

Learning Set Functions with Implicit Differentiation

Gözde Özcan, Chengzhi Shi, Stratis Ioannidis

Northeastern University, Boston, MA 02115, USA
{gozcan, cshi, ioannidis}@ece.neu.edu

Abstract

A recent work introduces the problem of learning set functions from data generated by a so-called optimal subset oracle. Their approach approximates the underlying utility function with an energy-based model, whose parameters are estimated via mean-field variational inference. This approximation reduces to fixed point iterations; however, as the number of iterations increases, automatic differentiation quickly becomes computationally prohibitive due to the size of the Jacobians that are stacked during backpropagation. We address this challenge with implicit differentiation and examine the convergence conditions for the fixed-point iterations. We empirically demonstrate the efficiency of our method on synthetic and real-world subset selection applications including product recommendation, set anomaly detection and compound selection tasks.

1 Introduction

Many interesting applications operate with set-valued outputs and/or inputs. Examples include product recommendation (Bonab et al. 2021; Schafer, Konstan, and Riedl 1999), compound selection (Ning, Walters, and Karypis 2011), set matching (Saito et al. 2020), set retrieval (Feng, Zhou, and Lan 2016), point cloud processing (Zhao et al. 2019; Giannis, Gunopulos, and Koudas 2001), set prediction (Zhang, Hare, and Prugel-Bennett 2019), and set anomaly detection (Mašková et al. 2024), to name a few. Several recent works (Zaheer et al. 2017; Lee et al. 2019) apply neural networks to learn set functions from input/function value pairs, assuming access to a dataset generated by a *function value* oracle. In other words, they assume having access to a dataset generated by an oracle that evaluates the value of the set function for any given input set.

Recently, Ou et al. (2022) proposed an approximate maximum likelihood estimation framework under the supervision of a so-called *optimal subset* oracle. In contrast to traditional function value oracles, a label produced by an optimal subset oracle is the subset that maximizes an (implicit) utility set function, in the face of several alternatives. The goal of inference is to learn, in a parametric form, this utility function, under which observed oracle selections are optimal. As MLE is intractable in this setting, Ou et al. (2022) propose performing variational inference instead. In turn, they show that approximating the distribution of oracle selections requires solving a fixed-point equation per sample. However,

these fixed-point iterations may diverge in practice. In addition, Ou et al. (2022) implement these iterations via *loop unrolling*, i.e., by stacking up neural network layers across iterations, and calculating the gradient using automatic differentiation; this makes backpropagation expensive, limiting their experiments to only a handful of iterations.

In this work, we establish a condition under which the fixed-point iterations proposed by Ou et al. (2022) are guaranteed to converge. We also propose a more effective gradient computation utilizing the recent advances in implicit differentiation (Bai, Kolter, and Koltun 2019; Kolter, Duvenaud, and Johnson 2020; Huang, Bai, and Kolter 2021), instead of unrolling the fixed-point iterations via automatic differentiation (Paszke et al. 2017). This corresponds to differentiating after infinite fixed point iterations, while remaining tractable; we experimentally show that this improves the predictive performance of the inferred models.

We make the following contributions:

- We prove that, as long as the multilinear relaxation (Calinescu et al. 2011) of the objective function is bounded, and this bound is inversely proportional to the size of the ground set, the fixed-point iterations arising during the MLE framework introduced by Ou et al. (2022) converge to a unique solution, regardless of the starting point.
- We propose a more effective gradient computation by using implicit differentiation instead of unrolling the fixed-point iterations via automatic differentiation. To the best of our knowledge, we are the first ones to propose utilizing implicit differentiation in the context of learning set functions.
- We conduct experiments to show the advantage of our approach on multiple subset selection applications including set anomaly detection, product recommendation, and compound selection tasks. We also show, in practice, that the fixed-point iterations converge by normalizing the gradient of the multilinear relaxation.

The remainder of the paper is organized as follows. We present related literature in Sec. 2. We summarize the learning set functions with optimal subset oracle setting introduced by Ou et al. (2022) in Sec. 3. We state our main contributions in Sec. 4. We present our experimental results in Sec. 5 and we conclude in Sec. 6.

2 Related Work

Learning Set Functions from Oracles. There is a line of work where a learning algorithm is assumed to have access to the value of an unknown utility function for a given set (Feldman and Kothari 2014; Balcan and Harvey 2018; Zaheer et al. 2017; Lee et al. 2019; Wendler et al. 2021; De and Chakrabarti 2022). This is the *function value oracle* setting. Zaheer et al. (2017) and De and Chakrabarti (2022) regress over input set - function value pairs by minimizing the squared loss of the predictions while Lee et al. (2019) minimize the mean absolute error. However, obtaining a function value to a given subset is not an easy task for real-world applications. The value of a set may not be straightforward to quantify or can be expensive to compute. Alternatively, Tschitschek, Sahin, and Krause (2018) and Ou et al. (2022) assume having access to an *optimal subset oracle* for a given ground set in the training data. Similarly, we do not learn the objective function explicitly from input set - output value pairs. We learn it implicitly in the optimal subset oracle setting.

Learning Set Functions with Neural Networks. Multiple works aim to extend the capabilities of neural networks for functions on discrete domains, i.e., *set functions* (Zaheer et al. 2017; Wendler, Püschel, and Alistarh 2019; Soelch et al. 2019; Lee et al. 2019; Wagstaff et al. 2019; Kim et al. 2021; Zhang et al. 2022a; Giannone and Winther 2022). Diverging from the traditional paradigm where the input data is assumed to be in a fixed dimensional vector format, set functions are characterized by their *permutation invariance*, i.e., the output of a set does not depend on the order of its elements. We refer the reader to a survey about permutation-invariant networks by Kimura et al. (2024) for a more detailed overview. In this work, we also enforce permutation invariance by combining the energy-based model in Sec. 3.1 with deep sets (Zaheer et al. 2017), following the proposed method of Ou et al. (2022) (see also App. A of Özcan, Shi, and Ioannidis (2024)).

Karalias et al. (2022) integrate neural networks with set functions by leveraging extensions of these functions to the continuous domain. Note that, their goal is not to learn a set function but to learn *with* a set function, which differs from our objective.

Learning Submodular Functions. It is common to impose some structure on the objective when learning set functions. The underlying objective is often assumed to be submodular, i.e., it exhibits a diminishing returns property, while the parameters of such function are typically learned from function value oracles (Dolhansky and Bilmes 2016; Bilmes and Bai 2017; Djolonga and Krause 2017; Kothawade et al. 2020; De and Chakrabarti 2022; Bhatt, Das, and Bilmes 2024; Gomez-Rodriguez, Leskovec, and Krause 2012; Bach 2013; Feldman and Kothari 2014; He et al. 2016). We do not make such assumptions, therefore, our results are applicable to a broader class of set functions.

