Proximal Mean Field Learning in Shallow Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

We propose a custom learning algorithm for shallow over-parameterized neural networks, i.e., networks with single hidden layer having infinite width. The infinite width of the hidden layer serves as an abstraction for the over-parameterization. Building on the recent mean field interpretations of learning dynamics in shallow neural networks, we realize mean field learning as a computational algorithm, rather than as an analytical tool. Specifically, we design a Sinkhorn regularized proximal algorithm to approximate the distributional flow for the learning dynamics over weighted point clouds. In this setting, a contractive fixed point recursion computes the time-varying weights, numerically realizing the interacting Wasserstein gradient flow of the parameter distribution supported over the neuronal ensemble. An appealing aspect of the proposed algorithm is that the measure-valued recursions allow meshless computation. We demonstrate the proposed computational framework of interacting weighted particle evolution on binary and multi-class classification. Our algorithm performs gradient descent of the free energy associated with the risk functional.

1 Introduction

While universal function approximation theorems for neural networks have long been known (Cybenko, 1989; Barron, 1993; Hornik et al., 1989), such guarantees do not account for the dynamics of the learning algorithms. Starting in 2018, several works (Mei et al., 2018; Chizat & Bach, 2018; Rotskoff & Vanden-Eijnden, 2018; Sirignano & Spiliopoulos, 2020; Rotskoff & Vanden-Eijnden, 2022; Boursier et al., 2022) pointed out that the first order learning dynamics for shallow (i.e., single hidden layer) neural networks in the infinite width (i.e., over-parameterization) limit leads to a nonlinear partial differential equation (PDE) that depends on a pair of advection and interaction potentials.

The Cauchy initial value problem associated with the PDE describes the evolution of neuronal parameter ensemble induced by the learning dynamics. This result can be interpreted as a dynamical version of the universal approximation theorem. In particular, the potentials depend on both the loss function as well as the activation functions of the neural network.

The advection potential in this nonlinear PDE induces a drift, while the interaction potential induces a nonlocal force. Remarkably, this PDE can be interpreted as an infinite dimensional gradient flow of the population risk w.r.t. the Wasserstein metric arising from the theory of optimal mass transport (Villani, 2009; 2021).

The nonlocal nonlinear PDE interpretation makes connection with the so-called "propagation of chaos"–a term due to Kac (Kac, 1956) that has grown into a substantial literature in statistical physics (McKean Jr, 1966; Sznitman, 1991; Carmona & Delarue, 2018). From this viewpoint, the first order algorithmic dynamics makes the individual neurons in the hidden layer behave as interacting particles. These particle-level or microscopic interactions manifest as a population-level or macroscopic gradient flow.

As an analytic tool, the mean field Wasserstein gradient flow interpretation helps shed light on the convergence of first order learning dynamics in over-parameterized networks. In this work, we propose Wasserstein proximal recursions to realize the mean field learning dynamics as a meshless computational algorithm.

1.1 Computational challenges

Transcribing the mean field Wasserstein gradient flow PDE from an analytical tool to a computational algorithm is particularly challenging in the neural network context. This is because the derivation of the PDE in (Mei et al., 2018; Chizat & Bach, 2018; Rotskoff & Vanden-Eijnden, 2018; Sirignano & Spiliopoulos, 2020), and the corresponding infinite dimensional gradient descent interpretation, is an asymptotic consistency result. Specifically, the PDE describes the time evolution of the joint population density (or population measure in general) for the hidden layer neuronal ensemble. By definition, this interpretation is valid in the mean field (infinite width) limit of the single hidden layer. In other words, to leverage the gradient flow PDE perspective in computation, the number of neurons in the hidden layer must be large.

However, from a numerical perspective, explicitly evolving the joint population in the large width regime is problematic. This is because the large width implies that the time-varying joint neuronal population distributions have high dimensional supports. Even though existing software tools routinely deploy stochastic gradient descent (SGD) algorithms at the *microscopic* (i.e., particle) level, it is practically infeasible to estimate the time-varying population distributions using Monte Carlo or other *a posteriori* function approximation algorithms near this limit. One also cannot resort to standard finite difference-type discretization approach for solving this gradient flow PDE because the large width limit brings the curse of dimensionality. Therefore, it is not obvious whether the mean field dynamics can lead to a learning algorithm in practice.

1.2 Related works

Beyond mean field learning, Wasserstein gradient flows appear in many other scientific (Ambrosio et al., 2005; Santambrogio, 2017) and engineering (Halder & Georgiou, 2017; Caluya & Halder, 2021b; Halder et al., 2022) contexts. Thus, there exists a substantial literature on numerically implementing the Wasserstein gradient flows – both with grid (Peyré, 2015; Benamou et al., 2016; Carlier et al., 2017; Carrillo et al., 2022) and without grid (Liu et al., 2019; Caluya & Halder, 2019a; Halder et al., 2020). The latter class of algorithms are more relevant for the mean field learning context since the underlying parameter space (i.e., the support) is high dimensional. The proximal recursion we consider is closely related to the forward or backward discretization (Salim et al., 2020; Frogner & Poggio, 2020) of the Jordan-Kinderlehrer-Otto (JKO) scheme (Jordan et al., 1998).

To bypass numerical optimization over the manifold of measures or densities, recent works (Mokrov et al., 2021; Alvarez-Melis et al., 2021; Bunne et al., 2022) propose using input convex neural networks (Amos et al., 2017) to perform the Wasserstein proximal time-stepping by learning the convex potentials (Brenier, 1991) associated with the respective pushforward maps. To alleviate the computational difficulties in high dimensions, Bonet et al. (2021) proposes replacing the Wasserstein distance with the sliced-Wasserstein distance (Rabin et al., 2011) scaled by the ambient dimension.

1.3 Contributions

With respect to the related works referenced above, the main contribution of the present work is that we propose a meshless Wasserstein proximal algorithm that directly implements the macroscopic learning dynamics in a fully connected shallow network. We do so by evolving population densities as *weighted* scattered particles.

Different from Monte Carlo-type algorithms, the weight updates in our setting are done explicitly by solving a regularized dual ascent. This computation occurs within the dashed box highlighted in Fig. 1. The particles' location updates are done via *nonlocal* Euler-Maruyama. These two updates interact with each other (see Fig. 1), and together set up a discrete time-stepping scheme.

The discrete time-stepping procedure we propose, is a novel interacting particle system in the form of a meshless algorithm. Our contribution advances the state-of-the-art as it allows for evolving the neuronal population distribution in an online manner, as needed in mean field learning. This is in contrast to *a posteriori* function approximation in existing Monte Carlo methods (cf. Sec. 1.1). Explicit proximal weight

updates allows us to bypass offline high dimensional function approximation, thereby realizing mean field learning at an algorithm level.

With respect to the computational challenges mentioned in Sec. 1.1, it is perhaps surprising that we are able to design an algorithm for explicitly evolving the population densities without directly discretizing the spatial domain of the underlying PDE. Our main idea to circumvent the computational difficulty is to solve the gradient flow PDE *not* as a PDE, but to instead direct the algorithmic development for a proximal recursion associated with the gradient flow PDE. This allows us to implement the associated proximal recursion over a suitably discrete time without directly discretizing the parameter space (the latter is what makes the computation otherwise problematic in the mean field regime).

For specificity, we illustrate the proposed framework on two numerical experiments involving binary and multi-class classification with quadratic risk. The proposed methodology should be of broad interest to other problems such as the policy optimization (Zhang et al., 2018; Chu et al., 2019; Zhang et al., 2020) and the adversarial learning (Domingo-Enrich et al., 2020; Mroueh & Nguyen, 2021; Lu, 2023).

We emphasize that the perspective taken in this work is somewhat non-standard w.r.t. the existing literature in that our main intent is to explore the possibility of designing a new class of algorithms by leveraging the connection between the mean field PDE and the Wasserstein proximal operator. This is a new line of idea for learning algorithm design that we show is feasible. As such, we do not aim to immediately surpass the carefully engineered existing state-of-the-art in experiments. Instead, this study demonstrates a proof-of-concept which should inspire follow up works.

1.4 Notations and preliminaries

We use the standard convention where the boldfaced lowercase letters denote vectors, boldfaced uppercase letters denote matrices, and non-boldfaced letters denote scalars. We use the symbols ∇ and Δ to denote the Euclidean gradient and Laplacian operators, respectively. In case of potential confusion, we attach subscripts to these symbols to clarify the differential operator is taken w.r.t. which variable. The symbols $\odot, \oslash$, exp, and tanh denote *elementwise* multiplication, division, exponential, and hyperbolic tangent, respectively. Furthermore, rand and randn denote draw of the uniform and standard normal distributed random vector of appropriate dimension. In addition, $\mathbf{1}$ represents a vector of ones of appropriate dimension.

Let $\mathcal{Z}_1, \mathcal{Z}_2 \subseteq \mathbb{R}^d$. The squared 2-Wasserstein metric W_2 (with standard Euclidean ground cost) between two probability measures $\pi_1(\mathrm{d}\boldsymbol{z}_1)$ and $\pi_2(\mathrm{d}\boldsymbol{z}_2)$ (or between the corresponding densities when the measures are absolutely continuous), where $\boldsymbol{z}_1 \in \mathcal{Z}_1$, $\boldsymbol{z}_2 \in \mathcal{Z}_2$, is defined as

$$W_2^2\left(\pi_1, \pi_2\right) := \inf_{\pi \in \Pi(\pi_1, \pi_2)} \int_{\mathcal{Z}_1 \times \mathcal{Z}_2} \|\boldsymbol{z}_1 - \boldsymbol{z}_2\|_2^2 \, \mathrm{d}\pi(\boldsymbol{z}_1, \boldsymbol{z}_2). \tag{1}$$

In (1), the symbol $\Pi\left(\pi_1, \pi_2\right)$ denotes the collection of joint measures (couplings) with finite second moments, whose first marginal is π_1, and the second marginal is π_2.

1.5 Organization

The remainder of this paper is organized as follows. In Sec. 2, we provide the necessary background for the empirical risk minimization and for the corresponding mean field limit. The proposed proximal algorithm (including its derivation, convergence guarantee and implementation) is detailed in Sec. 3. We then report numerical case studies in Sec. 4 and Sec. 5 for binary and multi-class classifications, respectively. Sec. 6 concludes the paper.

2 From empirical risk minimization to proximal mean field learning

To motivate the mean field learning formulation, we start by discussing the more familiar empirical risk minimization set up. We then explain the infinite width limit for the same.

2.1 Empirical risk minimization

We consider a supervised learning problem where the dataset comprises of the features $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$, and the labels $y \in \mathcal{Y} \subseteq \mathbb{R}$, i.e., the samples of the dataset are tuples of the form

$$(\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^{n_x} \times \mathbb{R}.$$

The objective of the supervised learning problem is to find the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^p$ such that $y \approx f(\boldsymbol{x}, \boldsymbol{\theta})$ where f is some function class parameterized by $\boldsymbol{\theta}$. In other words, f maps from the feature space $\mathcal{X}$ to the label space $\mathcal{Y}$. To this end, we consider a shallow neural network with a single hidden layer having n_{H} neurons. Then, the parameterized function f admits representation

$$f(\boldsymbol{x}, \boldsymbol{\theta}) := \frac{1}{n_{\mathrm{H}}} \sum_{i=1}^{n_{\mathrm{H}}} \Phi\left(\boldsymbol{x}, \boldsymbol{\theta}_i\right), \tag{2}$$

where $\Phi\left(\boldsymbol{x}, \boldsymbol{\theta}_i\right) := a_i \sigma(\langle \boldsymbol{w}_i, \boldsymbol{x} \rangle + b_i)$ for all $i \in [n_{\mathrm{H}}] := \{1, 2, \ldots, n_{\mathrm{H}}\}$, and $\sigma(\cdot)$ is a smooth activation function. The parameters $a_i, \boldsymbol{w}_i$ and b_i are the scaling, weights, and bias of the ith hidden neuron, respectively, and together comprise the specific parameter realization $\boldsymbol{\theta}_i \in \mathbb{R}^p$, $i \in [n_{\mathrm{H}}]$.

We stack the parameter vectors of all hidden neurons as

$$\overline{\boldsymbol{\theta}} := \left(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_{n_{\mathrm{H}}}\right)^\top \in \mathbb{R}^{p n_{\mathrm{H}}}$$

and consider minimizing the following quadratic loss:

$$l(y, \boldsymbol{x}, \overline{\boldsymbol{\theta}}) \equiv l(y, f(\boldsymbol{x}, \overline{\boldsymbol{\theta}})) := \underbrace{\left(y - f(\boldsymbol{x}, \overline{\boldsymbol{\theta}})\right)^2}_{\text{quadratic loss}}. \tag{3}$$

We suppose that the training data follows the joint probability distribution γ, i.e., $(\boldsymbol{x}, y) \sim \gamma$. Define the *population risk* R as the expected loss given by

$$R(f) := \mathbb{E}_{(\boldsymbol{x}, y) \sim \gamma}[l(y, \boldsymbol{x}, \overline{\boldsymbol{\theta}})]. \tag{4}$$

In practice, γ is unknown, so we approximate the population risk with the *empirical risk*

$$R(f) \approx \frac{1}{n_{\mathrm{data}}} \sum_{j=1}^{n_{\mathrm{data}}} l\left(y_j, \boldsymbol{x}_j, \overline{\boldsymbol{\theta}}\right) \tag{5}$$

where n_{data} is the number of data samples. Then, the supervised learning problem reduces to the empirical risk minimization problem

$$\min_{\overline{\boldsymbol{\theta}} \in \mathbb{R}^{p n_{\mathrm{H}}}} R(f). \tag{6}$$

Problem (6) is a large but finite dimensional optimization problem that is nonconvex in the decision variable $\overline{\boldsymbol{\theta}}$. The standard approach is to employ first or second order search algorithms such as the variants of SGD or ADAM (Kingma & Ba, 2014).

