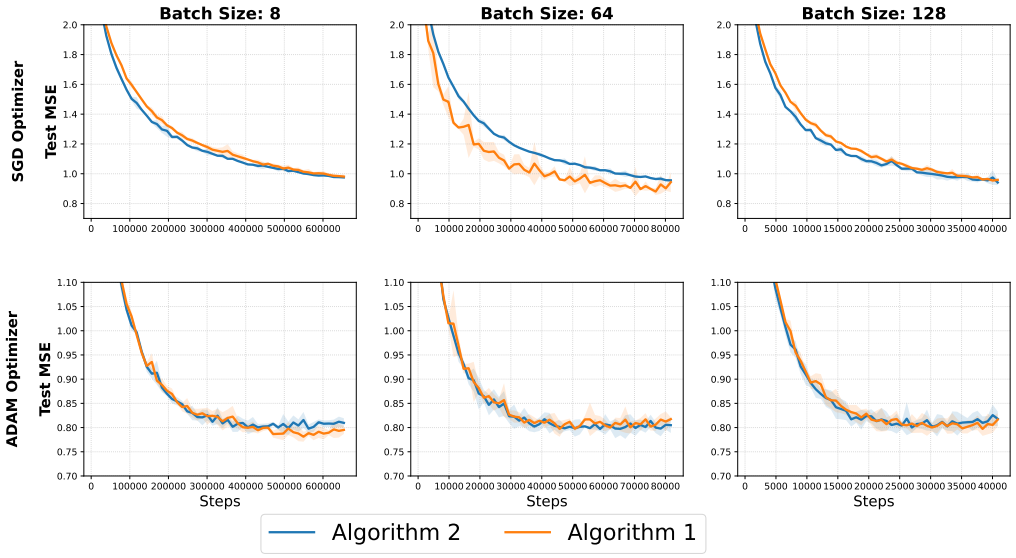


Figure G.7: **QM9 algorithms comparison**. X-axis is the number of iterations, Y-axis is a test set mean squared error (MSE), columns represents different batch sizes, rows represent different backbone optimizers. Different colors indicate different algorithms. We use SGD optimizer.

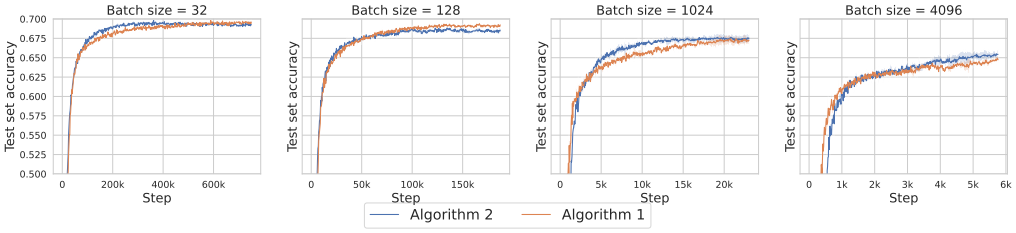


Figure G.8: **Comparison of Algorithm 1 and Algorithm 2**. X-axis is the number of iterations, Y-axis is a test set accuracy. A column indicates a batch size while a color represents an algorithm.

**Backbone dynamics diagnostics.** On QM9 at batch size 128 we compare "$\ell_2$ *c.f. proximal* ($\lambda$)" ($\lambda = 1000$) to "$\ell_2$ *loss*", both with SGD at learning rate 0.005 (optimal for "$\ell_2$ *loss*"); our method reaches test MSE 0.836 vs. 1.036. In Table G.4, we report the backbone gradient-norm ratio (ours / "$\ell_2$ *loss*"), averaged over three stages of training (under matched SGD the update norm is proportional, so it coincides). The backbone thus receives a learning signal of the same magnitude as under standard end-to-end training throughout—including early training—while reaching lower loss. We find no evidence of slowed feature learning.

Table G.4: **Backbone gradient-norm ratio** on QM9 ("$\ell_2$ *c.f. proximal* ($\lambda$)" / "$\ell_2$ *loss*"), batch size 128, SGD with learning rate 0.005, averaged over three windows of training steps.

| Window | 0–1k | 5k–6k | final 1k |
|---|---|---|---|
| Grad-norm ratio | 0.82 | 1.03 | 1.12 |

**Oscillations.** In the same setting, we measure the cosine similarity between consecutive backbone gradients over the first 2k steps: 0.08 for our method vs. 0.10 for SGD. The directional coherence of the backbone gradients is indistinguishable between the two: recomputing the last layer does not measurably perturb backbone optimization.