Implicit Differentiation. In the context of machine learning, implicit differentiation is used in hyperparameter optimization (Lorraine, Vicol, and Duvenaud 2020; Bertrand et al. 2020), optimal control (Xu, Molloy, and Gould 2024), rein-

forcement learning (Nikishin et al. 2022), bi-level optimization (Arbel and Mairal 2022; Zucchet and Sacramento 2022), neural ordinary differential equations (Chen et al. 2018; Li et al. 2020) and set prediction (Zhang et al. 2022b), to name a few. Inspired by the advantages observed over this wide-range of problems, we use implicit differentiation, i.e., a method for differentiating a function that is given implicitly (Krantz and Parks 2002), to learn set functions for subset selection tasks by leveraging the JAX-based, modular automatic implicit differentiation tool provided by Blondel et al. (2022).

Implicit Layers. Instead of specifying the output of a deep neural network layer as an explicit function over its inputs, *implicit layers* are specified implicitly, via the conditions that layer outputs and inputs must jointly satisfy (Kolter, Duvenaud, and Johnson 2020). Deep Equilibrium Models (DEQs) (Bai, Kolter, and Koltun 2019) and their variants (Winston and Kolter 2020; Huang, Bai, and Kolter 2021; Sittoni and Tudisco 2024) directly compute the fixed-point resulting from stacking up hidden implicit layers by black-box root-finding methods, while also directly differentiating through the stacked fixed-point equations via implicit differentiation. We adapt this approach when satisfying the fixed-point constraints arising in our setting. The main difference is that in the aforementioned works, implicit layers correspond to a weight-tied feedforward network while in our case, they correspond to a deep set (Zaheer et al. 2017) style architecture.

3 Problem Setup

In the setting introduced by Ou et al. (2022), the aim is to learn set functions from a dataset generated by a so-called *optimal subset oracle*. The dataset \mathcal{D} consists of sample pairs of the form (S^*, V) , where (query) $V \subseteq \Omega$ is a set of *options*, i.e., items from a universe Ω and (response) S^* is the *optimal subset* of V , as selected by an oracle. We further assume that each item is associated with a feature vector of dimension d_f , i.e., $\Omega \subseteq \mathbb{R}^{d_f}$. The goal is to learn a set function $F_\theta : 2^\Omega \times 2^\Omega \rightarrow \mathbb{R}$, parameterized by $\theta \in \mathbb{R}^d$, modeling the utility of the oracle, so that

$$S^* = \arg \max_{S \subseteq V} F_\theta(S, V), \quad (1)$$

for all pairs $(S^*, V) \in \mathcal{D}$. As a motivating example, consider the case of product recommendations. Given a ground set V of possible products to recommend, a recommender selects an optimal subset $S^* \subseteq V$ and suggests these to a user. In this setting, the function $F_\theta(S, V)$ captures, e.g., the recommender objective, the utility of the user, etc. Having access to a dataset of such pairs, the goal is to learn F_θ , effectively reverse-engineering the objective of the recommender engine, inferring the user’s preferences, etc.

3.1 MLE with Energy-Based Modeling

Ou et al. (2022) propose an approximate maximum likelihood estimation (MLE) by modeling oracle behavior via a Boltzmann energy (i.e., soft-max) model (Murphy 2012; Mnih and Hinton 2005; Hinton et al. 2006; LeCun et al. 2006).

They assume that the oracle selection is probabilistic, and the probability that S is selected given options V is given by:

$$p_{\theta}(S|V) = \frac{\exp(F_{\theta}(S, V))}{\sum_{S' \subseteq V} \exp(F_{\theta}(S', V))}. \quad (2)$$

This is equivalent to Eq. (1), presuming that the utility $F_{\theta(\cdot)}$ is distorted by Gumbel noise (Kirsch et al. 2023). Then, given a dataset $\mathcal{D} = \{(S_i^*, V_i)\}_{i=1}^N$, MLE amounts to:

$$\arg \max_{\theta} \sum_{i=1}^N [\log p_{\theta}(S_i^* | V_i)]. \quad (3)$$

Notice that multiplying F_{θ} with a constant $c > 0$ makes no difference in the behavior of the optimal subset oracle in Eq. (1): the oracle would return the same decision under arbitrary re-scaling. However, using $c \cdot F_{\theta}(\cdot)$ in the energy-based model of Eq. (2) corresponds to setting a temperature parameter c in the Boltzmann distribution (Murphy 2012; Kirsch et al. 2023), interpolating between the deterministic selection ($c \rightarrow \infty$) in Eq. (1) and the uniform distribution ($c \rightarrow 0$).¹

3.2 Variational Approximation of Energy-Based Models

Learning θ by MLE is challenging precisely due to the exponential number of terms in the denominator of Eq. (2). Instead, Ou et al. (2022) construct an alternative optimization objective via mean-field variational inference as follows. First, they introduce a mean field variational approximation of the density p_{θ} given by $q(S, V, \psi) = \prod_{j \in S} \psi_j \prod_{j \in V \setminus S} (1 - \psi_j)$, parameterized by the probability vector ψ : this represents the probability that each element $j \in V$ is in the optimal subset S^* . Then, estimation via variational inference amounts to the following optimization problem:

$$\text{Min. } \mathcal{L}(\{\psi_i^*\}) = \mathbb{E}_{\mathbb{P}(V, S)}[-\log q(S, V, \psi^*)] \approx \quad (4)$$

$$\frac{1}{N} \sum_{i=1}^N \left(-\sum_{j \in S_i^*} \log \psi_{ij}^* - \sum_{j \in V_i \setminus S_i^*} \log (1 - \psi_{ij}^*) \right),$$

$$\text{subj. to } \psi_i^* = \arg \min_{\psi} \mathbb{KL}(q(S_i, V_i, \psi) || p_{\theta}(S_i | V_i)),$$

for all $i \in \{1, \dots, n\}$,

where $\psi_i^* \in [0, 1]^{|V|}$ is the probability vector of elements in V_i being included in S_i , $\mathbb{KL}(\cdot || \cdot)$ is the Kullback-Leibler divergence, and $p_{\theta}(\cdot)$ is the energy-based model defined in Eq. (2). In turn, this is found through the ELBO maximization process we discuss in the next section.

3.3 ELBO Maximization

To compute ψ^* , Ou et al. (2022) show that minimizing the constraint in Eq. (4) via maximizing the corresponding evidence lower bound (ELBO) reduces to solving a fixed

¹From a Bayesian point of view, multiplying $F_{\theta}(\cdot)$ with $c > 0$ yields the posterior predictive distribution under an uninformative Dirichlet conjugate prior per set with parameter $\alpha = e^c$ (Murphy 2012).