2.2 Mean field limit

The mean field limit concerns with a continuum of hidden layer neuronal population by letting $n_{\mathrm{H}} \to \infty$. Then, we view (2) as the empirical average associated with the ensemble average

$$f_{\mathrm{MeanField}} := \int_{\mathbb{R}^p} \Phi(\boldsymbol{x}, \boldsymbol{\theta}) \underbrace{\mathrm{d}\mu(\boldsymbol{\theta})}_{\text{hidden neuronal population mass}} = \mathbb{E}_{\boldsymbol{\theta}}[\Phi(\boldsymbol{x}, \boldsymbol{\theta})], \tag{7}$$

where μ denotes the joint population measure supported on the hidden neuronal parameter space in $\mathbb{R}^p$. Assuming the absolute continuity of μ for all times, we write $\mathrm{d}\mu(\boldsymbol{\theta}) = \rho(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}$ where ρ denotes the joint population density function (PDF).

Thus, the risk functional R, now viewed as a function of the joint PDF ρ, takes the form

$$F(\rho) := R(f_{\text{MeanField}}(\boldsymbol{x}, \rho)) = \mathbb{E}_{(\boldsymbol{x},y)} \left(y - \int_{\mathbb{R}^p} \Phi(\boldsymbol{x}, \boldsymbol{\theta}) \rho(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} \right)^2$$
$$= F_0 + \int_{\mathbb{R}^p} V(\boldsymbol{\theta}) \rho(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} + \int_{\mathbb{R}^{2p}} U(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) \rho(\boldsymbol{\theta}) \rho(\tilde{\boldsymbol{\theta}}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\tilde{\boldsymbol{\theta}}, \tag{8}$$

where

$$F_0 := \mathbb{E}_{(\boldsymbol{x},y)} \left[y^2 \right], \quad V(\boldsymbol{\theta}) := \mathbb{E}_{(\boldsymbol{x},y)} [-2y\Phi(\boldsymbol{x}, \boldsymbol{\theta})], \quad U(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) := \mathbb{E}_{(\boldsymbol{x},y)} [\Phi(\boldsymbol{x}, \boldsymbol{\theta}) \Phi(\boldsymbol{x}, \tilde{\boldsymbol{\theta}})]. \tag{9}$$

Therefore, the supervised learning problem, in this mean field limit, becomes an infinite dimensional variational problem:

$$\min_{\rho} F(\rho) \tag{10}$$

where F is a sum of three functionals. The first summand F_0 is independent of ρ. The second summand is a potential energy given by expected value of "drift potential" V and is linear in ρ. The last summand is a bilinear interaction energy involving an "interaction potential" U and is nonlinear in ρ.

The main result in Mei et al. (2018) was that using first order SGD learning dynamics induces a gradient flow of the functional F w.r.t. the 2-Wasserstein metric W_2, i.e., the mean field learning dynamics results in a joint PDF trajectory $\rho(t, \boldsymbol{\theta})$. Then, the minimizer in (10) can be obtained from the large t limit of the following nonlinear PDE:

$$\frac{\partial \rho}{\partial t} = -\nabla^{W_2} F(\rho), \tag{11}$$

where the 2-Wasserstein gradient (Villani, 2021, Ch. 9.1) (Ambrosio et al., 2005, Ch. 8) of F is

$$\nabla^{W_2} F(\rho) := -\nabla \cdot \left(\rho \nabla \frac{\delta F}{\partial \rho} \right),$$

and $\frac{\delta}{\delta \rho}$ denotes the functional derivative w.r.t. ρ.

In particular, Mei et al. (2018) considered the regularized risk functional[1]

$$F_\beta(\rho) := F(\rho) + \beta^{-1} \int_{\mathbb{R}^p} \rho \log \rho \, \mathrm{d}\boldsymbol{\theta}, \quad \beta > 0, \tag{12}$$

by adding a strictly convex regularizer (scaled negative entropy) to the unregularized risk F. In that case, the sample path dynamics corresponding to the macroscopic dynamics (11) precisely becomes the noisy SGD:

$$\mathrm{d}\boldsymbol{\theta} = -\nabla_{\boldsymbol{\theta}} \left(V(\boldsymbol{\theta}) + \int_{\mathbb{R}^p} U(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) \rho(\tilde{\boldsymbol{\theta}}) \mathrm{d}\tilde{\boldsymbol{\theta}} \right) \mathrm{d}t + \sqrt{2\beta^{-1}} \, \mathrm{d}\boldsymbol{\eta}, \quad \boldsymbol{\theta}(t=0) \sim \rho_0, \tag{13}$$

where $\boldsymbol{\eta}$ is the standard Wiener process in $\mathbb{R}^p$, and the random initial condition $\boldsymbol{\theta}(t=0)$ follows the law of a suitable PDF ρ_0 supported over $\mathbb{R}^p$.

In this regularized case, (11) results in the following nonlinear PDE initial value problem (IVP):

$$\frac{\partial \rho}{\partial t} = \nabla_{\boldsymbol{\theta}} \cdot \left(\rho \left(V(\boldsymbol{\theta}) + \int_{\mathbb{R}^p} U(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) \rho(\tilde{\boldsymbol{\theta}}) \mathrm{d}\tilde{\boldsymbol{\theta}} \right) \right) + \beta^{-1} \Delta_{\boldsymbol{\theta}} \rho, \quad \rho(t=0, \boldsymbol{\theta}) = \rho_0. \tag{14}$$

In other words, the noisy SGD induces evolution of a PDF-valued trajectory governed by the advection, nonlocal interaction, and diffusion–the latter originating from regularization. Notice that a large value of $\beta > 0$ implies a small entropic regularization in (12), hence a small additive process noise in (13), and consequently, a small diffusion term in the PDE (14).

The regularized risk functional F_β in (12) can be interpreted as a free energy wherein F contributes a sum of the advection potential energy and interaction energy. The term $\beta^{-1} \int_{\mathbb{R}^p} \rho \log \rho \, \mathrm{d}\boldsymbol{\theta}$ contributes an internal energy due to the noisy fluctuations induced by the additive Wiener process noise $\sqrt{2\beta^{-1}} \mathrm{d}\boldsymbol{\eta}$ in (13).

In Mei et al. (2018), asymptotic guarantees were obtained for the solution of (14) to converge to the minimizer of F_β. Our idea, outlined next, is to solve the minimization of F_β using measure-valued proximal recursions.

[1] The parameter $\beta > 0$ is referred to as the inverse temperature.

2.3 Proximal mean field learning

For numerically computing the solution of the PDE IVP (14), we propose proximal recursions over $\mathcal{P}_2(\mathbb{R}^p)$, defined as the manifold of joint PDFs supported over $\mathbb{R}^p$ having finite second moments. Symbolically,

$$\mathcal{P}_2(\mathbb{R}^p) := \left\{ \text{Lebesgue integrable}\, \rho \,\text{over}\, \mathbb{R}^p \mid \rho \geq 0, \int_{\mathbb{R}^p} \rho \,\mathrm{d}\boldsymbol{\theta} = 1, \int_{\mathbb{R}^p} \boldsymbol{\theta}^\top \boldsymbol{\theta} \rho \,\mathrm{d}\boldsymbol{\theta} < \infty \right\}.$$

Proximal updates generalize the concept of gradient steps, and are of significant interest in both finite and infinite dimensional optimization (Rockafellar, 1976a;b; Bauschke et al., 2011; Teboulle, 1992; Bertsekas et al., 2011; Parikh et al., 2014). For a given input, these updates take the form of a structured optimization problem:

$$\begin{aligned} &\text{proximal update} \\ &= \underset{\text{decision variable}}{\arg\inf} \left\{ \frac{1}{2} \mathrm{dist}^2 \left(\text{decision variable}, \text{input}\right) + \text{time step} \times \text{functional}\left(\text{decision variable}\right) \right\}, \end{aligned} \tag{15}$$

for some suitable notion of distance $\mathrm{dist}(\cdot,\cdot)$ on the space of decision variables, and some associated functional. It is usual to view (15) as an *operator* mapping input $\mapsto$ proximal update, thus motivating the term *proximal operator*.

The connection between (15) and the gradient flow comes from recursively evaluating (15) with some initial choice for the input. For suitably designed pair ($\mathrm{dist}(\cdot,\cdot)$, functional), in the small time step limit, the sequence of proximal updates generated by (15) converge to the infimum of the functional. In other words, the gradient descent of the functional w.r.t. dist may be computed as the fixed point of the proximal operator (15). For a parallel between gradient descent and proximal recursions in finite and infinite dimensional gradient descent, see e.g., (Halder & Georgiou, 2017, Sec. I). Infinite dimensional proximal recursions over the manifold of PDFs have recently appeared in uncertainty propagation (Caluya & Halder, 2019b; Halder et al., 2022), stochastic filtering (Halder & Georgiou, 2017; 2018; 2019), and stochastic optimal control (Caluya & Halder, 2021b;a).

In our context, the decision variable $\rho \in \mathcal{P}_2(\mathbb{R}^p)$ and the distance metric $\mathrm{dist} \equiv \mathrm{W}_2$. Specifically, we propose recursions over discrete time $t_{k-1} := (k-1)h$ where the index $k \in \mathbb{N}$, and $h > 0$ is a constant time step-size. Leveraging that (14) describes gradient flow of the functional F_β w.r.t. the W_2 distance metric, the associated proximal recursion is of the form

$$\varrho_k = \mathrm{prox}_{hF_\beta}^{W_2}(\varrho_{k-1}) := \underset{\varrho \in \mathcal{P}_2(\mathbb{R}^p)}{\arg\inf} \left\{ \frac{1}{2} \left(W_2(\varrho, \varrho_{k-1})\right)^2 + h\, F_\beta(\varrho) \right\} \tag{16}$$

where $\varrho_{k-1}(\cdot) := \varrho(\cdot, t_{k-1})$, and $\varrho_0 \equiv \rho_0$. The notation $\mathrm{prox}_{hF_\beta}^{W_2}(\varrho_{k-1})$ can be parsed as "the proximal operator of the scaled functional hF_β w.r.t. the distance W_2, acting on the input $\varrho_{k-1} \in \mathcal{P}_2(\mathbb{R}^p)$". Our idea is to evaluate the recursion in the small h limit, i.e., for $h \downarrow 0$.

To account for the nonconvex bilinear term appearing in (12), following (Benamou et al., 2016, Sec. 4), we employ the approximation:

$$\int_{\mathbb{R}^{2p}} U(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) \varrho(\boldsymbol{\theta}) \varrho(\tilde{\boldsymbol{\theta}}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\tilde{\boldsymbol{\theta}} \approx \int_{\mathbb{R}^{2p}} U(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) \varrho(\boldsymbol{\theta}) \varrho_{k-1}(\tilde{\boldsymbol{\theta}}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\tilde{\boldsymbol{\theta}} \quad \forall k \in \mathbb{N}.$$

We refer to the resulting approximation of F_β as $\hat{F}_\beta$, i.e.,

$$\hat{F}_\beta(\varrho, \varrho_{k-1}) := \int_{\mathbb{R}^p} \left(F_0 + V(\boldsymbol{\theta}) + \left(\int_{\mathbb{R}^p} U(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) \varrho_{k-1}(\tilde{\boldsymbol{\theta}}) \mathrm{d}\tilde{\boldsymbol{\theta}} \right) + \beta^{-1} \log \varrho(\boldsymbol{\theta}) \right) \varrho(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}.$$

Notice in particular that $\hat{F}_\beta$ depends on both ϱ, ϱ_{k-1}, $k \in \mathbb{N}$. Consequently, this approximation results in a *semi-implicit* variant of (16), given by

$$\varrho_k := \underset{\varrho \in \mathcal{P}_2(\mathbb{R}^p)}{\arg\inf} \left\{ \frac{1}{2} \left(W_2(\varrho, \varrho_{k-1})\right)^2 + h \hat{F}_\beta(\varrho, \varrho_{k-1}) \right\}. \tag{17}$$

We have the following consistency guarantee (proof in Appendix A) among the solution of the PDE IVP (14) and that of the variational recursions (17).

Theorem 1. *Consider the regularized risk functional (12) wherein F is given by (8)-(9). Let $\rho(t,\boldsymbol{\theta})$ solve the IVP (14), and let $\{\varrho_{k-1}\}_{k\in\mathbb{N}}$ be the sequence generated by (17) with $\varrho_0 \equiv \rho_0$. Define the interpolation $\varrho_h : [0,\infty)\times\mathbb{R}^p \mapsto [0,\infty)$ as*

$$\varrho_h(t,\boldsymbol{\theta}) := \varrho_{k-1}(h,\boldsymbol{\theta})\ \forall t \in [(k-1)h, kh),\ k\in\mathbb{N}.$$

Then $\varrho_h(t,\boldsymbol{\theta}) \xrightarrow{h\downarrow 0} \rho(t,\boldsymbol{\theta})$ in $L^1(\mathbb{R}^p)$.