### G.6 Adaptive Proximal Coefficient

Our method is not very sensitive to $\lambda$ provided it is sufficiently large (see Figure G.2 and the study of $\lambda$ on QM9 in Section 6). We additionally implemented an adaptive strategy for $\lambda$ driven by the optimization dynamics—specifically, by how fast the backbone features are changing.

At training step $t$, as the backbone parameters update from $\theta_{t-1}$ to $\theta_t$, we evaluate the minibatch features under both, $\phi_{\theta_{t-1}}(X_t)$ and $\phi_{\theta_t}(X_t)$, and measure the relative feature drift

$$\delta_t = \frac{\|\phi_{\theta_t}(X_t) - \phi_{\theta_{t-1}}(X_t)\|_F}{\|\phi_{\theta_t}(X_t)\|_F + \varepsilon}, \qquad \varepsilon = 10^{-8}. \tag{72}$$

We then set $\lambda_t = \lambda_0/(\delta_t + s)$, with $s > 0$, and use $\lambda_t$ in place of $\lambda$ in the proximal closed-form update. Large drift (early training, large backbone steps) yields a smaller $\lambda_t$, relaxing the anchor to the previous last layer estimate so the last layer can track the changing features; small drift (late training, converged features) yields a larger $\lambda_t \approx \lambda_0/s$, stabilizing the last layer against minibatch noise. This adds one extra forward pass per step. We used $\lambda_0 = 1000$.

Results on QM9 are reported in Table G.5. The adaptive rule matches the tuned fixed $\lambda$ under SGD and yields a modest improvement under Adam.

Table G.5: **Adaptive proximal coefficient on QM9**. Test MSE at batch size 128, averaged over 3 random seeds.

| | | $\ell_2$ c.f. proximal | |
|---|---|---|---|
| Optimizer | $\ell_2$ loss | fixed $\lambda$ | adaptive $\lambda_t$ |
| SGD | 1.05 | 0.84 | 0.84 |
| Adam | 0.811 | 0.818 | 0.795 |

### G.7 Additional Ablations on QM9

**L-BFGS backbone optimizer.** In the main paper, we compare under both SGD and Adam backbone optimizers. We additionally ran L-BFGS as the backbone optimizer on QM9, using the standard `optax` library (disabling line search). We use batch size 128 and the same learning-rate sweep as in the main paper, and we report test MSE averaged over 3 seeds in Table G.6. Plain "$\ell_2$ *loss*" training did not reach acceptable performance with L-BFGS at this batch size across the learning-rate sweep. Notably, our method trains stably and reaches performance comparable to its SGD and Adam results, indicating that the improvements are not tied to a particular backbone optimizer.

Table G.6: **L-BFGS backbone optimizer on QM9**. Test MSE at batch size 128, averaged over 3 seeds.

| Method | Test MSE |
|---|---|
| $\ell_2$ loss | 12.6 |
| $\ell_2$ c.f. proximal ($\lambda$) | 0.8 |

**MLP width, depth and last-layer dimension.** We run ablations over the MLP size on QM9, varying the width and the depth of the backbone, as well as the dimension of the last hidden layer (i.e., the last-layer feature dimension $d$). We use batch size 128 and 3 random seeds, and report test MSE in Table G.7. Across all configurations our method trains stably, and it improves upon the baseline in every configuration except the two deepest backbones ($[256] \times 6$ and $[256] \times 9$), where both methods perform comparably. The improvement persists at small widths, far from any NTK-like regime, and we observed no optimization pathologies (divergence, oscillations, loss spikes) at any configuration. The corresponding compute and memory costs are analysed in Appendix I.

Table G.7: **MLP size ablations on QM9**. Test MSE at batch size 128, averaged over 3 seeds. Hidden dimensions list the widths of the hidden layers.

| Ablation | Hidden dims | $\ell_2$ loss | $\ell_2$ c.f. proximal ($\lambda$) |
|---|---|---|---|
| Width | $[64] \times 3$ | 1.19 | 1.07 |
| Width | $[256] \times 3$ | 1.08 | 0.87 |
| Width | $[1024] \times 3$ | 1.08 | 0.82 |
| Width | $[4096] \times 3$ | 1.43 | 0.80 |
| Depth | $[256] \times 2$ | 1.74 | 0.93 |
| Depth | $[256] \times 3$ | 1.11 | 0.88 |
| Depth | $[256] \times 6$ | 0.91 | 0.93 |
| Depth | $[256] \times 9$ | 0.92 | 0.93 |
| Last layer dim | $[256, 256, 64]$ | 1.15 | 0.88 |
| Last layer dim | $[256, 256, 256]$ | 1.08 | 0.83 |
| Last layer dim | $[256, 256, 1024]$ | 1.62 | 0.83 |
| Last layer dim | $[256, 256, 4096]$ | 1.64 | 0.82 |