Algorithm 1: DiffMF (Ou et al. 2022)

Input: training dataset $\{(S_i^*, V_i)\}_{i=1}^N$, learning rate η , number of samples m
Output: parameter θ

- 1: $\theta \leftarrow$ initialize
- 2: **repeat**
- 3: sample training data point $(S^*, V) \sim \{(S_i^*, V_i)\}_{i=1}^N$
- 4: initialize the variational parameter $\psi^{(0)} \leftarrow 0.5 * \mathbf{1}$
- 5: **repeat**
- 6: **for** $k \leftarrow 1, \dots, K$ **do**
- 7: **for** $j \leftarrow 1, \dots, |V|$ in parallel **do**
- 8: sample m subsets $S_{\ell} \sim q(S, (\psi^{(k-1)} | \psi_j^{(k-1)} \leftarrow 0))$
- 9: update variational parameter $\psi_j^{(k)} \leftarrow \sigma(\frac{1}{m} \sum_{\ell=1}^m [F_{\theta}(S_{\ell} \cup \{j\}) - F_{\theta}(S_{\ell})])$
- 10: **end for**
- 11: **end for**
- 12: **until** convergence of ψ
- 13: update parameter θ by unfolding the derivatives of the K -layer meta-network resulting from the fixed-point equations given in Eq. (8) during SGD

$$\partial_{\theta} \psi^{(K)}(\theta) \leftarrow \underbrace{\sigma(\nabla_{\psi^{(K-1)}} \tilde{F}(\dots(\sigma(\nabla_{\psi^{(0)}} \tilde{F}(\psi^{(0)})) \dots))}_{K \text{ nested functions}}$$

$$\nabla_{\theta} \mathcal{L}(\psi^{(K)}, \theta) \leftarrow \nabla_{\psi^{(K)}} \mathcal{L}(\psi^{(K)}(\theta)) \cdot \partial_{\theta} \psi^{(K)}(\theta)$$

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}(\psi^{(K)}, \theta)$$

- 14: **until** convergence of θ

point equation. In particular, omitting the dependence on i for brevity, the constraint in Eq. (4) is equivalent to the following ELBO maximization (Kingma and Welling 2013; Blei, Kucukelbir, and McAuliffe 2017):

$$\max_{\psi} \tilde{F}(\psi, \theta) + \mathbb{H}(q(S, V, \psi)), \quad (5)$$

where $\mathbb{H}(\cdot)$ is the entropy and $\tilde{F} : [0, 1]^{|V|} \times \mathbb{R}^d \rightarrow \mathbb{R}$ is the so-called multilinear extension of $F_{\theta}(S, V)$ (Calinescu et al. 2011), given by:

$$\tilde{F}(\psi, \theta) = \sum_{S \subseteq V} F_{\theta}(S, V) \prod_{j \in S} \psi_j \prod_{j \in V \setminus S} (1 - \psi_j). \quad (6)$$

Ou et al. (2022) show that a stationary point maximizing the ELBO in Eq. (5) must satisfy:

$$\psi - \sigma(\nabla_{\psi} \tilde{F}(\psi, \theta)) = 0, \quad (7)$$

where the function $\sigma : \mathbb{R}^{|V|} \rightarrow \mathbb{R}^{|V|}$ is defined as $\sigma(\mathbf{x}) = [\sigma(x_j)]_{j=1}^{|V|}$ and $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is the sigmoid function, i.e., $\sigma(x) = (1 + \exp(-x))^{-1}$. The detailed derivation of this condition can be found in App. C.1 of Özcan, Shi, and Ioannidis (2024).

Observing that the stationary condition in Eq. (7) is a fixed point equation, Ou et al. (2022) propose solving it via the following fixed-point iterations. Given $\theta \in \mathbb{R}^d$,

$$\begin{aligned} \psi^{(0)} &\leftarrow \text{Initialize in } [0, 1]^{|V|}, \\ \psi^{(k)} &\leftarrow \sigma(\nabla_{\psi} \tilde{F}(\psi^{(k-1)}, \theta)), \\ \psi^* &\leftarrow \psi^{(K)}, \end{aligned} \quad (8)$$

where $k \in \mathbb{N}$, and K is the number of iterations. The exact computation of the multilinear relaxation defined in Eq. (6) requires an exponential number of terms in the size of V . However, it is possible to efficiently estimate both the multilinear relaxation and its gradient $\nabla_{\psi} \tilde{F}(\psi, \theta)$ via Monte Carlo sampling (see App. C.2 of Özcan, Shi, and Ioannidis (2024) for details).

3.4 DiffMF and Variants

Putting everything together yields the DiffMF algorithm introduced by Ou et al. (2022). For completeness, we summarize this procedure in Alg. 1. In short, they implement the fixed-point iterative update steps in Eq. (8) by executing a fixed number of iterations K , given θ , and unrolling the loop: in their implementation, this amounts to stacking up K layers, each involving an estimate of the gradient of the multilinear relaxation via sampling, and thereby multiple copies of a neural network representing $F_{\theta}(\cdot)$ (one per sample). Subsequently, this extended network is entered in the loss given in Eq. (4), which is minimized w.r.t. θ via SGD.

They also introduce two variants of this algorithm, regressing also $\psi^{(0)}$ as a function of the item features via an extra recognition network, assuming the latter are independent (terming inference in this setting as $\text{EquiVSet}_{\text{ind}}$) or correlated by a Gaussian copula (Sklar 1973; Nelsen 2006) (termed $\text{EquiVSet}_{\text{copula}}$). Compared to DiffMF, both translate to additional initial layers and steps per epoch.

3.5 Challenges

The above approach by Ou et al. (2022), and its variants, have two drawbacks. First, the fixed-point iterative updates given in Eq. (8) are not guaranteed to converge to an optimal solution. We indeed frequently observed divergence experimentally, in practice. Without convergence and uniqueness guarantees, the quality of the output, $\psi^{(K)}$, is heavily dependent on the selection of the starting point, $\psi^{(0)}$. Moreover, as these iterations correspond to stacking up layers, each containing multiple copies of $F_{\theta}(\cdot)$ due to sampling, back-propagation is computationally prohibitive both in terms of time as well as space complexity. In fact, poor performance due to lack of convergence, as well as computational considerations, led Ou et al. to set the number of iterations to $K \leq 5$ (even $K = 1$) in their experiments. We address both of these challenges in the next section.