We next detail the proposed algorithmic approach to numerically solve (17).

3 ProxLearn: proposed proximal algorithm

The overall workflow of our proposed proximal mean field learning framework is shown in Fig. 1. We generate N samples from the known initial joint PDF ϱ_0 and store them as a weighted point cloud $\left\{\boldsymbol{\theta}_0^i, \varrho_0^i\right\}_{i=1}^N$. Here, $\varrho_0^i := \varrho_0\left(\boldsymbol{\theta}_0^i\right)$ for all $i\in[N]$. In other words, the weights of the samples are the joint PDF values evaluated at those samples.

For each $k\in\mathbb{N}$, the weighted point clouds $\left\{\boldsymbol{\theta}_k^i, \varrho_k^i\right\}_{i=1}^N$ are updated through the two-step process outlined in our proposed Algorithm 1, referred to as ProxLearn. At a high level, lines 9–18 in Algorithm 1 perform nonlinear block co-ordinate recursion on internally defined vectors $\boldsymbol{z},\boldsymbol{q}$ whose converged values yield the proximal update (line 19). We next explain where these recursions come from detailing both the derivation of ProxLearn and its convergence guarantee.

3.1 Derivation of ProxLearn

To derive the recursion given in ProxLearn, we first write the discrete version of (17) as

$$\boldsymbol{\varrho}_k = \arg\min_{\boldsymbol{\varrho}}\left\{\min_{\boldsymbol{M}\in\Pi(\boldsymbol{\varrho}_{k-1},\boldsymbol{\varrho})} \frac{1}{2}\left\langle \boldsymbol{C}_k, \boldsymbol{M}\right\rangle + h\left\langle \boldsymbol{v}_{k-1} + \boldsymbol{U}_{k-1}\boldsymbol{\varrho}_{k-1} + \beta^{-1}\log\boldsymbol{\varrho}, \boldsymbol{\varrho}\right\rangle\right\},\quad k\in\mathbb{N}, \tag{18}$$

where

$$\Pi\left(\boldsymbol{\varrho}_{k-1},\boldsymbol{\varrho}\right) := \{\boldsymbol{M}\in\mathbb{R}^{N\times N} \mid \boldsymbol{M}\geq \boldsymbol{0}\ (\text{elementwise}), \boldsymbol{M}\boldsymbol{1} = \boldsymbol{\varrho}_{k-1},\ \boldsymbol{M}^\top\boldsymbol{1} = \boldsymbol{\varrho}\}, \tag{19}$$

$$\boldsymbol{v}_{k-1} \equiv V\left(\boldsymbol{\theta}_{k-1}\right), \tag{20}$$

$$\boldsymbol{U}_{k-1} \equiv U\left(\boldsymbol{\theta}_{k-1}, \tilde{\boldsymbol{\theta}}_{k-1}\right), \tag{21}$$

and $\boldsymbol{C}_k\in\mathbb{R}^{N\times N}$ denotes the squared Euclidean distance matrix, i.e.,

$$\boldsymbol{C}_k(i,j) := \|\boldsymbol{\theta}_k^i - \boldsymbol{\theta}_{k-1}^j\|_2^2 \quad \forall (i,j)\in[N]\times[N].$$

We next follow a "regularize-then-dualize" approach. In particular, we regularize (18) by adding the entropic regularization $H(\boldsymbol{M}) := \langle \boldsymbol{M}, \log\boldsymbol{M}\rangle$, and write

$$\boldsymbol{\varrho}_k = \arg\min_{\boldsymbol{\varrho}}\left\{\min_{\boldsymbol{M}\in\Pi(\boldsymbol{\varrho}_{k-1},\boldsymbol{\varrho})} \frac{1}{2}\left\langle \boldsymbol{C}_k, \boldsymbol{M}\right\rangle + \epsilon H(\boldsymbol{M}) + h\left\langle \boldsymbol{v}_{k-1} + \boldsymbol{U}_{k-1}\boldsymbol{\varrho}_{k-1} + \beta^{-1}\log\boldsymbol{\varrho}, \boldsymbol{\varrho}\right\rangle\right\},\quad k\in\mathbb{N} \tag{22}$$

where $\epsilon > 0$ is a regularization parameter.

Following Karlsson & Ringh (2017, Lemma 3.5), Caluya & Halder (2019a, Sec. III), the Lagrange dual problem associated with (22) is

$$\left(\boldsymbol{\lambda}_0^{\text{opt}}, \boldsymbol{\lambda}_1^{\text{opt}}\right) = \arg\max_{\boldsymbol{\lambda}_0,\boldsymbol{\lambda}_1\in\mathbb{R}^N}\left\{\left\langle \boldsymbol{\lambda}_0, \boldsymbol{\varrho}_{k-1}\right\rangle - \hat{F}_\beta^\star\left(-\boldsymbol{\lambda}_1\right) - \frac{\epsilon}{h}\left(\exp\left(\boldsymbol{\lambda}_0^\top h/\epsilon\right)\exp\left(-\boldsymbol{C}_k/2\epsilon\right)\exp\left(\boldsymbol{\lambda}_1 h/\epsilon\right)\right)\right\} \tag{23}$$

where

$$\hat{F}^{\star}_{\beta}(\cdot) := \sup_{\vartheta}\{\langle \cdot, \vartheta\rangle - \hat{F}_{\beta}(\vartheta)\} \tag{24}$$

is the Legendre-Fenchel conjugate of the free energy $\hat{F}_{\beta}$ in (17), and the optimal coupling matrix $\boldsymbol{M}^{\text{opt}} := [m^{\text{opt}}(i,j)]_{i,j=1}^{N}$ in (22) has the Sinkhorn form

$$m^{\text{opt}}(i,j) = \exp\left(\boldsymbol{\lambda}_0(i)h/\epsilon\right)\exp\left(-\boldsymbol{C}_k(i,j)/(2\epsilon)\right)\exp\left(\boldsymbol{\lambda}_1(j)h/\epsilon\right). \tag{25}$$

To solve (23), considering (12), we write the "discrete free energy" as

$$\hat{F}_{\beta}(\boldsymbol{\varrho}) = \left\langle \boldsymbol{v}_{k-1} + \boldsymbol{U}_{k-1}\boldsymbol{\varrho}_{k-1} + \beta^{-1}\log\boldsymbol{\varrho}, \boldsymbol{\varrho}\right\rangle. \tag{26}$$

Its Legendre-Fenchel conjugate, by (24), is

$$\hat{F}^{\star}_{\beta}(\lambda) := \sup_{\vartheta}\{\boldsymbol{\lambda}^{\top}\boldsymbol{\varrho} - \boldsymbol{v}_{k-1}^{\top}\boldsymbol{\varrho} - \boldsymbol{\varrho}^{\top}\boldsymbol{U}_{k-1}\boldsymbol{\varrho}_{k-1} - \beta^{-1}\boldsymbol{\varrho}^{\top}\log\boldsymbol{\varrho}\}. \tag{27}$$

Setting the gradient of the objective in (27) w.r.t. $\boldsymbol{\varrho}$ to zero, and solving for $\boldsymbol{\varrho}$ gives the maximizer

$$\boldsymbol{\varrho}_{\max} = \exp\left(\beta\left(\boldsymbol{\lambda} - \boldsymbol{v}_{k-1} - \beta^{-1}\mathbf{1} - \boldsymbol{U}_{k-1}\boldsymbol{\varrho}_{k-1}\right)\right). \tag{28}$$

Substituting (28) back into (27), we obtain

$$\hat{F}^{\star}_{\beta}(\lambda) = \beta^{-1}\mathbf{1}\exp\left(\beta\left(\boldsymbol{\lambda} - \boldsymbol{v}_{k-1} - \boldsymbol{U}_{k-1}\boldsymbol{\varrho}_{k-1}\right) - \mathbf{1}\right). \tag{29}$$

Fixing $\boldsymbol{\lambda}_0$, and taking the gradient of the objective in (23) w.r.t. $\boldsymbol{\lambda}_1$ gives

$$\exp\left(\boldsymbol{\lambda}_1 h/\epsilon\right)\odot\left(\exp\left(-\boldsymbol{C}_k/2\epsilon\right)^{\top}\exp\left(\boldsymbol{\lambda}_0 h/\epsilon\right)\right) = \exp(-\beta\boldsymbol{v}_{k-1} - \beta\boldsymbol{U}_{k-1}\boldsymbol{\varrho}_{k-1} - \mathbf{1})\odot\left(\exp\left(\boldsymbol{\lambda}_1 h/\epsilon\right)\right)^{-\frac{\beta\epsilon}{h}}. \tag{30}$$

Likewise, fixing $\boldsymbol{\lambda}_1$, and taking the gradient of the objective in (23) w.r.t. $\boldsymbol{\lambda}_0$, gives

$$\exp\left(\boldsymbol{\lambda}_0 h/\epsilon\right)\odot\left(\exp\left(-\boldsymbol{C}_k/2\epsilon\right)\exp\left(\boldsymbol{\lambda}_1 h/\epsilon\right)\right) = \boldsymbol{\varrho}_{k-1}. \tag{31}$$

Next, letting $\boldsymbol{\Gamma}_k := \exp\left(-\boldsymbol{C}_k/2\epsilon\right)$, $\boldsymbol{q} := \exp\left(\boldsymbol{\lambda}_0 h/\epsilon\right)$, $\boldsymbol{z} := \exp\left(\boldsymbol{\lambda}_1 h/\epsilon\right)$, and $\boldsymbol{\xi}_{k-1} := \exp(-\beta\boldsymbol{v}_{k-1} - \beta\boldsymbol{U}_{k-1}\boldsymbol{\varrho}_{k-1} - \mathbf{1})$, we express (30) as

$$\boldsymbol{z}\odot\left(\boldsymbol{\Gamma}_k^{\top}\boldsymbol{q}\right) = \boldsymbol{\xi}_{k-1}\odot\boldsymbol{z}^{-\frac{\beta\epsilon}{h}}, \tag{32}$$

and (31) as

$$\boldsymbol{q}\odot\left(\boldsymbol{\Gamma}_k\boldsymbol{z}\right) = \boldsymbol{\varrho}_{k-1}. \tag{33}$$

Finally using (19), we obtain

$$\boldsymbol{\varrho}_k = \left(\boldsymbol{M}^{\text{opt}}\right)^{\top}\mathbf{1} = \sum_{j=1}^{N} m^{\text{opt}}(j,i) = \boldsymbol{z}(i)\sum_{j=1}^{N}\boldsymbol{\Gamma}_k(j,i)\boldsymbol{q}(j) = \boldsymbol{z}\odot\boldsymbol{\Gamma}_k^{\top}\boldsymbol{q}. \tag{34}$$

In summary, (34) allows us to numerically perform the proximal update.

Remark 1. *Note that in Algorithm 1 (i.e., ProxLearn) presented in Sec. 3, the lines 11, 12, and 19 correspond to (32), (33) and (34), respectively.*

3.2 Convergence of ProxLearn

Our next result provides the convergence guarantee for our proposed ProxLearn algorithm derived in Sec. 3.1.

Proposition 1. *The recursions given in lines 7–18 in Algorithm 1 ProxLearn, converge to a unique fixed point* $(\boldsymbol{q}^{\text{opt}}, \boldsymbol{z}^{\text{opt}}) \in \mathbb{R}^N_{>0} \times \mathbb{R}^N_{>0}$. *Consequently, the proximal update (34) (i.e., the evaluation at line 19 in Algorithm 1) is unique.*

Proof. Notice that the mappings $(\boldsymbol{q}(:,\ell), \boldsymbol{z}(:,\ell)) \mapsto (\boldsymbol{q}(:,\ell+1), \boldsymbol{z}(:,\ell+1))$ given in lines 11 and 12 in Algorithm 1, are cone preserving since these mappings preserve the product orthant $\mathbb{R}^N_{>0} \times \mathbb{R}^N_{>0}$. This is a direct consequence of the definition of $\boldsymbol{q}, \boldsymbol{z}$ in terms of $\boldsymbol{\lambda}_0, \boldsymbol{\lambda_1}$.

Now the idea is to show that the recursions in lines 11 and 12 in Algorithm 1, as composite nonlinear maps, are in fact contractive w.r.t. a suitable metric on this cone. Following Caluya & Halder (2019a, Theorem 3), the $\boldsymbol{z}$ iteration given in line 11 in Algorithm 1, ProxLearn, for $\ell = 1, 2, \ldots$, is strictly contractive in the Thompson's part metric (Thompson, 1963) and thanks to the Banach contraction mapping theorem, converges to a unique fixed point $\boldsymbol{z}^{\text{opt}} \in \mathbb{R}^N_{>0}$.