# H Wall-Clock Time Analysis on CIFAR-100

In order to understand the behavior of the proposed method in real scenarios, we conduct a wall-clock time analysis on CIFAR-100. We consider ResNet18 and ResNet50 backbones, where we add an additional last layer of a given dimensionality (**Last Layer Dim**). We report steps per second (**SPS**) metrics as well as total time (**Time**) for training a model for 100 epochs on CIFAR-100 using either our method (Algorithm 1 or Algorithm 2) or cross entropy (CE). We report metrics for different batch sizes and last layer dimensions. Moreover, we report **rSPS** = SPS(Our method) / SPS(CE) (higher means our method is faster than CE) and **rTime** = Time(Our method) / Time (CE) (lower means our method is faster than CE). We use A100 GPU. The results for ResNet-18 are given in Table H.1 and the results for ResNet-50 are given in Table H.2.

**Take-aways.** First, we observe that for small batch size (32), **rTime** of our method increases from 0.92 (last layer dim = 128) to 2.37 (last layer dim = 4096) for ResNet18, and from 1.66 (last layer dim = 128) to 1.84 (last layer dim = 4096) for ResNet50. This means that for small batch sizes, as we increase the last layer dimension, our method becomes significantly slower than cross entropy. However, we also notice that the **rTime** decreases as we increase the model size from ResNet18 to ResNet50. This highlights the fact that as the model size increases, the computation required for the last layer becomes relatively smaller compared to the computation of the backbone.

Second, we observe that for large batch size (1024), **rTime** only increases from 1.09 (last layer dim = 128) to 1.18 (last layer dim = 4096) for ResNet18; and from 1.16 (last layer dim = 128) to 1.20 (last layer dim = 4096). Finally, for ResNet18, we see that for batch size = 4096, **rTime** basically stays very similar (from 1.12 to 1.14). This highlights that in the large batch size regime, our method is roughly 10-15% slower than cross entropy.

Table H.1: Wall clock time comparisons on ResNet-18 with A100 GPU.

| Batch Size | Last Layer Dim | SPS (Ours) | SPS (CE) | rSPS | Time (Ours) | Time (CE) | rTime |
|---|---|---|---|---|---|---|---|
| 32 | 128 | 89.98 | 81.02 | 1.11 | 2164.20 | 2348.11 | 0.92 |
| 32 | 256 | 87.35 | 78.97 | 1.11 | 2214.96 | 2400.73 | 0.92 |
| 32 | 512 | 78.48 | 79.86 | 0.98 | 2413.57 | 2374.19 | 1.02 |
| 32 | 1024 | 70.23 | 81.89 | 0.86 | 2633.73 | 2325.33 | 1.13 |
| 32 | 2048 | 51.47 | 79.66 | 0.65 | 3426.96 | 2383.31 | 1.44 |
| 32 | 4096 | 30.41 | 82.44 | 0.37 | 5488.86 | 2318.55 | 2.37 |
| 128 | 128 | 40.57 | 41.24 | 0.98 | 1293.17 | 1277.97 | 1.01 |
| 128 | 256 | 39.59 | 40.74 | 0.97 | 1320.46 | 1281.52 | 1.03 |
| 128 | 512 | 38.77 | 40.72 | 0.95 | 1333.75 | 1282.12 | 1.04 |
| 128 | 1024 | 35.47 | 40.44 | 0.88 | 1422.86 | 1291.61 | 1.10 |
| 128 | 2048 | 29.76 | 41.14 | 0.72 | 1614.35 | 1278.58 | 1.26 |
| 128 | 4096 | 20.92 | 40.54 | 0.52 | 2140.42 | 1294.91 | 1.65 |
| 1024 | 128 | 4.66 | 5.07 | 0.92 | 1055.30 | 967.64 | 1.09 |
| 1024 | 256 | 4.68 | 5.04 | 0.93 | 1046.71 | 975.81 | 1.07 |
| 1024 | 512 | 4.68 | 5.09 | 0.92 | 1050.68 | 966.34 | 1.09 |
| 1024 | 1024 | 4.66 | 5.05 | 0.92 | 1051.08 | 972.14 | 1.08 |
| 1024 | 2048 | 4.57 | 5.10 | 0.89 | 1072.17 | 963.36 | 1.11 |
| 1024 | 4096 | 4.21 | 5.02 | 0.84 | 1164.10 | 982.72 | 1.18 |
| 4096 | 128 | 1.11 | 1.24 | 0.89 | 1084.92 | 966.99 | 1.12 |
| 4096 | 256 | 1.09 | 1.24 | 0.88 | 1097.92 | 968.55 | 1.13 |
| 4096 | 512 | 1.08 | 1.24 | 0.87 | 1105.35 | 963.45 | 1.15 |
| 4096 | 1024 | 1.10 | 1.23 | 0.90 | 1089.28 | 973.90 | 1.12 |
| 4096 | 2048 | 1.09 | 1.23 | 0.89 | 1100.39 | 974.79 | 1.13 |
| 4096 | 4096 | 1.08 | 1.24 | 0.88 | 1107.10 | 971.15 | 1.14 |