4 Our Approach

Recall from the previous section that minimizing the constraint of the optimization problem given in Eq. (4) is the

equivalent of the ELBO in Eq. (5), and the stationary condition of optimizing this ELBO reduces to Eq. (7). Stitching everything together, we wish to solve the following optimization problem:

$$\begin{aligned} \text{Min.}_{\{\psi_i^*\}, \theta} \mathcal{L}(\{\psi_i^*\}) &\approx \quad (9) \\ \frac{1}{N} \sum_{i=1}^N \left(- \sum_{j \in S_i^*} \log \psi_{ij} - \sum_{j \in V_i \setminus S_i^*} \log (1 - \psi_{ij}) \right), \end{aligned}$$

subj. to $\psi_i^* = \sigma(\nabla_{\psi} \tilde{F}(\psi_i^*, \theta))$, for all $i \in \{i, \dots, n\}$.

To achieve this goal, we (a) establish conditions under which iterations of Eq. (8) converge to a unique solution, by utilizing the Banach fixed-point theorem and (b) establish a way to efficiently compute the gradient of the loss at the fixed-point by using the implicit function theorem. Our results pave the way to utilize recent tools developed in the context of implicit differentiation (Bai, Kolter, and Koltun 2019; Kolter, Duvenaud, and Johnson 2020; Blondel et al. 2022) to the setting of Ou et al. (2022).

4.1 Convergence Condition for the Fixed-Point

Fixed-points can be attracting, repelling, or neutral (Davies 2018; Reznitzner 2003). We characterize the condition under which the convergence is guaranteed in the following assumption.

Assumption 4.1. Consider the multilinear relaxation $\tilde{F} : [0, 1]^{|V|} \times \mathbb{R}^d \rightarrow \mathbb{R}$ of $F_{\theta}(\cdot)$, as defined in Eq. (6). For all $\theta \in \mathbb{R}^d$,

$$\sup_{\psi \in [0, 1]^{|V|}} |\tilde{F}(\psi, \theta)| < \frac{1}{|V|}. \quad (10)$$

As discussed in Sec. 3, scaling $F_{\theta}(S, V)$ by a positive scalar amounts to setting the temperature of a Boltzmann distribution. Moreover, neural networks are often Lipschitz-regularized for bounded inputs and weights (Szegedy et al. 2014; Virmaux and Scaman 2018; Gouk et al. 2021). Therefore, for any such Lipschitz neural network, we can satisfy Asm. 4.1 by appropriately setting the temperature parameter of the EBM in Eq. (2). Most importantly, satisfying this condition guarantees convergence:

Theorem 4.2. Assume a set function $F_{\theta} : 2^V \rightarrow \mathbb{R}$ satisfies Asm. 4.1. Then, the fixed-point given in Eq. (7) has a unique solution $\psi^* \in [0, 1]^{|V|}$ where $\psi^* = \sigma(\nabla_{\psi} \tilde{F}(\psi^*, \theta))$. Moreover, starting with an arbitrary point $\psi^{(0)} \in [0, 1]^{|V|}$, ψ^* can be found via the fixed-point iterative sequence described in Eq. (8) where $\lim_{k \rightarrow \infty} \psi^{(k)} = \psi^*$.

The proof can be found in App. E of Özcan, Shi, and Ioannidis (2024) and relies on the Banach fixed-point theorem (Banach 1922). Thm. 4.2 implies that as long as $\tilde{F}(\psi, \theta)$ is bounded and this bound is inversely correlated with the size of the ground set, we can find a unique solution to Eq. (7), no matter where we start the iterations in Eq. (8).

4.2 Efficient Differentiation through Implicit Layers

Our second contribution is to disentangle gradient computation from stacking layers together, by using the implicit

function theorem (Krantz and Parks 2002). This allows us to use the recent work on deep equilibrium models (DEQs) (Bai, Kolter, and Koltun 2019; Kolter, Duvenaud, and Johnson 2020).

Define $\psi^*(\cdot)$ to be the map $\theta \mapsto \psi^*(\theta)$ induced by Eq. (7); equivalently, given θ , $\psi^*(\theta)$ is the (unique by Thm. 4.2) limit point of iterations given in Eq. (8). Observe that, by the chain rule:

$$\nabla_{\theta} \mathcal{L}(\psi^*(\theta)) = \nabla_{\psi} \mathcal{L}(\psi^*(\theta)) \cdot \partial_{\theta} \psi^*(\theta). \quad (11)$$

The term that is difficult to compute here via back-propagation, that required stacking in Ou et al. (2022), is the Jacobian $\partial_{\theta} \psi^*(\theta)$, as we do not have the map $\psi^*(\cdot)$ in a closed form. Nevertheless, we can use the implicit function theorem (see Thm. D.4 in Özcan, Shi, and Ioannidis (2024)) to compute this quantity.

Indeed, to simplify the notation for clarity, we define a function $G : [0, 1]^{|V|} \times \mathbb{R}^d \rightarrow [0, 1]^{|V|}$, where

$$G(\psi(\theta), \theta) \triangleq \sigma(\nabla_{\psi} \tilde{F}(\psi, \theta)) - \psi$$

and rewrite Eq. (7) as $G(\psi(\theta), \theta) = 0$. Using the implicit function theorem, given in App. D of Özcan, Shi, and Ioannidis (2024), we obtain

$$\underbrace{-\partial_{\psi} G(\psi^*(\theta), \theta)}_{A \in \mathbb{R}^{|V| \times |V|}} \underbrace{\partial_{\theta} \psi^*(\theta)}_{J \in \mathbb{R}^{|V| \times d}} = \underbrace{\partial_{\theta} G(\psi^*(\theta), \theta)}_{B \in \mathbb{R}^{|V| \times d}}. \quad (12)$$

This yields the following way of computing the Jacobian via implicit differentiation:

Theorem 4.3. *Computing $\partial_{\theta} \psi^*(\theta)$ is the equivalent of solving a linear system of equations, i.e., $\partial_{\theta} \psi^*(\theta) = A^{-1}B$,*

$$\begin{aligned} A &= I - \Sigma'(\nabla_{\psi} \tilde{F}(\psi, \theta)) \cdot \nabla_{\psi}^2 \tilde{F}(\psi, \theta), \text{ and} \\ B &= \Sigma'(\nabla_{\psi} \tilde{F}(\psi, \theta)) \cdot \partial_{\theta} \nabla_{\psi} \tilde{F}(\psi, \theta), \end{aligned} \quad (13)$$

where $\Sigma'(x) = \text{diag}([\sigma'(x_j)]_{j=1}^{|V|})$, and $\sigma'(x) = (1 + \exp(-x))^{-2} \cdot \exp(-x)$.

The proof is in App. F of Özcan, Shi, and Ioannidis (2024). Eq. (12) shows that the Jacobian of the fixed-point solution, $\partial_{\theta} \psi^*(\theta)$, can be expressed in terms of Jacobians of G at the solution point. This means implicit differentiation only needs the final fixed point value, whereas automatic differentiation via the approach by Ou et al. (2022) required all the iterates (see also (Kolter, Duvenaud, and Johnson 2020)). In practice, we use JAXopt (Blondel et al. 2022) for its out-of-the-box implicit differentiation support. This allows us to handle Hessian inverse computations efficiently (see App. G of Özcan, Shi, and Ioannidis (2024)).