We note that our definition of $\boldsymbol{\xi}_{k-1}$ is slightly different compared to the same in Caluya & Halder (2019a, Theorem 3), but this does not affect the proof. From definition of $\boldsymbol{C}_k$, we have $\boldsymbol{C}_k \in [0, \infty)$ which implies $\boldsymbol{\Gamma}_k(i,j) \in (0,1]$. Therefore, $\boldsymbol{\Gamma}_k$ is a positive linear map for each $k \in \mathbb{N}$. Thus, by (linear) Perron-Frobenius theorem, the linear maps $\boldsymbol{\Gamma}_k$ are contractive. Consequently the $\boldsymbol{q}$ iterates also converge to unique fixed point $\boldsymbol{q}^{\text{opt}} \in \mathbb{R}^N_{>0}$.

Since converged pair $(\boldsymbol{q}^{\text{opt}}, \boldsymbol{z}^{\text{opt}}) \in \mathbb{R}^N_{>0} \times \mathbb{R}^N_{>0}$ is unique, so is the proximal update (34), i.e., the evaluation at line 19 in Algorithm 1. □

We next discuss the implementation details for the proposed ProxLearn algorithm.

3.3 Implementation of ProxLearn

We start by emphasizing that ProxLearn updates both the parameter sample locations $\boldsymbol{\theta}^i_k$ and the joint PDF values ϱ^i_k evaluated at those locations, without gridding the parameter space. In particular, the PDF values are updated online, not as an offline *a posteriori* function approximation as in traditional Monte Carlo algorithms.

We will apply ProxLearn, as outlined here, in Sec. 4. In Sec. 5, we will detail additional modifications of ProxLearn to showcase its flexibility.

Required inputs of ProxLearn are the inverse temperature β, the time step-size h, a regularization parameter ε, and the number of samples N. Additional required inputs are the training feature data $\boldsymbol{X} := [\boldsymbol{x}_1 \ldots \boldsymbol{x}_{n_{\text{data}}}]^\top \in \mathbb{R}^{n_{\text{data}} \times n_x}$ and the corresponding training labels $\boldsymbol{y} := [y_1 \ldots y_{n_{\text{data}}}]^\top \in \mathbb{R}^{n_{\text{data}}}$, as well the weighted point cloud $\{\boldsymbol{\theta}^i_{k-1}, \varrho^i_{k-1}\}_{i=1}^N$ for each $k \in \mathbb{N}$. Furthermore, ProxLearn requires two internal parameters as user input: the numerical tolerance δ, and the maximum number of iterations L.

For $k \in \mathbb{N}$, let

$$\boldsymbol{\Theta}_{k-1} := \begin{pmatrix} (\boldsymbol{\theta}^1_{k-1})^\top \\ (\boldsymbol{\theta}^2_{k-1})^\top \\ \vdots \\ (\boldsymbol{\theta}^N_{k-1})^\top \end{pmatrix} \in \mathbb{R}^{N \times p}, \quad \boldsymbol{\varrho}_{k-1} := \begin{pmatrix} \varrho^1_{k-1} \\ \varrho^2_{k-1} \\ \vdots \\ \varrho^N_{k-1} \end{pmatrix} \in \mathbb{R}^N_{>0}.$$

In line 2 of Algorithm 1, ProxLearn updates the locations of the parameter vector samples $\boldsymbol{\theta}^i_k$ in $\mathbb{R}^p$ via Algorithm 2, EulerMaruyama. This location update takes the form:

$$\boldsymbol{\theta}^i_k = \boldsymbol{\theta}^i_{k-1} - h\nabla\left(V\left(\boldsymbol{\theta}^i_{k-1}\right) + \omega\left(\boldsymbol{\theta}^i_{k-1}\right)\right) + \sqrt{2\beta^{-1}}\left(\boldsymbol{\eta}^i_k - \boldsymbol{\eta}^i_{k-1}\right), \tag{35}$$

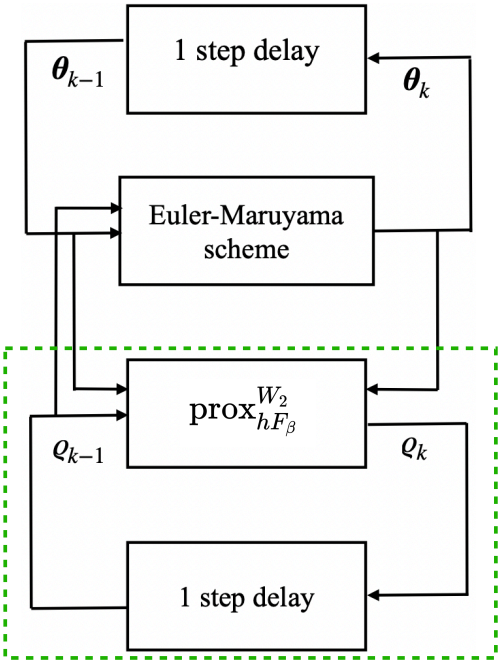


Figure 1: Schematic of the proposed proximal algorithm for mean field learning, updating scattered point cloud $\left\{\boldsymbol{\theta}_{k-1}^i, \varrho_{k-1}^i\right\}_{i=1}^N$ for $k \in \mathbb{N}$. The location of the points $\left\{\boldsymbol{\theta}_{k-1}^i\right\}_{i=1}^N$ are updated via the Euler-Maruyama scheme; the corresponding probability weights are computed via proximal updates highlighted within the dashed box. Explicitly performing the proximal updates via the proposed algorithm, and thereby enabling mean filed learning as an interacting weighted particle system, is our novel contribution

.

where $\omega(\cdot) := \int U(\cdot, \tilde{\boldsymbol{\theta}})\varrho(\tilde{\boldsymbol{\theta}})\mathrm{d}\tilde{\boldsymbol{\theta}}$, and $\boldsymbol{\eta}_{k-1}^i := \boldsymbol{\eta}^i(t = (k-1)h)$, $\forall k \in \mathbb{N}$.

To perform this update, EulerMaruyama constructs a matrix $\boldsymbol{P}_{k-1}$ whose (i,j)th element is $\boldsymbol{P}_{k-1}(i,j) = \Phi(\boldsymbol{x}_j, \boldsymbol{\theta}_{k-1}^i)$. From $\boldsymbol{P}_{k-1}$, we construct $\boldsymbol{v}_{k-1}$ and $\boldsymbol{U}_{k-1}$ as in lines 3 and 5. In line 6 of Algorithm 2, EulerMaruyama uses the automatic differentiation module of PyTorch Library, Backward (Paszke et al., 2017), to calculate the gradients needed to update $\boldsymbol{\Theta}_{k-1}$ to $\boldsymbol{\Theta}_k$ $\forall k \in \mathbb{N}$.

Once $\boldsymbol{\Theta}_k$, $\boldsymbol{v}_{k-1}$, and $\boldsymbol{U}_{k-1}$ have been constructed via EulerMaruyama, ProxLearn maps the $N \times 1$ vector $\boldsymbol{\varrho}_{k-1}$ to the proximal update $\boldsymbol{\varrho}_k$.

We next illustrate the implementation of ProxLearn for binary and multi-class classification case studies. A GitHub repository containing our code for the implementation of these applications can be found at `https://github.com/zalexis12/Proximal-Mean-Field-Learning.git`. Please refer to the Readme file therein for an outline of the structure of our code.

4 Case study: binary classification

In this Section, we report numerical results for our first case study, where we apply the proposed ProxLearn algorithm for binary classification.

For this case study, we perform two implementations on different computing platforms. Our first implementation is on a PC with 3.4 GHz 6-Core Intel Core i5 processor, and 8 GB RAM. For runtime improvement, we then use a Jetson TX2 with a NVIDIA Pascal GPU with 256 CUDA cores, 64-bit NVIDIA Denver and ARM Cortex-A57 CPUs.

Algorithm 1 Proximal Algorithm

procedure ProxLearn($\boldsymbol{\varrho}_{k-1}, \boldsymbol{\Theta}_{k-1}, \beta, h, \varepsilon, N, \boldsymbol{X}, \boldsymbol{y}, \delta, L$)
  $\boldsymbol{v}_{k-1},\ \boldsymbol{U}_{k-1},\ \boldsymbol{\Theta}_k \leftarrow$ EulerMaruyama($h, \beta, \boldsymbol{\Theta}_{k-1}, \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\varrho}_{k-1}$)
  $\boldsymbol{C}_k(i,j) \leftarrow \left\| \boldsymbol{\theta}_k{}^i - \boldsymbol{\theta}_{k-1}^j \right\|_2^2$
  $\boldsymbol{\Gamma}_k \leftarrow \exp(-\boldsymbol{C}_k/2\varepsilon)$
  $\boldsymbol{\xi}_{k-1} \leftarrow \exp(-\beta \boldsymbol{v}_{k-1} - \beta \boldsymbol{U}_{k-1}\boldsymbol{\varrho}_{k-1} - \mathbf{1})$
  $\boldsymbol{z}_0 \leftarrow \text{rand}_{N\times 1}$
  $\boldsymbol{z} \leftarrow [\boldsymbol{z}_0, \mathbf{0}_{N\times(L-1)}]$
  $\boldsymbol{q} \leftarrow [\boldsymbol{\varrho}_{k-1} \oslash (\boldsymbol{\Gamma_k z_0}), \mathbf{0}_{N\times(L-1)}]$
  $\ell = 1$
  while $\ell \leq L$ **do**
    $\boldsymbol{z}(:,\ell+1) \leftarrow (\boldsymbol{\xi}_{k-1} \oslash (\boldsymbol{\Gamma}_k^\top \boldsymbol{q}(:,\ell))^{\frac{1}{1+\beta\varepsilon/h}}$
    $\boldsymbol{q}(:,\ell+1) \leftarrow \boldsymbol{\varrho}_{k-1} \oslash (\boldsymbol{\Gamma}_k \boldsymbol{z}(:,\ell+1))$
    if $||\boldsymbol{q}(:,\ell+1) - \boldsymbol{q}(:,\ell)|| < \delta$ and $||\boldsymbol{z}(:,\ell+1) - \boldsymbol{z}(:,\ell)|| < \delta$ **then**
      Break
    else
      $\ell \leftarrow \ell + 1$
    end if
  end while
  return $\boldsymbol{\varrho}_k \leftarrow \boldsymbol{z}(:,\ell) \odot (\boldsymbol{\Gamma}_k^\top \boldsymbol{q}(:,\ell))$
end procedure

Algorithm 2 Euler-Maruyama Algorithm

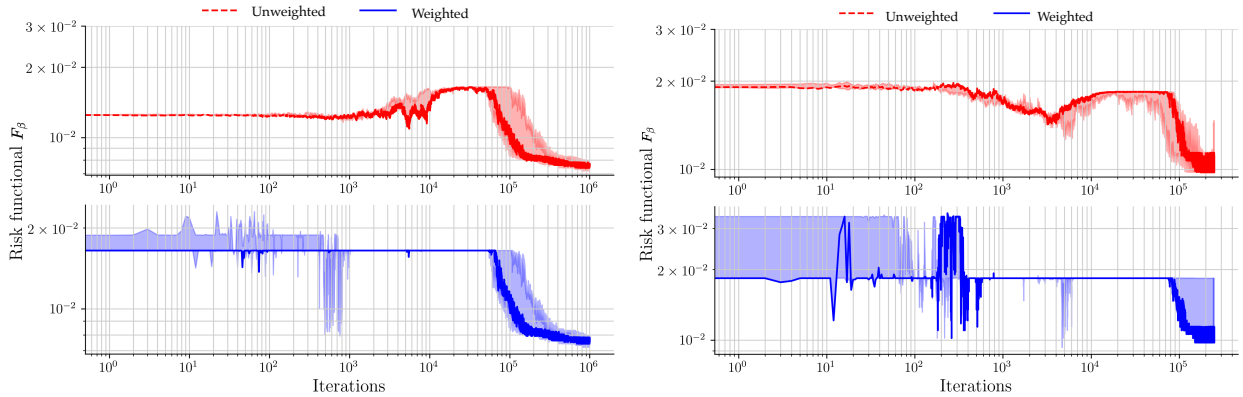
procedure EulerMaruyama($h, \beta, \boldsymbol{\Theta}_{k-1}, \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\varrho}_{k-1}$)
  $\boldsymbol{P}_{k-1} \leftarrow \boldsymbol{\Phi}(\boldsymbol{\Theta}_{k-1}, \boldsymbol{X})$
  $\boldsymbol{U}_{k-1} \leftarrow 1/n_{\text{data}} \boldsymbol{P}_{k-1}\boldsymbol{P}_{k-1}^\top$
  $\boldsymbol{u}_{k-1} \leftarrow \boldsymbol{U}_{k-1}\boldsymbol{\varrho}_{k-1}$
  $\boldsymbol{v}_{k-1} \leftarrow -2/n_{\text{data}} \boldsymbol{P}_{k-1}\boldsymbol{y}$
  $\boldsymbol{D} \leftarrow$ Backward $(\boldsymbol{u}_{k-1} + \boldsymbol{v}_{k-1})$
  $\boldsymbol{G} \leftarrow \sqrt{2h/\beta} \times \text{randn}_{N\times p}$
  $\boldsymbol{\Theta}_k \leftarrow \boldsymbol{\Theta}_{k-1} + h \times \boldsymbol{D} + \boldsymbol{G}$
end procedure

4.1 WDBC data set

We apply the proposed algorithm to perform a binary classification on the Wisconsin Diagnostic Breast Cancer (henceforth, WDBC) data set available at the UC Irvine machine learning repository (Dua & Graff, 2017). This data set consists of the data of scans from 569 patients. There are $n_x = 30$ features from each scan. Scans are classified as "benign" (which we label as -1) or "malignant" (labeled as $+1$).