Table H.2: Wall clock time comparisons on ResNet-50 with A100 GPU.

| Batch Size | Last Layer Dim | SPS (Ours) | SPS (CE) | rSPS | Time (Ours) | Time (CE) | rTime |
|---|---|---|---|---|---|---|---|
| 32 | 128 | 36.96 | 42.94 | 0.86 | 4735.09 | 2855.43 | 1.66 |
| 32 | 256 | 41.18 | 42.84 | 0.96 | 4240.65 | 4056.31 | 1.05 |
| 32 | 512 | 35.85 | 39.59 | 0.91 | 4903.87 | 3496.42 | 1.40 |
| 32 | 1024 | 35.22 | 43.28 | 0.81 | 4903.89 | 4002.80 | 1.23 |
| 32 | 2048 | 30.85 | 41.52 | 0.74 | 5482.36 | 4173.49 | 1.31 |
| 32 | 4096 | 21.68 | 41.97 | 0.52 | 7605.07 | 4133.77 | 1.84 |
| 128 | 128 | 16.53 | 18.62 | 0.89 | 2645.24 | 2384.37 | 1.11 |
| 128 | 256 | 16.37 | 18.49 | 0.89 | 2659.99 | 2397.10 | 1.11 |
| 128 | 512 | 16.15 | 18.30 | 0.88 | 2698.11 | 2423.22 | 1.11 |
| 128 | 1024 | 15.45 | 18.52 | 0.83 | 2801.06 | 2405.96 | 1.16 |
| 128 | 2048 | 14.31 | 18.44 | 0.78 | 3000.61 | 2409.51 | 1.25 |
| 128 | 4096 | 11.95 | 18.05 | 0.66 | 3532.58 | 2448.23 | 1.44 |
| 1024 | 128 | 2.26 | 2.59 | 0.87 | 2154.44 | 1875.26 | 1.15 |
| 1024 | 256 | 2.25 | 2.61 | 0.86 | 2151.89 | 1866.88 | 1.15 |
| 1024 | 512 | 2.24 | 2.60 | 0.86 | 2163.26 | 1874.76 | 1.15 |
| 1024 | 1024 | 2.23 | 2.59 | 0.86 | 2174.04 | 1876.25 | 1.16 |
| 1024 | 2048 | 2.19 | 2.57 | 0.85 | 2216.16 | 1889.88 | 1.17 |
| 1024 | 4096 | 2.12 | 2.56 | 0.83 | 2286.04 | 1898.44 | 1.20 |

# I Wall-Clock Time, Memory Footprint, and Scalability Analysis on QM9

To complement the CIFAR-100 timings in Appendix H, we evaluate how the wall-clock step time and peak GPU memory of closed-form last-layer optimization scale on QM9 [24]. We vary five dimensions of the network $f(x; W, \theta) = W\phi_\theta(x)$: backbone depth $L$, backbone hidden width $H$, last-layer feature dimension $d$ ($\phi_\theta(x) \in \mathbb{R}^d$), batch size $B = |\mathcal{B}_t|$, and output dimension $o$ ($W \in \mathbb{R}^{o\times d}$).

Following Appendix H, we report step time (ms), **rTime** $=$ Time(Ours)/Time($\ell_2$ loss), relative overhead (**rTime** $-1$) $\times 100\%$, and peak GPU memory (GiB).

### I.1 Computational Cost: Algorithm 1 vs. Algorithm 2

On a batch $\mathcal{B}_t$ of size $B$, let $\Phi_t := \phi_\theta(X_t) \in \mathbb{R}^{d\times B}$ denote the penultimate features. A standard gradient step on "$\ell_2$ *loss*" performs one forward pass ($T_{\text{fwd}}$) and one backward pass ($T_{\text{bwd}} \approx 2\,T_{\text{fwd}}$, since each layer computes two matrix multiplications: one for weight gradients $\nabla_{W_\ell}\mathcal{L}$ and one for activation gradients $\nabla_{h_{\ell-1}}\mathcal{L}$), totaling $T_{\text{backbone}} = T_{\text{fwd}} + T_{\text{bwd}} \approx 3\,T_{\text{fwd}} = \mathcal{O}(LBH^2 + BHd + Bdo)$ FLOPs. Solving the closed-form normal equations (14) via Cholesky factorization costs

$$T_{\text{chol}}(B,d,o) = \underbrace{Bd^2}_{\text{Gram } \Phi_t\Phi_t^\top} + \underbrace{\tfrac{1}{3}d^3}_{\text{Cholesky of } \Phi_t\Phi_t^\top+\lambda I_d} + \underbrace{2Bdo}_{\text{RHS } Y_t\Phi_t^\top+\lambda W_t} + \underbrace{2d^2o}_{\text{Triangular solve}}. \tag{73}$$