4.3 Implicit Differentiable Mean Field Variation

Putting everything together, we propose *implicitly Differentiable Mean Field variation* (iDiffMF) algorithm. This algorithm finds the solution of the fixed-point in Eq. (7) by a root-finding method. Then, computes the gradient of the loss given in Eq. (11) by using the result of the implicit function theorem given in Thm. 4.3, and updates parameter θ in the direction of this gradient. We summarize this process in Alg. 2.

Algorithm 2: iDiffMF

Input: training dataset $\{(S_i^*, V_i)\}_{i=1}^N$, learning rate η , number of samples m
Output: parameter θ

- 1: $\theta \leftarrow$ initialize
- 2: **repeat**
- 3: sample training data point $(S^*, V) \sim \{(S_i^*, V_i)\}_{i=1}^N$
- 4: initialize the variational parameter $\psi^{(0)} \leftarrow 0.5 * \mathbf{1}$
- 5: **for** $j \leftarrow 1, \dots, |V|$ in parallel **do**
- 6: sample m subsets $S_{\ell} \sim q(S, (\psi | \psi_j \leftarrow 0))$
- 7: update variational parameter $\psi_j^* \leftarrow \sigma(\frac{1}{m} \sum_{\ell=1}^m [F_{\theta}(S_{\ell} \cup \{j\}) - F_{\theta}(S_{\ell})])$
- 8: **end for**
- 9: update parameter θ by computing Eq. (11) through Thm. 4.3 $\partial_{\theta} \psi^*(\theta) \leftarrow A^{-1}B$ (see Thm. 4.3) $\nabla_{\theta} \mathcal{L}(\psi^*, \theta) \leftarrow \nabla_{\psi} \mathcal{L}(\psi^*(\theta)) \cdot \partial_{\theta} \psi^*(\theta)$ $\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}(\psi^*, \theta)$
- 10: **until** convergence of θ

To emphasize the difference between Alg. 1 and Alg. 2, let us focus on lines 13 and 9, respectively. On Line 13 of the pseudo-code for the DiffMF algorithm, gradient of the loss corresponds to

$$\nabla_{\theta} \mathcal{L}(\psi^{(K)}) = \nabla_{\psi} \mathcal{L}(\psi^{(K)}) \cdot \partial_{\theta} \psi^{(K)},$$

where $\psi^{(K)}$ is a nested function in the form of

$$\psi^{(K)} = \sigma(\nabla_{\psi} \tilde{F}(\dots(\sigma(\nabla_{\psi} \tilde{F}(\psi^{(0)}), \theta)), \dots, \theta)).$$

Therefore, automatic differentiation has to unroll all K layers during gradient computation. On the other hand, on Line 9 of the iDiffMF algorithm, gradient of the loss is computed through Eq. (11) where $\partial_{\theta} \psi^*(\theta)$ has a closed form formula as a result of Thm. 4.3.

4.4 Complexity

Reverse mode automatic differentiation has a memory complexity that scales linearly with the number of iterations performed for finding the root of the fixed-point, i.e., it has a memory complexity of $\mathcal{O}(K)$ where K is the total number of iterations (Bai, Kolter, and Koltun 2019). On the other hand, reverse mode implicit differentiation has a constant memory complexity, $\mathcal{O}(1)$, because the differentiation is performed *analytically* as a result of using the implicit function theorem. Fig. 1 in Sec. 5 reflects the advantage of using implicit differentiation in terms of space requirements numerically.

In the forward mode, the time complexity of the iterative sequence inside DiffMF is again $\mathcal{O}(K)$ as the number of iterations is pre-selected and does not change with the rate of convergence. Inside iDiffMF, the convergence rate depends on the Lipschitz constant of the fixed-point in Eq. (7) and the size of the ground set. In particular, the number of iterations required for finding the root of

Eq. (7) is bounded by $\frac{\log(\epsilon(1-\omega)/\sqrt{|V|})}{\log \omega}$, where ϵ is the tolerance threshold and ω is the Lipschitz constant, i.e., $\|\sigma(\nabla_{\psi} \tilde{F}(\mathbf{x}, \theta)) - \sigma(\nabla_{\psi} \tilde{F}(\mathbf{y}, \theta))\|_2 \leq \omega \|\mathbf{x} - \mathbf{y}\|_2$ (see App. H of Özcan, Shi, and Ioannidis (2024) for computation steps). Thus, the root-finding routine inside `iDiffMF` has $\mathcal{O}\left(\frac{\log(\epsilon(1-\omega)/\sqrt{|V|})}{\log \omega}\right)$ time complexity.

5 Experiments

We evaluate our proposed method on five datasets including set anomaly detection, product recommendation, and compound selection tasks (see Tab. 1 and App. I of Özcan, Shi, and Ioannidis (2024) for a datasets summary and for detailed dataset descriptions). The Gaussian and Moons are synthetic datasets, while the rest are real-world datasets. We closely follow the experimental setup of Ou et al. (2022) w.r.t. competing algorithm setup, experiments, and metrics.²

5.1 Algorithms

We compare three competitor algorithms from (Ou et al. 2022) to three variants of our `iDiffMF` algorithm (Alg. 2). Additional implementation details are in App. I of Özcan, Shi, and Ioannidis (2024).

DiffMF (Ou et al. 2022): This is the differentiable mean field variational inference algorithm described in Alg. 1. As per Ou et al., we set the number of iterations to $K = 5$ for all datasets.

EquiVSet_{ind} (Ou et al. 2022): This is the equivariant variational inference algorithm proposed by Ou et al. (2022). It is a variation of the DiffMF algorithm where the parameter ψ is predicted by an additional recognition network as a function of the data. As per Ou et al. (2022), we set $K = 1$ for all datasets.

EquiVSet_{copula} (Ou et al. 2022): A correlation-aware version of the EquiVSet_{ind} algorithm where the relations among the input elements are modeled by a Gaussian copula. As per Ou et al. (2022), we set $K = 1$ for all datasets.

iDiffMF (Alg. 2): Our proposed implicit alternative to the DiffMF algorithm where we solve the fixed-point condition in Eq. (7) with a low tolerance threshold ($\epsilon = 10^{-6}$), instead of running the fixed-point iterations in Eq. (8) for only a fixed number of times. Although DNNs are bounded, the exact computation of their Lipschitz constant is, even for two-layer Multi-Layer-Perceptrons (MLP), NP-hard (Virmaux and Scaman 2018). In our implementation, we use several heuristic approaches to satisfy the condition in Asm. 4.1. First, we multiply the multilinear relaxation \tilde{F} by a constant scaling factor $2/(|V|c)$, treating c as a hyperparameter. We refer to this as `iDiffMFc`. We also consider a dynamic adaptation per batch and fixed-point iteration, normalizing the gradient of the multilinear relaxation by its norm as well as size of the ground set; we describe this heuristic in App. I.3 of Özcan, Shi, and Ioannidis (2024). We propose two variants, termed

`iDiffMF2` and `iDiffMF*`, using ℓ_2 ($\|\cdot\|_2$) and nuclear ($\|\cdot\|_*$) norms when scaling, respectively.