In (2), we define $\Phi\left(\boldsymbol{x}, \boldsymbol{\theta}_{k-1}^i\right) := a_{k-1}^i \tanh(\langle \boldsymbol{w}_{k-1}^i, \boldsymbol{x}\rangle + b_{k-1}^i)\ \forall i \in [N]$ after $(k-1)$ updates. The parameters $a_{k-1}^i, \boldsymbol{w}_{k-1}^i$ and b_{k-1}^i are the scaling, weight and bias of the ith sample after $(k-1)$ updates, respectively. Letting $p := n_x + 2$, the parameter vector of the ith sample after $(k-1)$ updates is

$$\boldsymbol{\theta}_{k-1}^i := \begin{pmatrix} a_{k-1}^i \\ b_{k-1}^i \\ \boldsymbol{w}_{k-1}^i \end{pmatrix} \in \mathbb{R}^p, \quad \forall\, i \in [N].$$



(a) Regularized risk calculated via CPU.

(b) Regularized risk calculated via GPU.

Figure 2: The solid line shows the regularized risk functional F_β versus the number of proximal recursions shown for the WDBC dataset with $\beta = 0.05$. The shadow shows the F_β variation range for different values of $\beta \in \{0.03, 0.05, 0.07\}$.

We set $\varrho_0 \equiv \text{Unif}\left([0.9, 1.1] \times [-0.1, 0.1] \times [-1, 1]^{n_x}\right)$, a uniform joint PDF supported over $n_p = n_x + 2 = 32$ dimensional mean field parameter space.

We use 70% of the entire data set as training data. As discussed in Sec. 3, we learn the mean field parameter distribution via weighted scattered point cloud evolution using ProxLearn. We then use the confusion matrix method (Visa et al., 2011) to evaluate the accuracy of the obtained model over the test data, which is the remaining 30% of the full data set, containing n_{test} points.

For each test point $\boldsymbol{x}_{\text{test}} \in \mathbb{R}^{n_x}$, we construct

$$\boldsymbol{\varphi}(\boldsymbol{x}_{\text{test}}) := \begin{pmatrix} \Phi(\boldsymbol{x}_{\text{test}}, \boldsymbol{\theta}^1_{k-1}) \\ \Phi(\boldsymbol{x}_{\text{test}}, \boldsymbol{\theta}^2_{k-1}) \\ \vdots \\ \Phi(\boldsymbol{x}_{\text{test}}, \boldsymbol{\theta}^N_{k-1}) \end{pmatrix} \in \mathbb{R}^N$$

where $\boldsymbol{\theta}^i_{k-1}$ is obtained from the training process. We estimate $f_{\text{MeanField}}$ in (7) in two ways. First, we estimate $f_{\text{MeanField}}$ as a sample average of the elements of $\boldsymbol{\varphi}$. Second, we estimate $f_{\text{MeanField}}$ by numerically approximating the integral in (7) using the propagated samples $\{\boldsymbol{\theta}^i_{k-1}, \varrho^i_{k-1}\}_{i=1}^N$ for $k \in \mathbb{N}$. We refer to these as the "unweighted estimate" and "weighted estimate," respectively. While the first estimate is an empirical average, the second uses the weights $\{\varrho^i_{k-1}\}_{i=1}^N$ obtained from the proposed proximal algorithm. The $f_{\text{MeanField}}$ "unweighted estimate" and "weighted estimate" are then passed through the softmax and argmax functions respectively, to produce the predicted labels.

4.2 Numerical experiments

We set the number of samples $N = 1000$, numerical tolerance $\delta = 10^{-3}$, the maximum number of iterations $L = 300$, and the regularizing parameter $\varepsilon = 1$. Additionally, we set the time step to $h = 10^{-3}$. We run the simulation for different values of the inverse temperature β, and list the corresponding classification accuracy in Table 1. The "weighted estimate," the ensemble average using proximal updates, produces more accurate results, whereas the "unweighted estimate," the empirical average, is found to be more sensitive to the inverse temperature β.

For each fixed β, we perform 10^6 proximal recursions incurring approx. 33 hours of computational time. Fig. 2a shows the risk functional, computed as the averaged loss over the test data using each of the two estimates described above.

Table 1: Classification accuracy of the proposed computational framework for the WDBC Dataset

β	Unweighted	Weighted
0.03	91.17%	92.35%
0.05	92.94%	92.94%
0.07	78.23%	92.94%

Table 2: Classification accuracy on Jetson Tx2, after 2.5×10^5 iterations

β	Unweighted	Weighted	Runtime (hr)
0.03	91.18%	91.18%	1.415
0.05	91.18%	92.94%	1.533
0.07	90.59%	91.76%	1.704

4.3 Runtime improvements

To improve the runtime of our algorithm, we run our code on a Jetson TX2 module, converting data and variables to PyTorch variables.

We begin calculations in Float32, switching to Float64 only when needed to avoid not-a-number (NaN) errors. This switch typically occurs after 2×10^5 to 3×10^5 iterations. As shown in Table 2, we train the neural network to a comparable accuracy in only 2.5×10^5 iterations. The new runtime is around 6% of the original runtime for the CPU-based computation. Fig. 2b shows the risk functional calculated via this updated code. We parallelize these calculations, taking advantage of the GPU capacity of the Jetson TX2.

4.4 Computational complexity

In this case study, we determine the computational complexity of ProxLearn as follows. Letting $\boldsymbol{a}_{k-1} := \begin{pmatrix} a_{k-1}^1 & \dots & a_{k-1}^N \end{pmatrix}^\top$, $\boldsymbol{b}_{k-1} := \begin{pmatrix} b_{k-1}^1 & \dots & b_{k-1}^N \end{pmatrix}^\top$, $\boldsymbol{W}_{k-1} := \begin{pmatrix} \boldsymbol{w}_{k-1}^1 & \dots & \boldsymbol{w}_{k-1}^N \end{pmatrix}^\top$, we create the $N \times n_{\text{data}}$ matrix

$$\boldsymbol{P}_{k-1} := \left(\boldsymbol{a}_{k-1}\mathbf{1}^\top\right) \odot \tanh(\boldsymbol{W}_{k-1}\boldsymbol{X}^\top + \boldsymbol{b}_{k-1}\mathbf{1}^\top), \tag{36}$$

which has complexity $O(n_{\text{data}} N n_x)$. The subsequent creation of matrix $\boldsymbol{U}_{k-1}$ in line 3 of Algorithm 2 has complexity $O(n_{\text{data}} N^2)$, and creating $\boldsymbol{v}_{k-1}$ and $\boldsymbol{u}_{k-1}$ takes complexity $O(N n_{\text{data}})$ and $O(N^2)$ respectively.

The complexity in calculating the relevant derivatives of $\boldsymbol{v}_{k-1}$ and $\boldsymbol{u}_{k-1}$ is $O(N^2 n_{\text{data}} n_x)$ (these derivatives are calculated in Appendices B and C). Updating $\boldsymbol{\Theta}_{k-1}$ using these results has complexity of $O(Np) = O(N n_x)$. Therefore, the process of updating $\boldsymbol{\Theta}_{k-1}$ via EulerMaruyama is $O(N^2 n_{\text{data}} n_x)$.

The significant complexity in the remainder of ProxLearn arises in the construction of matrix $\boldsymbol{C}_k$ in line 3 and the matrix-vector multiplications within the while loop in lines 11, 12.

The creation of the $N \times N$ matrix $\boldsymbol{C}_k$, in which each element is the vector norm of a $n_x \times 1$ vector, is $O(n_x N^2)$. In a worst-case scenario, the while loop runs L times. The operations of leading complexity within the while loop are the multiplications of the $\boldsymbol{\Gamma}_k$ matrix of size $N \times N$ with the $N \times 1$ vectors, which have a complexity of $O(N^2)$. Therefore, the while loop has a complexity of $O(LN^2)$.

Updating $\boldsymbol{\varrho}_{k-1}$ thus has a complexity of $O((n_x + L)N^2)$. In practice, the while loop typically ends far before reaching the maximum number of iterations L.

From this analysis, we find that the overall complexity of ProxLearn is $O(N^2(n_{\text{data}} n_x + L))$.

4.5 Comparisons to existing results

As a first study, our numerical results achieve reasonable accuracy compared to the state-of-art, even though our proposed meshless proximal algorithm is very different from the existing implementations. In this Section,

Table 3: Comparison of average classification accuracy from Bonet et al. (2022, Table 1) to our algorithm, ProxLearn

Dataset	JKO-ICNN	SWGF + RealNVP	ProxLearn, Weighted	ProxLearn, Unweighted
Banana	0.550 ± 10^{-2}	0.559 ± 10^{-2}	0.551 ± 10^{-2}	$0.535 \pm 5 \cdot 10^{-2}$
Diabetes	$0.777 \pm 7 \cdot 10^{-3}$	$0.778 \pm 2 \cdot 10^{-3}$	$0.736 \pm 2 \cdot 10^{-2}$	0.731 ± 10^{-2}
Twonorm	$0.981 \pm 2 \cdot 10^{-4}$	$0.981 \pm 6 \cdot 10^{-4}$	$0.972 \pm 2 \cdot 10^{-3}$	$0.972 \pm 2 \cdot 10^{-3}$

we compare the numerical performance with existing methods as in Mokrov et al. (2021) and Bonet et al. (2022). We apply our proximal algorithm for binary classification to three datasets also considered in Mokrov et al. (2021) and Bonet et al. (2022): the banana, diabetes, and twonorm datasets.

The banana dataset consists of 5300 data points, each with $n_x = 2$ features, which we rescale to lie between 0 and 8. We set $\beta = 0.05$, draw our initial weights $\boldsymbol{w}$ from $\text{Unif}\left([-2, 2]^{n_x}\right)$ and bias b from $\text{Unif}\left([-0.3, 0.3]\right)$, and set $\boldsymbol{\varrho}_0 \equiv \text{Unif}\left(0, 1000\right)$. We run our code for 3500 iterations, splitting the data evenly between test and training data.

The diabetes dataset consists of $n_x = 8$ features from each of 768 patients. Based on our experimental results, we make the following adjustments to our algorithm: we redefine $\beta = 0.65$, $\boldsymbol{\varrho}_0 \equiv \text{Unif}\left(0, 1000\right)$, and draw our initial weights $\boldsymbol{w}$ from $\text{Unif}\left([-2, 2]^{n_x}\right)$. We rescale the data to lie between 0 and 1, and use half of the dataset for training purposes, and the remainder as test data. In this case, we run our code for 4.99×10^5 iterations.

The twonorm dataset consists of 7,400 samples drawn from two different normal distributions, with $n_x = 20$ features. We again consider 50% of the same as training data and used the remaining 50% as test data, and rescale the given data by a factor of 8. Based on our empirical observations, we redefine $\beta = 1.95$, and once more draw our initial weights $\boldsymbol{w}$ from $\text{Unif}\left([-2, 2]^{n_x}\right)$ and set $\boldsymbol{\varrho}_0 \equiv \text{Unif}\left(0, 1000\right)$. In this case, we perform 10^4 proximal recursions in each separate run.

We run our code five times for each of the three datasets under consideration and compute "unweighted estimates" and "weighted estimates" in each case, as described above. These estimates assign each data point a value: negative values predict the label as 0, while positive values predict the label as 1. From these results, we calculate the weighted and unweighted accuracy by finding the percentage of predicted test labels that match the actual test labels. The average accuracy over all five runs is reported in Table 3, alongside the results reported in Bonet et al. (2022, Table 1). We achieve comparable accuracy to these recent results.

5 Case study: multi-class classification

We next apply the proposed proximal algorithm to a ten-class classification problem using the Semeion Handwritten Digit (hereafter SHD) Data Set (Dua & Graff, 2017). This numerical experiment is performed on the aforementioned Jetson TX2.

5.1 SHD data set

The SHD Data Set consists of 1593 handwritten digits. By viewing each digit as 16×16 pixel image, each image is represented by $n_x = 16^2 = 256$ features. Each feature is a boolean value indicating whether a particular pixel is filled. We subsequently re-scale these features such that $\boldsymbol{x}_i \in \{-1, 1\}^{n_x}$.

5.2 Adaptations to ProxLearn for multi-class case

To apply ProxLearn for a multi-class case, we make several adaptations. For instance, rather than attempting to determine $f(\boldsymbol{x}) \approx y$, we redefine $f(\boldsymbol{x})$ to represent a mapping of input data to the predicted likelihood of the correct label. We therefore redefine the variables, parameters, and risk function as follows.

Each label is represented by a 1×10 vector of booleans, stored in a $n_{\text{data}} \times 10$ matrix $\boldsymbol{Y}$ where $Y_{i,j} = 1$ if the ith data point $\boldsymbol{x}_i$ has label j, and $Y_{i,j} = 0$ otherwise.