This decomposition explains four practical scaling properties:

1. **Algorithm 1 (1 forward pass) vs. Algorithm 2 (2 forward passes).** In Algorithm 2, $\theta_t$ is updated before evaluating $W_t = W^\star_{\mathcal{B}_t, W_{t-1}}(\theta_t)$, which requires a second forward pass $\phi_{\theta_t}(X_t)$ on $\mathcal{B}_t$. Since $T_{\text{backbone}} \approx 3\,T_{\text{fwd}}$, this extra forward pass adds $+1\,T_{\text{fwd}}$, creating an asymptotic $+33.3\%$ overhead floor (**rTime** $\to 4/3$) as $L \to \infty$. In contrast, Algorithm 1 evaluates both updates at the same backbone parameters (or reuses the cached features $\Phi_t$ from the forward pass), requiring only **one forward pass per batch**. Its relative overhead therefore decays to zero with depth $L$:

$$\textbf{rTime}_{\text{Algorithm 1}}(L,H,d) - 1 = \frac{T_{\text{chol}}(B,d,o)}{3\,T_{\text{fwd}}(L,H,B)} = \mathcal{O}\!\left(\frac{Bd^2+d^3}{LBH^2}\right) \xrightarrow{L\to\infty} 0\%. \tag{74}$$

2. **Bottleneck width decoupling ($H \gg d$).** Because $T_{\text{chol}}(B,d,o)$ depends only on the last-layer dimension $d$ and not on the hidden width $H$, projecting a wide backbone ($H \in [2048, 8192]$) to a bottleneck dimension $d = 256$ keeps the Cholesky solve under $0.25$ ms on an A100 GPU.

3. **Invariance to output dimension $o$.** Neither the Gram matrix $\Phi_t\Phi_t^\top + \lambda I_d \in \mathbb{R}^{d\times d}$ nor its $\mathcal{O}(d^3)$ Cholesky factorization depends on $o$. Only the right-hand side ($\mathcal{O}(Bdo)$) and triangular back-substitution ($\mathcal{O}(d^2o)$) depend on $o$.

4. **Negligible memory overhead.** By Proposition 4.1, autodiff does not backpropagate through $(\Phi_t\Phi_t^\top + \lambda I_d)^{-1}$. The only extra memory is the $d \times d$ Gram matrix (256 KB at $d = 256$, and 64 MB at $d = 4096$).

### I.2 Empirical Scaling Results on A100 GPUs

Figures I.1 and I.2 and Table I.1 summarize the benchmark results across depth $L$, bottleneck width $H$, and output dimension $o$:

- **Depth $L$ (Fig. I.1a, Fig. I.2a)**: At $d = 256$ ($H = 2048, B = 2048$), the overhead of Algorithm 1 drops from $+24.5\%$ at $L = 4$ to $+14.5\%$ at $L = 8$, $+7.6\%$ at $L = 16$, and $+3.6\%$ (**rTime** $= 1.04$) at $L = 32$. At $d = 512$, overhead drops to $+7.7\%$ at $L = 32$. Even at $d = 4096$, increasing depth from $L = 1$ to $L = 32$ reduces relative overhead by $11\times$ ($+1078\% \to +98.5\%$).
- **Wide backbones with $d = 256$ bottleneck (Fig. I.1b, Fig. I.2b)**: At $H = 4096, L = 16$ (254.6M parameters) and $H = 8192, L = 8$ (475.5M parameters), Algorithm 1 adds only $+4.5\%$ and $+4.3\%$ step-time overhead, respectively, while peak GPU memory increases by just $+0.7\%$ ($18.08 \to 18.20$ GiB).
- **Output dimension $o$ (Fig. I.1c, Fig. I.2c)**: Increasing $o$ from 1 to 4096 at $L = 32$ ($H = 1024, d = 256$) changes the overhead of Algorithm 1 from $+9.3\%$ to $+10.6\%$.