For all algorithms, we use permutation-invariant NN architectures as introduced by Ou et al., described in App. I.6 of Özcan, Shi, and Ioannidis (2024). We report all experiment results with the best-performing hyperparameters based on a 5-fold cross-validation. More specifically, we partition each dataset to a training set and a hold out/test set (see Tab. 1 of Özcan, Shi, and Ioannidis (2024) for split ratios). We then divide the training dataset in 5 folds. We identify the best hyperparameter combination through cross-validation across all folds. To produce standard-deviations, we then report the mean and the standard variation of the performance of the 5 models trained under the best hyperparameter combination on the test dataset.

We explore the following hyper-parameters: learning rate η , number of layers L , and different forward and backward solvers. Additional details, including ranges and optimal hyperparameter combinations, can be found in App. I.7 of Özcan, Shi, and Ioannidis (2024).

We use the PyTorch code repository provided by Ou et al. (2022) for all three competitor algorithms.³ We use the JAX+Flax framework (Bradbury et al. 2018; Frostig, Johnson, and Leary 2018; Heek et al. 2023) for its functional programming abilities for our `iDiffMF` implementations. In particular, we implement implicit differentiation using the JAXopt library (Blondel et al. 2022). It offers a modular differentiation tool that can be combined with the existing solvers and it is readily integrated in JAX. We include our code in the supplementary material and will make it public after the review process.

5.2 Metrics

Following Ou et al. (2022), we measure the performance of different algorithms by (a) using the trained neural network to predict the optimal subsets corresponding to each query on the test set, and (b) measure the mean Jaccard Coefficient (JC) score across all predictions. We describe how the trained objective $F_{\theta}(\cdot)$ is used to produce an optimal subset \hat{S}_i^* given query V_i in the test set in App. I.5 of Özcan, Shi, and Ioannidis (2024).

We also measure the running time and the GPU memory usage of the algorithms. During training, we track the amount of memory used every 5 seconds with the `nvidia-smi` command while varying the number of maximum iterations. For each number of maximum iterations, we report the minimum, maximum, and average memory usage.

5.3 Results

We report the predictive performance of our proposed `iDiffMF2` and `iDiffMF*` methods against the existing DiffMF method and its variants on Tab. 1, and `iDiffMFc` in App. I.7 of Özcan, Shi, and Ioannidis (2024). For the vast majority of the test cases, `iDiffMF` variants achieve either the best or the second-best JC score. While the next best competitor, `EquiVSetcopula`, performs the best on some

²<https://github.com/neu-spiral/LearnSetsImplicit>

³<https://github.com/SubsetSelection/EquiVSet>

	Datasets	EquiVSet _{ind}	EquiVSet _{copula}	DiffMF	iDiffMF ₂	iDiffMF _*
		Test JC	Test JC	Test JC	Test JC	Test JC
AD	CelebA	55.02 ± 0.20	56.16 ± 0.81	54.42 ± 0.70	<u>56.30 ± 0.58</u>	56.55 ± 0.49
	Gaussian	90.55 ± 0.06	90.94 ± 0.09	90.96 ± 0.05	<u>90.95 ± 0.18</u>	91.03 ± 0.09
	Moons	57.76 ± 0.11	<u>58.67 ± 0.18</u>	58.45 ± 0.15	<u>58.48 ± 0.15</u>	58.97 ± 0.04
PR (Amazon)	apparel	68.45 ± 0.96	78.19 ± 0.89	70.60 ± 1.35	76.13 ± 4.65	73.80 ± 5.71
	bath	67.51 ± 1.19	77.72 ± 1.98	71.87 ± 0.27	<u>77.68 ± 0.98</u>	76.43 ± 0.81
	bedding	66.20 ± 1.10	<u>77.26 ± 1.24</u>	67.66 ± 0.39	77.88 ± 0.80	76.94 ± 1.05
	carseats	19.99 ± 1.01	20.03 ± 0.15	20.15 ± 0.65	<u>21.94 ± 1.43</u>	22.42 ± 1.04
	diaper	74.26 ± 0.73	83.66 ± 0.69	81.74 ± 1.18	<u>82.76 ± 0.62</u>	82.07 ± 0.90
	feeding	71.46 ± 0.43	82.47 ± 0.19	77.44 ± 0.46	<u>81.93 ± 1.00</u>	81.52 ± 1.84
	furniture	17.28 ± 0.88	17.95 ± 0.80	16.84 ± 0.05	19.93 ± 2.68	<u>18.69 ± 0.93</u>
	gear	65.35 ± 0.91	77.33 ± 0.90	66.06 ± 2.86	<u>73.90 ± 10.29</u>	73.57 ± 6.74
	health	63.04 ± 0.41	72.03 ± 0.77	59.64 ± 0.81	72.55 ± 1.10	72.32 ± 1.03
	media	56.60 ± 0.56	55.73 ± 1.18	51.32 ± 1.11	<u>56.39 ± 2.68</u>	55.58 ± 1.75
	safety	21.99 ± 1.85	22.09 ± 3.30	24.66 ± 5.56	26.02 ± 1.68	<u>25.38 ± 1.88</u>
	toys	62.36 ± 1.31	69.08 ± 1.04	64.39 ± 1.64	68.53 ± 1.35	<u>68.91 ± 1.00</u>
CS	BindingDB	73.59 ± 0.75	73.57 ± 2.05	73.22 ± 1.08	<u>76.83 ± 0.50</u>	77.48 ± 1.04

Table 1: Test Jaccard Coefficient (JC) for set anomaly detection (AD), product recommendation (PR), and compound selection (CS) tasks, across all five algorithms. iDiffMF₂ and iDiffMF_{*} correspond to our algorithm with Frobenius and nuclear norm scaling. **Bold** and underline indicate the best and second-best performance results, respectively. The confidence intervals on the table come from the standard variation of the measurements between folds during cross-validation.