We construct the $N \times n_{\text{data}}$ matrix $\boldsymbol{P}_{k-1}$ by defining the (j, i) element of $\boldsymbol{P}_{k-1}$ as

$$\begin{aligned} \boldsymbol{P}_{k-1}(j,i) &:= \boldsymbol{\Phi}(\boldsymbol{\theta}^j_{k-1}, \boldsymbol{X}(i,:), \boldsymbol{Y}(i,:)) \\ &:= \left\langle \textsc{softmax}(\boldsymbol{X}(i,:)(\boldsymbol{\theta}^j_{k-1})^\top), (\boldsymbol{Y}(i,:))^\top \right\rangle \end{aligned} \tag{37}$$

where $\langle \cdot, \cdot \rangle$ denotes the standard Euclidean inner product. The SOFTMAX function in (37) produces a 10×1 vector of non-negative entries that sum to 1. By taking the inner product of this vector with the Boolean vector $\boldsymbol{Y}(i,:)$, we define $\boldsymbol{P}_{k-1}(j,i) = \boldsymbol{\Phi}(\boldsymbol{\theta}^j_{k-1}, \boldsymbol{X}(i,:), \boldsymbol{Y}(i,:))$ as the perceived likelihood that the data point i is labeled correctly by sample j. As our model improves, this value approaches 1, which causes the probability of an incorrect label to drop.

As this newly defined $\boldsymbol{P}_{k-1}$ does not call for bias or scaling, the weights alone are stored in the $N \times p$ matrix $\boldsymbol{\Theta}_{k-1}$. In this case, $p := 10n_x$, as each of the n_x features requires a distinct weight for each of the ten labels. For convenience, we reshape $\boldsymbol{\Theta}_{k-1} = (\boldsymbol{\theta}^1_{k-1}, ..., \boldsymbol{\theta}^N_{k-1})$, where each $\boldsymbol{\theta}^i_{k-1}$ is a $10 \times n_x$ matrix. Therefore, $\boldsymbol{\Theta}_{k-1}$ is a $10 \times n_x \times N$ tensor.

We redefine the unregularized risk to reflect our new $\boldsymbol{\Phi}$ as follows:

$$F(\rho) := \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})} \left(1 - \int_{\mathbb{R}^p} \Phi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) \rho(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} \right)^2. \tag{38}$$

Expanding the above, we arrive at a form that resembles (8), now with the following adjusted definitions:

$$F_0 := 1, \tag{39}$$

$$V(\boldsymbol{\theta}) := \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})}[-2\Phi(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y})], \tag{40}$$

$$U(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) := \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})}[\Phi(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y}) \Phi(\tilde{\boldsymbol{\theta}}, \boldsymbol{x}, \boldsymbol{y})]. \tag{41}$$

We use the regularized risk functional F_β as in (12) where F now is given by (38). Due to the described changes in the structure of $\boldsymbol{\Theta}_{k-1}$, the creation of $\boldsymbol{C}_k$ in line 3 of PROXLEARN results in a $10 \times N \times N$ tensor, which we then sum along the ten element axis, returning an $N \times N$ matrix.

Finally, we add a scaling in line 7 of EULERMARUYAMA, scaling the noise by a factor of 1/100.

5.3 Numerical experiments

With the adaptations mentioned above, we set the inverse temperature $\beta = 0.5$, $\epsilon = 10$, the step size $h = 10^{-3}$, and $N = 100$. We draw the initial weights from $\text{Unif}\left([-1, 1]^{10n_x}\right)$. We take the first $n_{\text{data}} = 1000$ images as training data, reserving the remaining $n_{\text{test}} = 593$ images as test data, and execute 30 independent runs of our code, each for 10^6 proximal recursions.

To evaluate the training process, we create a matrix $\boldsymbol{P}^{\text{test}}_{k-1} \in \mathbb{R}^{N \times n_{\text{test}}}$, using test data rather than training data, but otherwise defined as in (37). We then calculate a weighted approximation of F_β:

$$F_\beta \approx \frac{1}{n_{\text{test}}} \left\| \mathbf{1} - (\boldsymbol{P}^{\text{test}}_{k-1})^\top \boldsymbol{\varrho} \right\|_2^2, \tag{42}$$

and an unweighted approximation:

$$F_\beta \approx \frac{1}{n_{\text{test}}} \left\| \mathbf{1} - \frac{1}{N} (\boldsymbol{P}^{\text{test}}_{k-1})^\top \mathbf{1} \right\|_2^2, \tag{43}$$

which we use to produce the risk and weighted risk log-log plots shown in Fig. 3.

Notably, despite the new activation function and the adaptations described above, our algorithm produces similar risk plots in the binary and multi-class cases. The run time for 10^6 iterations is approximately 5.3 hours.

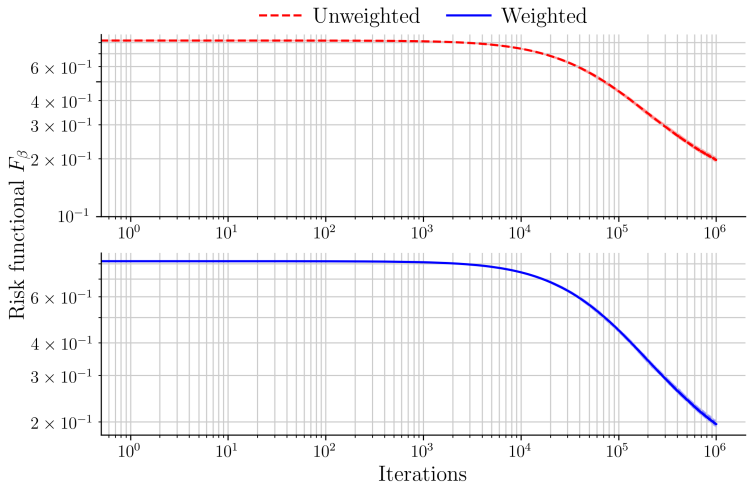


Figure 3: The solid line shows the average regularized risk functional F_β versus the number of proximal recursions shown for the Semeion dataset with $\beta = 0.5$. The narrow shadow shows the F_β variation range with the same β using the results of 30 independent runs, each starting from the same initial point cloud $\{\boldsymbol{\theta}_0^i, \varrho_0^i\}_{i=1}^N$.

To evaluate our multi-class model, we calculate the percentage of accurately labeled test data by first taking ARGMAX $(\boldsymbol{X}_{\text{test}}\boldsymbol{\Theta}_{k-1})$ along the ten dimensional axis, to determine the predicted labels for each test data point using each sample of $\boldsymbol{\Theta}_{k-1}$. We then compare these predicted labels with the actual labels. We achieved 61.079% accuracy for the test data, and 75.330% accuracy for the training data.

5.4 Updated computational complexity of ProxLearn

In the binary case, the creation of matrix $\boldsymbol{C}_k$ requires $O(n_x N^2)$ flops. In the case of multi-class classification concerning m classes, $\boldsymbol{C}_k$ is redefined as the sum of m such matrices. Therefore, creating the updated matrix $\boldsymbol{C}_k$ takes $O(mn_x N^2)$ flops. Thus, updating $\boldsymbol{\varrho}_{k-1}$ is of complexity $O((mn_x + L)N^2)$. The complexity of EULERMARUYAMA can be generalized from the discussion in Sec. 4.

6 Conclusions

This work presents a proximal mean field learning algorithm to train a shallow neural network in the over-parameterized regime. The proposed algorithm is meshless, non-parametric and implements the Wasserstein proximal recursions realizing the gradient descent of entropic-regularized risk. Numerical case studies in binary and multi-class classification demonstrate that the ideas of mean field learning can be attractive as computational framework, beyond purely theoretical interests. Our contribution should be of interest to other learning and variational inference tasks such as the policy optimization and adversarial learning.

We clarify that the proposed algorithm is specifically designed for a neural network with single hidden layer in large width regime. For multi-hidden layer neural networks, the mean field limit in the sense of width as pursued here, is relatively less explored for the training dynamics. In the multiple hidden layer setting, theoretical understanding of the limits is a frontier of current research; see e.g., Fang et al. (2021); Sirignano & Spiliopoulos (2022). Extending these ideas to design variants of proximal algorithms requires new lines of thought, and as such, is out-of-scope of this paper. Pursuing this will comprise our future work.

References

David Alvarez-Melis, Yair Schiff, and Youssef Mroueh. Optimizing functionals on the space of probabilities with input convex neural network. In *Annual Conference on Neural Information Processing Systems*, 2021.

Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2005.

Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pp. 146–155. PMLR, 2017.

Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

Heinz H Bauschke, Patrick L Combettes, et al. *Convex analysis and monotone operator theory in Hilbert spaces*, volume 408. Springer, 2011.

Jean-David Benamou, Guillaume Carlier, and Maxime Laborde. An augmented Lagrangian approach to Wasserstein gradient flows and applications. *ESAIM: Proceedings and surveys*, 54:1–17, 2016.

Dimitri P Bertsekas et al. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2010(1-38):3, 2011.

Clément Bonet, Nicolas Courty, François Septier, and Lucas Drumetz. Sliced-Wasserstein gradient flows. *arXiv preprint arXiv:2110.10972*, 2021.

Clément Bonet, Nicolas Courty, François Septier, and Lucas Drumetz. Efficient gradient flows in sliced-wasserstein space. *arXiv preprint arxiv:2110.10972*, 2022.

Etienne Boursier, Loucas Pillaud-Vivien, and Nicolas Flammarion. Gradient flow dynamics of shallow relu networks for square loss and orthogonal inputs. *Advances in Neural Information Processing Systems*, 35: 20105–20118, 2022.

Yann Brenier. Polar factorization and monotone rearrangement of vector-valued functions. *Communications on pure and applied mathematics*, 44(4):375–417, 1991.

Charlotte Bunne, Laetitia Papaxanthos, Andreas Krause, and Marco Cuturi. Proximal optimal transport modeling of population dynamics. In *International Conference on Artificial Intelligence and Statistics*, pp. 6511–6528. PMLR, 2022.

Kenneth F Caluya and Abhishek Halder. Gradient flow algorithms for density propagation in stochastic systems. *IEEE Transactions on Automatic Control*, 65(10):3991–4004, 2019a.

Kenneth F Caluya and Abhishek Halder. Proximal recursion for solving the Fokker-Planck equation. In *2019 American Control Conference (ACC)*, pp. 4098–4103. IEEE, 2019b.

Kenneth F Caluya and Abhishek Halder. Reflected Schrödinger bridge: Density control with path constraints. In *2021 American Control Conference (ACC)*, pp. 1137–1142. IEEE, 2021a.

Kenneth F Caluya and Abhishek Halder. Wasserstein proximal algorithms for the Schrödinger bridge problem: Density control with nonlinear drift. *IEEE Transactions on Automatic Control*, 67(3):1163–1178, 2021b.

Guillaume Carlier, Vincent Duval, Gabriel Peyré, and BeSchrnhard SchmitzeSchr. Convergence of entropic schemes for optimal transport and gradient flows. *SIAM Journal on Mathematical Analysis*, 49(2):1385–1418, 2017.

René Carmona and François Delarue. *Probabilistic theory of mean field games with applications I-II*. Springer, 2018.

José A Carrillo, Katy Craig, Li Wang, and Chaozhen Wei. Primal dual methods for Wasserstein gradient flows. *Foundations of Computational Mathematics*, 22(2):389–443, 2022.

Lenaic Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. *Advances in neural information processing systems*, 31, 2018.

Casey Chu, Jose Blanchet, and Peter Glynn. Probability functional descent: A unifying perspective on GANS, variational inference, and reinforcement learning. In *International Conference on Machine Learning*, pp. 1213–1222. PMLR, 2019.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Carles Domingo-Enrich, Samy Jelassi, Arthur Mensch, Grant Rotskoff, and Joan Bruna. A mean-field analysis of two-player zero-sum games. *Advances in neural information processing systems*, 33:20215–20226, 2020.

Dheeru Dua and Casey Graff. UCI machine learning machine learning repository. 2017. URL "`http://archive.ics.uci.edu/ml`".

Cong Fang, Jason Lee, Pengkun Yang, and Tong Zhang. Modeling from features: a mean-field framework for over-parameterized deep neural networks. In *Conference on learning theory*, pp. 1887–1936. PMLR, 2021.

Charlie Frogner and Tomaso Poggio. Approximate inference with Wasserstein gradient flows. In *International Conference on Artificial Intelligence and Statistics*, pp. 2581–2590. PMLR, 2020.

Abhishek Halder and Tryphon T Georgiou. Gradient flows in uncertainty propagation and filtering of linear gaussian systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 3081–3088. IEEE, 2017.

Abhishek Halder and Tryphon T Georgiou. Gradient flows in filtering and Fisher-Rao geometry. In *2018 Annual American Control Conference (ACC)*, pp. 4281–4286. IEEE, 2018.

Abhishek Halder and Tryphon T Georgiou. Proximal recursion for the Wonham filter. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 660–665. IEEE, 2019.

Abhishek Halder, Kenneth F Caluya, Bertrand Travacca, and Scott J Moura. Hopfield neural network flow: A geometric viewpoint. *IEEE Transactions on Neural Networks and Learning Systems*, 31(11):4869–4880, 2020.

Abhishek Halder, Kenneth F Caluya, Pegah Ojaghi, and Xinbo Geng. Stochastic uncertainty propagation in power system dynamics using measure-valued proximal recursions. *IEEE Transactions on Power Systems*, 2022.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Richard Jordan, David Kinderlehrer, and Felix Otto. The variational formulation of the Fokker–Planck equation. *SIAM journal on mathematical analysis*, 29(1):1–17, 1998.