### I.3 Shallow Backbones ($L = 3$) and End-to-End QM9 Training

Finally, Figures I.3 and I.4 examine shallow 3-layer MLPs ($L = 3$) across $d \in [64, 4096]$. When $d \le 512$, Algorithm 2 and Algorithm 1 add $< 0.5\%$ peak memory and low step-time overhead (**rTime** $\in [0.97, 1.21]$ at $B = 128$) while achieving 20%–50% lower test MSE and reaching the

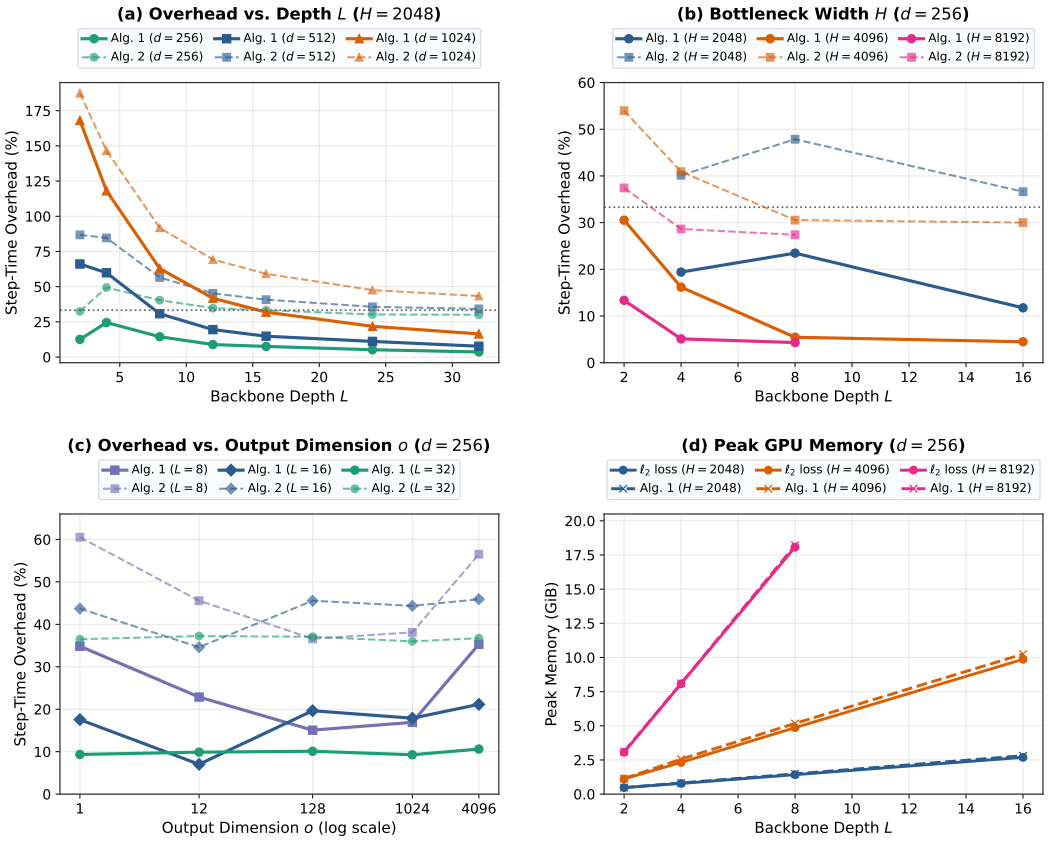


Figure I.1: **Scalability summary on QM9 (A100 GPU). (a)** Relative step-time overhead vs. depth $L$ ($H = 2048, B = 2048$): Algorithm 2 plateaus at $+33.3\%$, whereas Algorithm 1 decays as $\mathcal{O}(1/L)$ to $+3.6\%$ at $L = 32, d = 256$. **(b)** Wide backbones ($H \in [2048, 8192]$) with a $d = 256$ bottleneck achieve $< 5\%$ **overhead** with Algorithm 1. **(c)** Scaling the output dimension $o$ by $4096\times$ ($o = 1 \to 4096$) changes the overhead of Algorithm 1 by only $+1.3\%$ at $L = 32$. **(d)** Peak GPU memory of Algorithm 1 is nearly identical to "$\ell_2$ *loss*" ($< 1.5\%$ overhead).

final test MSE of "$\ell_2$ *loss*" in $2\times$–$4\times$ less wall-clock time. When $d \geq 2048$ on a shallow network ($L = 3$), the $\mathcal{O}(d^3)$ Cholesky factor dominates step time; this regime is avoided by using either a deeper backbone (Fig. I.2a) or a $d = 256$ bottleneck layer (Fig. I.2b).