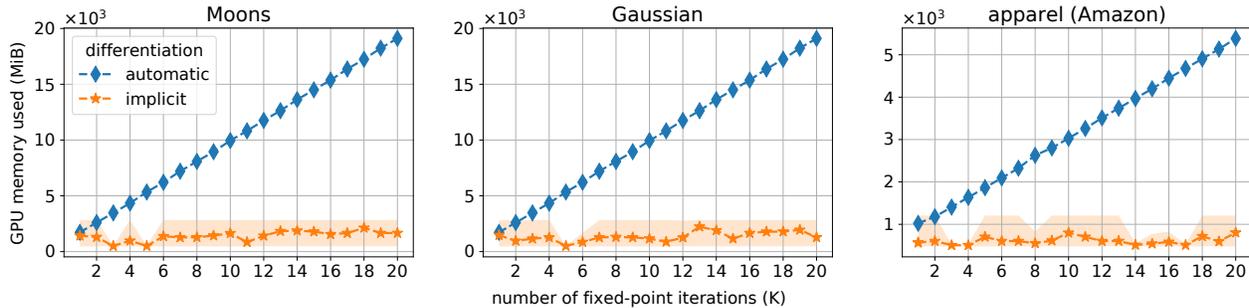


Figure 1: Effects of the choice of differentiation method on the relationship between the allocated GPU memory and the number of fixed-point iterations across different datasets. Blue lines represent automatic differentiation (DiffMF), while the orange lines represent implicit differentiation (iDiffMF). The markers denote the average memory usage. The area between the recorded minimum and maximum memory usage is shaded.

datasets, its performance is not consistent on the remaining datasets, not being even the second best. For the Amazon carseats, furniture and safety datasets, iDiffMF variants give significantly better results than EquiVSet_{copula}, even though EquiVSet_{copula} is faster. This is probably because EquiVSet_{copula} converges to a local optimum and finishes training earlier. It is also important to highlight that we evaluate iDiffMF using JAX+Flax while we use PyTorch to evaluate the baselines. Therefore, the differences in running time can also be explained with the framework differences. Even though iDiffMF executes fixed-point iterations until convergence, as opposed to $K = 1$ or $K = 5$ in remaining methods (Ou et al. 2022), the average running times are comparable across datasets (see Tab. 2 of Özcan, Shi, and Ioannidis (2024)).

In Fig. 1, we demonstrate the advantages of using implicit differentiation in terms of space complexity. As discussed in Sec. 4.4, memory requirements remain constant in an inter-

val as the number of fixed-point iterations increases during implicit differentiation. On the contrast, memory requirements increase linearly with the number of iterations during automatic differentiation.

6 Conclusion

We improve upon an existing learning set functions with an optimal subset oracle setting by characterizing the convergence condition of the fixed point iterations resulting during MLE approximation and by using implicit differentiation over automatic differentiation. Our results perform better than or comparable to the baselines for the majority of the cases without the need of an additional recognition network while requiring less memory.

Acknowledgments

We gratefully acknowledge support from the National Science Foundation (grant 1750539).

References

- Arbel, M.; and Mairal, J. 2022. Amortized implicit differentiation for stochastic bilevel optimization. In *ICLR*.
- Bach, F. 2013. Learning with Submodular Functions: A Convex Optimization Perspective. *Foundations and Trends® in machine learning*, 6(2-3): 145–373.
- Bai, S.; Kolter, J. Z.; and Koltun, V. 2019. Deep Equilibrium Models. *NeurIPS*.
- Balcan, M.-F.; and Harvey, N. J. 2018. Submodular functions: Learnability, structure, and optimization. *SICOMP*.
- Banach, S. 1922. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta mathematicae*, 3(1): 133–181.
- Bertrand, Q.; Klopfenstein, Q.; Blondel, M.; Vaiter, S.; Gramfort, A.; and Salmon, J. 2020. Implicit differentiation of lasso-type models for hyperparameter optimization. In *ICML*. PMLR.
- Bhatt, G.; Das, A.; and Bilmes, J. 2024. Deep Submodular Peripteral Network. *NeurIPS*.
- Bilmes, J.; and Bai, W. 2017. Deep submodular functions. *arXiv preprint arXiv:1701.08939*.
- Blei, D. M.; Kucukelbir, A.; and McAuliffe, J. D. 2017. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518): 859–877.
- Blondel, M.; Berthet, Q.; Cuturi, M.; Frostig, R.; Hoyer, S.; Llinares-López, F.; Pedregosa, F.; and Vert, J.-P. 2022. Efficient and Modular Implicit Differentiation. *NeurIPS*.
- Bonab, H.; Aliannejadi, M.; Vardasbi, A.; Kanoulas, E.; and Allan, J. 2021. Cross-market product recommendation. In *CIKM*.
- Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; and Zhang, Q. 2018. JAX: composable transformations of Python+NumPy programs.
- Calinescu, G.; Chekuri, C.; Pal, M.; and Vondrák, J. 2011. Maximizing a monotone submodular function subject to a matroid constraint. *SICOMP*.
- Chen, R. T.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018. Neural ordinary differential equations. *NeurIPS*.
- Davies, B. 2018. *Exploring chaos: Theory and experiment*. CRC Press.
- De, A.; and Chakrabarti, S. 2022. Neural estimation of submodular functions with applications to differentiable subset selection. *NeurIPS*.
- Djolonga, J.; and Krause, A. 2017. Differentiable learning of submodular models. *NeurIPS*.
- Dolhansky, B. W.; and Bilmes, J. A. 2016. Deep submodular functions: Definitions and learning. *NeurIPS*.
- Feldman, V.; and Kothari, P. 2014. Learning coverage functions and private release of marginals. In *COLT*. PMLR.
- Feng, Q.; Zhou, Y.; and Lan, R. 2016. Pairwise linear regression classification for image set retrieval. In *CVPR*.
- Frostig, R.; Johnson, M. J.; and Leary, C. 2018. Compiling machine learning programs via high-level tracing. *Systems for Machine Learning*, 4(9).
- Giannone, G.; and Winther, O. 2022. Scha-vae: Hierarchical context aggregation for few-shot generation. In *ICML*.
- Gionis, A.; Gunopulos, D.; and Koudas, N. 2001. Efficient and tumble similar set retrieval. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, 247–258.
- Gomez-Rodriguez, M.; Leskovec, J.; and Krause, A. 2012. Inferring networks of diffusion and influence. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(4): 1–37.
- Gouk, H.; Frank, E.; Pfahringer, B.; and Cree, M. J. 2021. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110: 393–416.
- He, X.; Xu, K.; Kempe, D.; and Liu, Y. 2016. Learning influence functions from incomplete observations. *NeurIPS*.
- Heek, J.; Levskaya, A.; Oliver, A.; Ritter, M.; Rondepierre, B.; Steiner, A.; and van Zee, M. 2023. Flax: A neural network library and ecosystem for JAX.
- Hinton, G.; Osindero, S.; Welling, M.; and Teh, Y.-W. 2006. Unsupervised discovery of nonlinear structure using contrastive backpropagation. *Cognitive science*, 30(4): 725–731.
- Huang, Z.; Bai, S.; and Kolter, J. Z. 2021. (Implicit)²: Implicit Layers for Implicit Representations. *NeurIPS*.
- Karalias, N.; Robinson, J.; Loukas, A.; and Jegelka, S. 2022. Neural set function extensions: Learning with discrete functions in high dimensions. *NeurIPS*.
- Kim, J.; Yoo, J.; Lee, J.; and Hong, S. 2021. SetVAE: Learning Hierarchical Composition for Generative Modeling of Set-Structured Data. In *CVPR*.
- Kimura, M.; Shimizu, R.; Hirakawa, Y.; Goto, R.; and Saito, Y. 2024. On permutation-invariant neural networks. *arXiv preprint arXiv:2403.17410*.
- Kingma, D. P.; and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.
- Kirsch, A.; Farquhar, S.; Atighehchian, P.; Jesson, A.; Branchaud-Charron, F.; and Gal, Y. 2023. Stochastic Batch Acquisition: A Simple Baseline for Deep Active Learning. *TMLR*.
- Kolter, Z.; Duvenaud, D.; and Johnson, M. 2020. Deep Implicit Layers - Neural ODEs, Deep Equilibrium Models, and Beyond. *NeurIPS*.
- Kothawade, S.; Girdhar, J.; Lavania, C.; and Iyer, R. 2020. Deep submodular networks for extractive data summarization. *arXiv preprint arXiv:2010.08593*.
- Krantz, S. G.; and Parks, H. R. 2002. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media.
- LeCun, Y.; Chopra, S.; Hadsell, R.; Ranzato, M.; and Huang, F. J. 2006. A tutorial on energy-based learning. *Predicting structured data*, 1(0).
- Lee, J.; Lee, Y.; Kim, J.; Kosiorek, A.; Choi, S.; and Teh, Y. W. 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In *ICML*.
- Li, X.; Wong, T.-K. L.; Chen, R. T.; and Duvenaud, D. 2020. Scalable gradients for stochastic differential equations. In *AISTATS*.