Mark Kac. Foundations of kinetic theory. In *Proceedings of The third Berkeley symposium on mathematical statistics and probability*, volume 3, pp. 171–197, 1956.

Johan Karlsson and Axel Ringh. Generalized Sinkhorn iterations for regularizing inverse problems using optimal mass transport. *SIAM Journal on Imaging Sciences*, 10(4):1935–1962, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Maxime Laborde. On some nonlinear evolution systems which are perturbations of Wasserstein gradient flows. *Topological Optimization and Optimal Transport: In the Applied Sciences*, 17:304–332, 2017.

Chang Liu, Jingwei Zhuo, Pengyu Cheng, Ruiyi Zhang, and Jun Zhu. Understanding and accelerating particle-based variational inference. In *International Conference on Machine Learning*, pp. 4082–4092. PMLR, 2019.

Yulong Lu. Two-scale gradient descent ascent dynamics finds mixed nash equilibria of continuous games: A mean-field perspective. In *International Conference on Machine Learning*, pp. 22790–22811. PMLR, 2023.

Henry P McKean Jr. A class of Markov processes associated with nonlinear parabolic equations. *Proceedings of the National Academy of Sciences*, 56(6):1907–1911, 1966.

Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

Petr Mokrov, Alexander Korotin, Lingxiao Li, Aude Genevay, Justin M Solomon, and Evgeny Burnaev. Large-scale Wasserstein gradient flows. *Advances in Neural Information Processing Systems*, 34:15243–15256, 2021.

Youssef Mroueh and Truyen Nguyen. On the convergence of gradient descent in GANs: MMD GAN as a gradient flow. In *International Conference on Artificial Intelligence and Statistics*, pp. 1720–1728. PMLR, 2021.

Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1(3): 127–239, 2014.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Gabriel Peyré. Entropic approximation of Wasserstein gradient flows. *SIAM Journal on Imaging Sciences*, 8(4):2323–2351, 2015.

Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. Wasserstein barycenter and its application to texture mixing. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pp. 435–446. Springer, 2011.

R. Tyrrell Rockafellar. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of operations research*, 1(2):97–116, 1976a.

R Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976b.

Grant Rotskoff and Eric Vanden-Eijnden. Trainability and accuracy of artificial neural networks: An interacting particle system approach. *Communications on Pure and Applied Mathematics*, 75(9):1889–1935, 2022.

Grant M Rotskoff and Eric Vanden-Eijnden. Neural networks as interacting particle systems: Asymptotic convexity of the loss landscape and universal scaling of the approximation error. *stat*, 1050:22, 2018.

Adil Salim, Anna Korba, and Giulia Luise. The Wasserstein proximal gradient algorithm. *Advances in Neural Information Processing Systems*, 33:12356–12366, 2020.

Filippo Santambrogio. {Euclidean, metric, and Wasserstein} gradient flows: an overview. *Bulletin of Mathematical Sciences*, 7(1):87–154, 2017.

Justin Sirignano and Konstantinos Spiliopoulos. Mean field analysis of neural networks: A central limit theorem. *Stochastic Processes and their Applications*, 130(3):1820–1852, 2020.

Justin Sirignano and Konstantinos Spiliopoulos. Mean field analysis of deep neural networks. *Mathematics of Operations Research*, 47(1):120–152, 2022.

Alain-Sol Sznitman. Topics in propagation of chaos. In *Ecole d'été de probabilités de Saint-Flour XIX—1989*, pp. 165–251. Springer, 1991.

Marc Teboulle. Entropic proximal mappings with applications to nonlinear programming. *Mathematics of Operations Research*, 17(3):670–690, 1992.

Anthony C Thompson. On certain contraction mappings in a partially ordered vector space. *Proceedings of the American Mathematical Society*, 14(3):438–443, 1963.

Cédric Villani. *Optimal transport: old and new*, volume 338. Springer, 2009.

Cédric Villani. *Topics in optimal transportation*, volume 58. American Mathematical Soc., 2021.

Sofia Visa, Brian Ramsay, Anca L Ralescu, and Esther Van Der Knaap. Confusion matrix-based feature selection. *MAICS*, 710:120–127, 2011.

Junyu Zhang, Alec Koppel, Amrit Singh Bedi, Csaba Szepesvari, and Mengdi Wang. Variational policy gradient method for reinforcement learning with general utilities. *Advances in Neural Information Processing Systems*, 33:4572–4583, 2020.

Ruiyi Zhang, Changyou Chen, Chunyuan Li, and Lawrence Carin. Policy optimization as Wasserstein gradient flows. In *International Conference on Machine Learning*, pp. 5737–5746. PMLR, 2018.

A Proof of Theorem 1

Our proof follows the general development in Laborde (2017, Sec. 12.3). In the following, we sketch the main ideas.

We have the semi-implicit free energy

$$\hat{F}_\beta(\varrho, \varrho_{k-1}) = \underbrace{\mathbb{E}_\varrho \left[F_0 + V(\boldsymbol{\theta}) + \int_{\mathbb{R}^p} U(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) \varrho_{k-1}(\tilde{\boldsymbol{\theta}}) \mathrm{d}\tilde{\boldsymbol{\theta}} \right]}_{=:\mathcal{V}_{\mathrm{advec}}(\varrho)}$$

$$+\beta^{-1}\mathbb{E}_\varrho \left[\log \varrho\right], \quad k \in \mathbb{N},$$

wherein the summand

$$\mathcal{V}_{\mathrm{advec}}(\varrho) := \mathbb{E}_\varrho \left[F_0 + V(\boldsymbol{\theta}) + \int_{\mathbb{R}^p} U(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) \varrho_{k-1}(\tilde{\boldsymbol{\theta}}) \mathrm{d}\tilde{\boldsymbol{\theta}} \right]$$

is linear in ϱ, and contributes as an effective advection potential energy. The remaining summand $\beta^{-1}\mathbb{E}_\varrho \left[\log \varrho\right]$ results in from diffusion regularization and contributes as an internal energy term.

We note from equation 9 that the functional $\mathcal{V}_{\mathrm{advec}}(\varrho)$ is lower bounded for all $\varrho \in \mathcal{P}_2\left(\mathbb{R}^p\right)$. Furthermore, $\mathcal{V}_{\mathrm{advec}}(\varrho)$ and $\nabla\mathcal{V}_{\mathrm{advec}}(\varrho)$ are uniformly Lipschitz continuous, i.e., there exists $C_1 > 0$ such that

$$\|\nabla\mathcal{V}_{\mathrm{advec}}(\varrho)\|_{L^\infty(\mathbb{R}^p)} + \|\nabla^2\mathcal{V}_{\mathrm{advec}}(\varrho)\|_{L^\infty(\mathbb{R}^p)} \leq C_1$$

for all $\varrho \in \mathcal{P}_2\left(\mathbb{R}^p\right)$ where the constant $C_1 > 0$ is independent of ϱ, and ∇^2 denotes the Euclidean Hessian operator.

Moreover, there exists $C_2 > 0$ such that for all $\varrho, \tilde{\varrho} \in \mathcal{P}_2(\mathbb{R}^p)$, we have

$$\|\nabla\mathcal{V}_{\mathrm{advec}}(\varrho) - \nabla\mathcal{V}_{\mathrm{advec}}(\tilde{\varrho})\|_{L^\infty(\mathbb{R}^p)} \leq C_2 W_2\left(\varrho, \tilde{\varrho}\right).$$

Thus, $\mathcal{V}_{\mathrm{advec}}(\varrho)$ satisfy the hypotheses in Laborde (2017, Sec. 12.2).

For $t \in [0, T]$, we say that $\rho(t, \boldsymbol{\theta}) \in C\left([0, T], \mathcal{P}_2\left(\mathbb{R}^p\right)\right)$ is a weak solution of the IVP equation 14 if for any smooth compactly supported test function $\varphi \in C_c^\infty\left([0, \infty) \times \mathbb{R}^p\right)$, we have

$$\int_0^\infty \int_{\mathbb{R}^p} \left(\left(\frac{\partial \varphi}{\partial t} - \langle \nabla \varphi, \nabla \mathcal{V}_{\text{advec}}(\varrho) \rangle \right) \rho + \beta^{-1} \rho \Delta \varphi \right) \mathrm{d}\boldsymbol{\theta} \mathrm{d}t = - \int_{\mathbb{R}^p} \varphi(t = 0, \boldsymbol{\theta}) \rho_0(\boldsymbol{\theta}). \quad (44)$$

Following Laborde (2017, Sec. 12.2), under the stated hypotheses on $\mathcal{V}_{\text{advec}}(\varrho)$, there exists weak solution of the IVP equation 14 that is continuous w.r.t. the W_2 metric.

The remaining of the proof follows the outline below.

- Using the Dunford-Pettis' theorem, establish that the sequence of functions $\{\varrho_k(h, \boldsymbol{\theta})\}_{k \in \mathbb{N}}$ solving equation 17 is unique.
- Define the interpolation $\varrho_h(t) := \varrho_k(h, \boldsymbol{\theta})$ if $t \in ((k-1)h, kh]$ for all $k \in \mathbb{N}$. Then establish that $\varrho_h(t)$ solves a discrete approximation of equation 44.
- Finally combine the gradient estimates and pass to the limit $h \downarrow 0$, to conclude that $\rho_h(t)$ in this limit solves converges to the weak solution of equation 44 in strong $L^1(\mathbb{R}^p)$ sense.

For the detailed calculations on the passage to the limit, we refer the readers to Laborde (2017, Sec. 12.5). □

B Expressions involving the derivatives of $\boldsymbol{v}$

We define matrices

$$\boldsymbol{T} := \tanh\left(\boldsymbol{W}\boldsymbol{X}^\top + \boldsymbol{b}\boldsymbol{1}^\top\right),$$
$$\boldsymbol{S} := \operatorname{sech}^2\left(\boldsymbol{W}\boldsymbol{X}^\top + \boldsymbol{b}\boldsymbol{1}^\top\right),$$

where $\boldsymbol{1}$ is a vector of all ones of size $n_{\text{data}} \times 1$, and the functions $\tanh(\cdot)$ and $\operatorname{sech}^2(\cdot)$ are elementwise. Notice that $\boldsymbol{T}, \boldsymbol{S} \in \mathbb{R}^{N \times n_{\text{data}}}$. Then

$$\boldsymbol{v} = -\frac{2}{n_{\text{data}}} \boldsymbol{a} \odot \left(\tanh(\boldsymbol{W}\boldsymbol{X}^\top + \boldsymbol{b}\boldsymbol{1}^\top)\boldsymbol{y}\right) = -\frac{2}{n_{\text{data}}} \boldsymbol{a} \odot (\boldsymbol{T}\boldsymbol{y}).$$

Proposition 2. *With the above notations in place, we have*

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{a}} \boldsymbol{1} = \sum_{k=1}^N \frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{a}} = -\frac{2}{n_{\text{data}}} \boldsymbol{T}\boldsymbol{y}, \quad (45)$$

and

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{b}} \boldsymbol{1} = \sum_{k=1}^N \frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{b}} = -\frac{2}{n_{\text{data}}} \boldsymbol{a} \odot \boldsymbol{S}\boldsymbol{y}. \quad (46)$$

Furthermore,

$$\sum_{k=1}^N \frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{W}} = -\frac{2}{n_{\text{data}}} \left[\left(\boldsymbol{a}\boldsymbol{1}^\top\right) \odot \left(\boldsymbol{S}\left(\boldsymbol{X} \odot \boldsymbol{y}\boldsymbol{1}^\top\right)\right) \right]. \quad (47)$$

Proof. The kth element of $\boldsymbol{v}$ is $\boldsymbol{v}_k = -\frac{2}{n_{\text{data}}}\boldsymbol{a}_k \sum_{i=1}^{n_{\text{data}}} [\boldsymbol{T}_{k,i}\boldsymbol{y}_i]$. Thus,

$$\frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{a}_j} = \begin{cases} 0 & \text{for} \quad k \neq j, \\ -\frac{2}{n_{\text{data}}} \sum_{i=1}^{n_{\text{data}}} [\boldsymbol{T}_{k,i}\boldsymbol{y}_i] & \text{for} \quad k = j. \end{cases}$$

So the matrix $\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{a}}$ is diagonal, and

$$\left[\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{a}}\mathbf{1}\right]_k = -\frac{2}{n_{\text{data}}} \sum_{i=1}^{n_{\text{data}}} [\boldsymbol{T}_{k,i}\boldsymbol{y}_i] = -\frac{2}{n_{\text{data}}} [\boldsymbol{T}\boldsymbol{y}]_k \,.$$

Hence, we obtain

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{a}}\mathbf{1} = -\frac{2}{n_{\text{data}}}\boldsymbol{T}\boldsymbol{y},$$

which is (45).