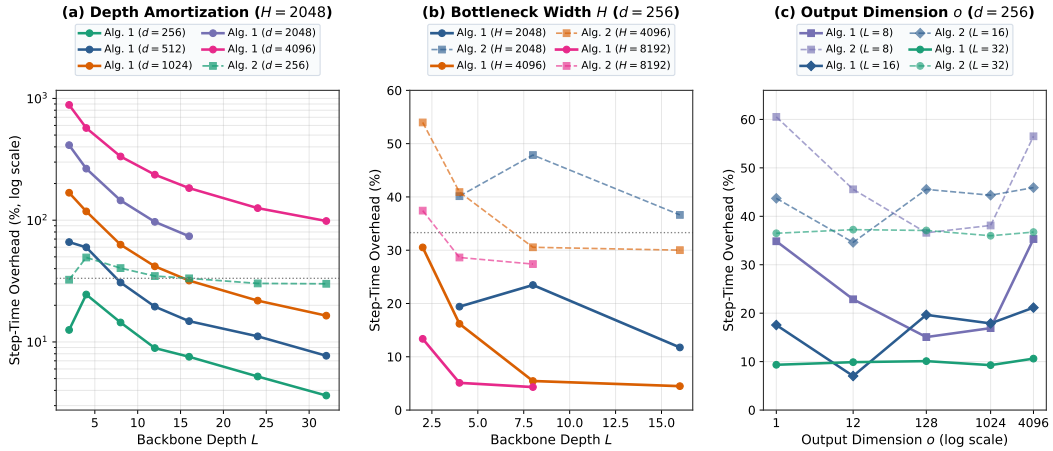


Figure I.2: **Detailed scaling curves on A100 GPU. (a)** Log-scale overhead vs. depth $L$ across $d \in \{256, 512, 1024, 2048, 4096\}$. **(b)** Bottleneck regime ($d = 256$) across hidden widths $H \in \{2048, 4096, 8192\}$. **(c)** Scaling with output dimension $o \in \{1, 12, 128, 1024, 4096\}$.

Table I.1: **Wall-clock step time, rTime, and peak GPU memory on QM9 (A100 GPU).** Comparison of "$\ell_2$ *loss*" (SGD), Algorithm 2, and Algorithm 1.

| Regime | $H$ | $d$ | $L$ | Params | $\ell_2$ loss | Algorithm 2 (rTime) | Algorithm 1 (rTime) | Overhead (Algorithm 1) | Memory (GiB) |
|---|---|---|---|---|---|---|---|---|---|
| Depth $L$ ($B = 2048$, $o = 12$) | 2048 | 256 | 4 | 14.0M | 2.72 ms | 4.07 ms (1.49×) | **3.39 ms (1.25×)** | +24.5% | 0.81 → 0.87 |
| | 2048 | 256 | 8 | 30.8M | 5.29 ms | 7.43 ms (1.40×) | **6.06 ms (1.15×)** | +14.5% | 1.44 → 1.50 |
| | 2048 | 256 | 16 | 64.4M | 10.66 ms | 14.20 ms (1.33×) | **11.46 ms (1.08×)** | +7.6% | 2.69 → 2.81 |
| | 2048 | 256 | 32 | 131.5M | 21.35 ms | 27.76 ms (1.30×) | **22.12 ms (1.04×)** | +3.6% | 5.33 → 5.46 |
| | 2048 | 512 | 32 | 132.0M | 21.41 ms | 28.73 ms (1.34×) | **23.06 ms (1.08×)** | +7.7% | 5.35 → 5.44 |
| | 2048 | 1024 | 32 | 133.1M | 21.60 ms | 30.96 ms (1.43×) | **25.15 ms (1.16×)** | +16.4% | 5.37 → 5.53 |
| | 2048 | 2048 | 32 | 135.2M | 21.84 ms | 36.31 ms (1.66×) | **30.42 ms (1.39×)** | +39.3% | 5.45 → 5.61 |
| | 2048 | 4096 | 32 | 139.4M | 22.55 ms | 50.48 ms (2.24×) | **44.77 ms (1.99×)** | +98.5% | 5.61 → 6.07 |
| Bottleneck ($d = 256$, $B = 1024$) | 4096 | 256 | 8 | 120.3M | 9.87 ms | 12.89 ms (1.31×) | **10.41 ms (1.05×)** | +5.5% | 4.86 → 5.17 |
| | 4096 | 256 | 16 | 254.6M | 20.30 ms | 26.39 ms (1.30×) | **21.21 ms (1.04×)** | +4.5% | 9.87 → 10.25 |
| | 8192 | 256 | 4 | 207.0M | 14.86 ms | 19.11 ms (1.29×) | **15.62 ms (1.05×)** | +5.1% | 8.07 → 8.11 |
| | 8192 | 256 | 8 | 475.5M | 33.54 ms | 42.73 ms (1.27×) | **34.99 ms (1.04×)** | +4.3% | 18.08 → 18.20 |
| Output $o$ ($H = 1024, L = 32$) | $o = 1$ ($d = 256, B = 2048$) | | | 33.8M | 7.66 ms | 10.45 ms (1.36×) | **8.37 ms (1.09×)** | +9.3% | 1.43 → 1.49 |
| | $o = 4096$ ($d = 256, B = 2048$) | | | 34.9M | 8.06 ms | 11.02 ms (1.37×) | **8.91 ms (1.11×)** | +10.6% | 1.54 → 1.62 |