- Lorraine, J.; Vicol, P.; and Duvenaud, D. 2020. Optimizing millions of hyperparameters by implicit differentiation. In *AISTATS*.
- Mašková, M.; Zorek, M.; Pevný, T.; and Šmídl, V. 2024. Deep anomaly detection on set data: Survey and comparison. *Pattern Recognition*, 110381.
- Mnih, A.; and Hinton, G. 2005. Learning nonlinear constraints with contrastive backpropagation. In *IJCNN*. IEEE.
- Murphy, K. P. 2012. *Machine learning: a probabilistic perspective*. MIT press.
- Nelsen, R. B. 2006. *An Introduction to Copulas*. New York, NY, USA: Springer, second edition.
- Nikishin, E.; Abachi, R.; Agarwal, R.; and Bacon, P.-L. 2022. Control-oriented model-based reinforcement learning with implicit differentiation. In *AAAI*.
- Ning, X.; Walters, M.; and Karypis, G. 2011. Improved machine learning models for predicting selective compounds. In *Proceedings of the 2nd ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, 106–115.
- Ou, Z.; Xu, T.; Su, Q.; Li, Y.; Zhao, P.; and Bian, Y. 2022. Learning Neural Set Functions Under the Optimal Subset Oracle. *NeurIPS*.
- Özcan, G.; Shi, C.; and Ioannidis, S. 2024. Learning Set Functions with Implicit Differentiation. *arXiv preprint arXiv:2412.11239*.
- Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. In *NeurIPS Autodiff Workshop*.
- Rechnitzer, A. 2003. Fixed Points - Summary [Lecture Notes]. *Dynamical Systems and Chaos* — 620341.
- Saito, Y.; Nakamura, T.; Hachiya, H.; and Fukumizu, K. 2020. Exchangeable deep neural networks for set-to-set matching and learning. In *ECCV*.
- Schafer, J. B.; Konstan, J.; and Riedl, J. 1999. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, 158–166.
- Sittoni, P.; and Tudisco, F. 2024. Subhomogeneous Deep Equilibrium Models. In *ICML*.
- Sklar, A. 1973. Random variables, joint distribution functions, and copulas. *Kybernetika*, 9(6): 449–460.
- Soelch, M.; Akhundov, A.; van der Smagt, P.; and Bayer, J. 2019. On Deep Set Learning and the Choice of Aggregations. In *Artificial Neural Networks and Machine Learning – ICANN 2019: Theoretical Neural Computation: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings, Part I*, 444–457. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-030-30486-7.
- Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *ICLR*.
- Tschiatschek, S.; Sahin, A.; and Krause, A. 2018. Differentiable submodular maximization. In *IJCAI*.
- Virmaux, A.; and Scaman, K. 2018. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *NeurIPS*.
- Wagstaff, E.; Fuchs, F.; Engelcke, M.; Posner, I.; and Osborne, M. A. 2019. On the limitations of representing functions on sets. In *ICML*.
- Wendler, C.; Amrollahi, A.; Seifert, B.; Krause, A.; and Püschel, M. 2021. Learning set functions that are sparse in non-orthogonal Fourier bases. In *AAAI*.
- Wendler, C.; Püschel, M.; and Alistarh, D. 2019. Powerset convolutional neural networks. *NeurIPS*.
- Winston, E.; and Kolter, J. Z. 2020. Monotone operator equilibrium networks. *NeurIPS*.
- Xu, M.; Molloy, T. L.; and Gould, S. 2024. Revisiting implicit differentiation for learning problems in optimal control. *NeurIPS*.
- Zaheer, M.; Kottur, S.; Ravanbakhsh, S.; Poczos, B.; Salakhutdinov, R. R.; and Smola, A. J. 2017. Deep sets. *NeurIPS*.
- Zhang, L.; Tozzo, V.; Higgins, J.; and Ranganath, R. 2022a. Set norm and equivariant skip connections: Putting the deep in deep sets. In *ICML*. PMLR.
- Zhang, Y.; Hare, J.; and Prugel-Bennett, A. 2019. Deep set prediction networks. *NeurIPS*.
- Zhang, Y.; Zhang, D. W.; Lacoste-Julien, S.; Burghouts, G. J.; and Snoek, C. G. 2022b. Multiset-Equivariant Set Prediction with Approximate Implicit Differentiation. In *ICLR*.
- Zhao, H.; Jiang, L.; Fu, C.-W.; and Jia, J. 2019. PointWeb: Enhancing Local Neighborhood Features for Point Cloud Processing. In *CVPR*.
- Zucchet, N.; and Sacramento, J. 2022. Beyond backpropagation: bilevel optimization through implicit differentiation and equilibrium propagation. *Neural Computation*, 34(12): 2309–2346.