On the other hand,

$$\begin{aligned} \frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{b}_k} &= \frac{\partial}{\partial \boldsymbol{b}_k}\left[-\frac{2}{n_{\text{data}}}\boldsymbol{a}_k \sum_{i=1}^{n_{\text{data}}} [\boldsymbol{T}_{k,i}\boldsymbol{y}_i]\right] \\ &= -\frac{2}{n_{\text{data}}}\boldsymbol{a}_k \sum_{i=1}^{n_{\text{data}}} \frac{\partial}{\partial \boldsymbol{b}_k} [\boldsymbol{T}_{k,i}\boldsymbol{y}_i] \,. \end{aligned}$$

Note that

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{b}_k} [\boldsymbol{T}_{k,i}\boldsymbol{y}_i] &= \frac{\partial}{\partial \boldsymbol{b}_k} \tanh\left(\sum_{j=1}^{n_x} (\boldsymbol{W}_{k,j}\boldsymbol{X}_{i,j}) + \boldsymbol{b}_k\right)\boldsymbol{y}_i \\ &= \operatorname{sech}^2\left(\sum_{j=1}^{n_x} (\boldsymbol{W}_{k,j}\boldsymbol{X}_{i,j}) + \boldsymbol{b}_k\right)\boldsymbol{y}_i = \boldsymbol{S}_{k,i}\boldsymbol{y}_i. \end{aligned}$$

Therefore,

$$\frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{b}_j} = \begin{cases} 0 & \text{for} \quad k \neq j, \\ -\frac{2}{n_{\text{data}}}\boldsymbol{a}_k \sum_{i=1}^{n_{\text{data}}} [\boldsymbol{S}_{k,i}\boldsymbol{y}_i] & \text{for} \quad k = j. \end{cases}$$

As the matrix $\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{b}}$ is diagonal, we get

$$\left[\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{b}}\mathbf{1}\right]_k = -\frac{2}{n_{\text{data}}}\boldsymbol{a}_k \sum_{i=1}^{n_{\text{data}}} [\boldsymbol{S}_{k,i}\boldsymbol{y}_i] = -\frac{2}{n_{\text{data}}}\boldsymbol{a}_k [\boldsymbol{S}\boldsymbol{y}]_k \,,$$

and so

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{b}}\mathbf{1} = -\frac{2}{n_{\text{data}}}\boldsymbol{a} \odot \boldsymbol{S}\boldsymbol{y},$$

which is indeed (46).

Likewise, we take an element-wise approach to the derivatives with respect to weights $\boldsymbol{W}_{k,m}$. Note that such a weight $\boldsymbol{W}_{k,m}$ will only appear in the kth element of $\boldsymbol{v}$, and so we only need to compute

$$\frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{W}_{k,m}} = -\frac{2}{n_{\text{data}}}\boldsymbol{a}_k \sum_{i=1}^{n_{\text{data}}} \frac{\partial}{\partial \boldsymbol{W}_{k,m}} [\boldsymbol{T}_{k,i}\boldsymbol{y}_i] \,.$$

Since

$$\frac{\partial}{\partial \boldsymbol{W}_{k,m}}\left[\boldsymbol{T}_{k,i}\boldsymbol{y}_i\right] = \frac{\partial}{\partial \boldsymbol{W}_{k,m}} \tanh\left(\sum_{j=1}^{n_x}(\boldsymbol{W}_{k,j}\boldsymbol{X}_{j,i}) + \boldsymbol{b}_k\right)\boldsymbol{y}_i$$
$$= \operatorname{sech}^2\left(\sum_{j=1}^{n_x}(\boldsymbol{W}_{k,j}\boldsymbol{X}_{i,j}) + \boldsymbol{b}_k\right)\boldsymbol{X}_{i,m}\boldsymbol{y}_i = \boldsymbol{S}_{k,i}\boldsymbol{X}_{i,m}\boldsymbol{y}_i,$$

we have

$$\frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{W}_{m,j}} = \begin{cases} 0 & \text{for} \quad k \neq m, \\ -\frac{2}{n_{\text{data}}}\boldsymbol{a}_k \sum_{i=1}^{n_{\text{data}}}\left[\boldsymbol{S}_{k,i}\boldsymbol{X}_{i,m}\boldsymbol{y}_i\right] & \text{for} \quad k = m. \end{cases}$$

Thus,

$$\sum_{k=1}^{N}\frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{W}_{m,j}} = -\frac{2}{n_{\text{data}}}\boldsymbol{a}_m \sum_{i=1}^{n_{\text{data}}}\left[\boldsymbol{S}_{m,i}\boldsymbol{X}_{i,m}\boldsymbol{y}_i\right]$$
$$= -\frac{2}{n_{\text{data}}}\boldsymbol{a}_m\left[\boldsymbol{S}\left(\boldsymbol{X}\odot\left(\boldsymbol{y}\boldsymbol{1}^\top\right)\right)\right]_{m,j}.$$

Therefore, considering $\boldsymbol{1} \in \mathbb{R}^{n_x}$, we write

$$\sum_{k=1}^{N}\frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{W}} = -\frac{2}{n_{\text{data}}}\left[\left(\boldsymbol{a}\boldsymbol{1}^\top\right)\odot\left(\boldsymbol{S}\left(\boldsymbol{X}\odot\boldsymbol{y}\boldsymbol{1}^\top\right)\right)\right],$$

thus arriving at (47). This completes the proof. □

C Expressions involving the derivatives of $\boldsymbol{u}$

Expressions involving the derivatives of $\boldsymbol{u}$, are summarized in the Proposition next. These results find use in Sec. 4. We start by noting that

$$\boldsymbol{u} = \frac{1}{n_{\text{data}}}(\boldsymbol{1}^\top\boldsymbol{a}\odot\boldsymbol{T})(\boldsymbol{1}^\top\boldsymbol{a}\odot\boldsymbol{T})^\top\boldsymbol{\rho}$$
$$= \frac{1}{n_{\text{data}}}(\boldsymbol{1}^\top\boldsymbol{a}\odot\boldsymbol{T})(\boldsymbol{a}^\top\boldsymbol{1}\odot\boldsymbol{T}^\top)\boldsymbol{\rho}.$$

Proposition 3. *With the above notations in place, we have*

$$\underbrace{\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{a}}}_{N\times N}\underbrace{\boldsymbol{1}}_{N\times 1} = \frac{1}{n_{\text{data}}}\left[\left(\left(\boldsymbol{\rho}\boldsymbol{a}^\top\right)\odot\left(\boldsymbol{T}\boldsymbol{T}^\top\right)\right)\boldsymbol{1} + \boldsymbol{1}\left(\boldsymbol{a}\odot\boldsymbol{\varrho}\right)^\top\boldsymbol{T}\boldsymbol{T}^\top\boldsymbol{1}\right], \tag{48}$$

and

$$\underbrace{\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{b}}}_{N\times N}\underbrace{\boldsymbol{1}}_{N\times 1} = \frac{1}{n_{\text{data}}}\left[\left(\left(\boldsymbol{a}\boldsymbol{1}^\top\right)\odot\left(\boldsymbol{S}\boldsymbol{T}^\top\right)\odot\left(\boldsymbol{1}\left(\boldsymbol{a}\odot\boldsymbol{\varrho}\right)^\top\right)\right)\boldsymbol{1}\right.$$
$$\left. + \left(\left(\boldsymbol{1}\boldsymbol{a}^\top\right)\odot\left(\boldsymbol{S}\boldsymbol{T}^\top\right)\odot\left(\left(\boldsymbol{a}\odot\boldsymbol{\varrho}\right)\boldsymbol{1}^\top\right)\right)\boldsymbol{1}\right]. \tag{49}$$

Furthermore,

$$\sum_{k=1}^{N}\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{W}_{i,j}} = \frac{1}{n_{\text{data}}}\sum_{k=1}^{N}\sum_{m=1}^{n_{\text{data}}} a_i a_k(\varrho_i + \varrho_k)T_{k,m}S_{i,m}X_{m,j}. \tag{50}$$

Proof. Letting $\boldsymbol{t}_i^\top$ denote the ith row of $\boldsymbol{T}$, we rewrite $\boldsymbol{u}$ as follows:

$$\begin{aligned}\boldsymbol{u} &= \frac{1}{n_{\text{data}}}\begin{bmatrix} a_1\boldsymbol{t}_1^\top \\ \vdots \\ a_N\boldsymbol{t}_N^\top \end{bmatrix}\begin{bmatrix}\rho_1 a_1\boldsymbol{t}_1 + \ldots + \rho_N a_N \boldsymbol{t}_N\end{bmatrix} \\ &= \frac{1}{n_{\text{data}}}\begin{bmatrix} a_1\boldsymbol{t}_1^\top(\rho_1 a_1\boldsymbol{t}_1 + \ldots + \rho_N a_N \boldsymbol{t}_N) \\ \vdots \\ a_N\boldsymbol{t}_N^\top(\rho_1 a_1\boldsymbol{t}_1 + \ldots + \rho_N a_N \boldsymbol{t}_N) \end{bmatrix}.\end{aligned}$$

For $i \neq k$, we thus have

$$\frac{\partial \boldsymbol{u}_i}{\partial \boldsymbol{a}_k} = \frac{1}{n_{\text{data}}} a_i \boldsymbol{t}_i^\top(\rho_k \boldsymbol{t}_k) = \frac{1}{n_{\text{data}}} a_i \rho_k \boldsymbol{t}_i^\top \boldsymbol{t}_k.$$

Likewise, for $i = k$, we have

$$\frac{\partial \boldsymbol{u}_k}{\partial \boldsymbol{a}_k} = \frac{1}{n_{\text{data}}} a_k \rho_k \boldsymbol{t}_k^\top \boldsymbol{t}_k + \frac{1}{n_{\text{data}}} \boldsymbol{t}_k^\top(\rho_1 a_1 \boldsymbol{t}_1 + \ldots + \rho_N a_N \boldsymbol{t}_N).$$

Combining the above, we obtain $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{a}}$, and hence (48) follows.

On the other hand, for $i \neq k$, we have

$$\frac{\partial \boldsymbol{u}_i}{\partial \boldsymbol{b}_k} = \frac{1}{n_{\text{data}}} a_i \boldsymbol{t}_i^\top \rho_k a_k \frac{\partial \boldsymbol{t}_k}{\partial \boldsymbol{b}_k} = \frac{1}{n_{\text{data}}} a_i \boldsymbol{t}_i^\top \rho_k a_k \boldsymbol{s}_k,$$

and for $i = k$, we obtain

$$\begin{aligned}\frac{\partial \boldsymbol{u}_i}{\partial \boldsymbol{b}_k} &= \frac{1}{n_{\text{data}}} a_i \left(\frac{\partial \boldsymbol{t}_k}{\partial \boldsymbol{b}_k}\right)^\top (\rho_1 a_1 \boldsymbol{t}_1 + \ldots + \rho_N a_N \boldsymbol{t}_N) \\ &\qquad + \frac{1}{n_{\text{data}}} a_k \boldsymbol{t}_k^\top \rho_k a_k \frac{\partial \boldsymbol{t}_k}{\partial \boldsymbol{b}_k} \\ &= \frac{1}{n_{\text{data}}} a_i (\boldsymbol{s}_k)^\top (\rho_1 a_1 \boldsymbol{t}_1 + \ldots + \rho_N a_N \boldsymbol{t}_N) \\ &\qquad + \frac{1}{n_{\text{data}}} a_k \boldsymbol{t}_k^\top \rho_k a_k \boldsymbol{s}_k.\end{aligned}$$

Combining the above, we obtain $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{b}}$, and hence (49) follows.

Finally, noting that $\boldsymbol{W}_{i,j}$ is in the ith row of $\boldsymbol{T}$, for $i \neq k$, we obtain

$$\frac{\partial \boldsymbol{u}_k}{\partial \boldsymbol{W}_{i,j}} = \frac{1}{n_{\text{data}}} a_k \boldsymbol{t}_k^\top \rho_i a_i \frac{\partial \boldsymbol{t}_i}{\partial \boldsymbol{W}_{i,j}} = \frac{1}{n_{\text{data}}} a_k \boldsymbol{t}_k^\top \rho_i a_i (\boldsymbol{s}_i \odot \boldsymbol{x}_j),$$

where $\boldsymbol{x}_j$ is the jth column of $\boldsymbol{X}$. Likewise, for $i = k$, we get

$$\begin{aligned}\frac{\partial \boldsymbol{u}_k}{\partial \boldsymbol{W}_{i,j}} &= \frac{1}{n_{\text{data}}} a_k \left(\frac{\partial \boldsymbol{t}_k}{\partial \boldsymbol{W}_{i,j}}\right)^\top (\rho_1 a_1 \boldsymbol{t}_1 + \ldots + \rho_N a_N \boldsymbol{t}_N) \\ &\qquad + \frac{1}{n_{\text{data}}} a_k \boldsymbol{t}_k^\top \rho_k a_k \frac{\partial \boldsymbol{t}_k}{\partial \boldsymbol{W}_{i,j}} \\ &= \frac{1}{n_{\text{data}}} a_k (\boldsymbol{s}_i \odot \boldsymbol{x}_j)^\top (\rho_1 a_1 \boldsymbol{t}_1 + \ldots + \rho_N a_N \boldsymbol{t}_N) \\ &\qquad + \frac{1}{n_{\text{data}}} a_k \boldsymbol{t}_k^\top \rho_k a_k (\boldsymbol{s}_i \odot \boldsymbol{x}_j).\end{aligned}$$

Combining the above, we obtain $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{W}_{i,j}}$, thereby arriving at (50). □