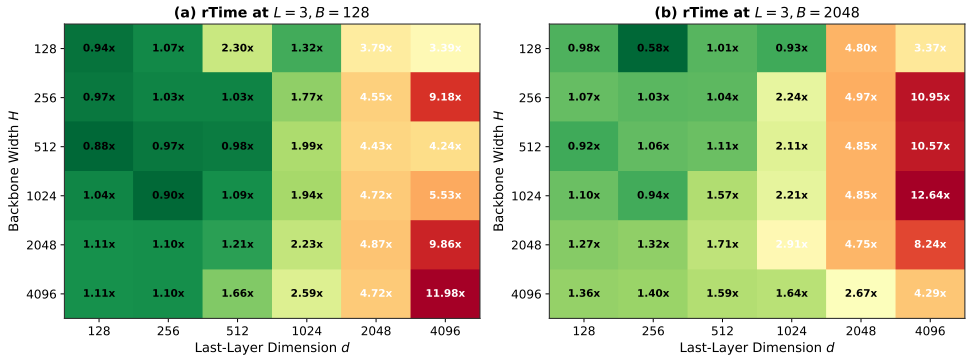


Figure I.3: **Shallow-backbone ($L = 3$) rTime heatmaps across $(H, d)$ at $B = 128$ and $B = 2048$.** For $d \leq 512$, **rTime** remains near $1.0\times$.

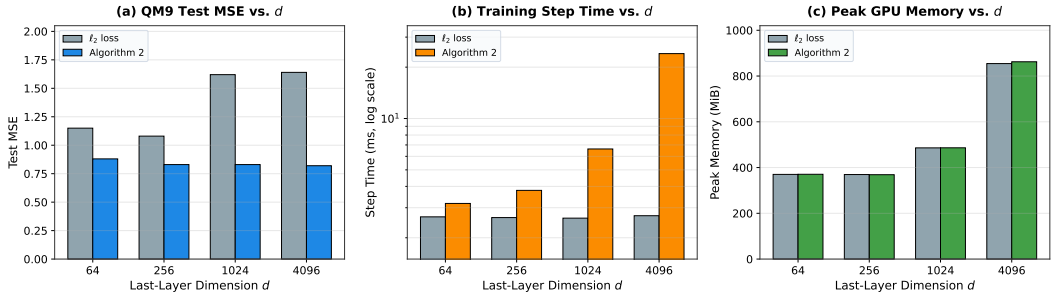


Figure I.4: **End-to-end QM9 training ablations** ($H = 256, L = 3, B = 128$)**. (a)** Test MSE vs. $d$. **(b)** Training step time vs. $d$. **(c)** Peak GPU memory vs. $d$.

## NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

**The checklist answers are an integral part of your paper submission.** They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

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 theoretical claims are verified in Sections 3 to 5 as well as Appendix C, while the experimental claims are verified in Section 6 as well as in Appendix G.

   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: Limitations are discussed in Section 7.

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: The assumptions are described in the main text (Sections 3 to 5), and any remaining proofs are in the appendix (Appendix C).

   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: Details on the experiments can be found in Section 6 and Appendix F.

   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: [No]

   Justification: While we do not include the code implementation, we include extensive experimental details in Appendix F. Moreover, implementing all the algorithms from the paper is straightforward by following the steps in Algorithm 1 or in Algorithm 2, or in Algorithm 3.

   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: All experimental details can be found in Section 6 and Appendix F.

   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: Shaded areas are used as means of error bars in Section 6.

   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 regarding the hardware in Appendix F. Moreover, every experiment has an extensive amount of details about number of iterations (epochs) and the considered hyperparameters.

   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: We fulfill all requirements.

   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: Our paper conducts no societally harmful research.

    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: There is no risk of misuse associated to the models used in the present work.

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 credit the creators of each dataset used in Section 6 and Appendix G.

    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: [N/A]

    Justification: We do not release new assets.

    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: We do not use crowdsourcing.

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: We do not conduct experiments on humans.

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: We have not made use of LLMs in any significant way.

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.