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; Gionis, 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, Tschiatschek, 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).

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), reinforcement 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 $F_{\boldsymbol{\theta}} : 2^{\Omega} \times 2^{\Omega} \to \mathbb{R}$, parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$, modeling the utility of the oracle, so that

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

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 $F_{\boldsymbol{\theta}}(S, V)$ 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 $F_{\boldsymbol{\theta}}$, 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:

$$p_{\boldsymbol{\theta}}(S \mid V) = \frac{\exp\left(F_{\boldsymbol{\theta}}(S,\, V)\right)}{\sum_{S' \subseteq V} \exp\left(F_{\boldsymbol{\theta}}(S',\, V)\right)}. \tag{2}$$

This is equivalent to Eq. (1), presuming that the utility $F_{\boldsymbol{\theta}(\cdot)}$ 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:

$$\arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left[\log p_{\boldsymbol{\theta}}(S_i^* \mid V_i)\right]. \tag{3}$$

Notice that multiplying $F_{\boldsymbol{\theta}}$ 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 $c \cdot F_{\boldsymbol{\theta}}(\cdot)$ 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 ($c \to \infty$) in Eq. (1) and the uniform distribution ($c \to 0$).[1]

3.2 Variational Approximation of Energy-Based Models

Learning $\boldsymbol{\theta}$ 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 $p_{\boldsymbol{\theta}}$ given by $q(S, V, \boldsymbol{\psi}) = \prod_{j \in S} \psi_j \prod_{j \in V \setminus S}(1 - \psi_j)$, parameterized by the probability vector $\boldsymbol{\psi}$: 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:

$$\begin{aligned}
\text{Min.} \quad & \mathcal{L}(\{\boldsymbol{\psi}_i^*\}) = \mathbb{E}_{\mathbb{P}(V,\, S)}[-\log q(S,\, V\,, \boldsymbol{\psi}^*)] \approx \\
& \frac{1}{N} \sum_{i=1}^{N} \left(- \sum_{j \in S_i^*} \log \psi_{ij}^* - \sum_{j \in V_i \setminus S_i^*} \log\left(1 - \psi_{ij}^*\right) \right), \\
\text{subj. to} \quad & \boldsymbol{\psi}_i^* = \arg\min_{\boldsymbol{\psi}} \mathbb{KL}(q(S_i,\, V_i,\, \boldsymbol{\psi}) \,||\, p_{\boldsymbol{\theta}}(S_i \mid V_i)), \\
& \text{for all } i \in \{1, \ldots, n\},
\end{aligned} \tag{4}$$

where $\boldsymbol{\psi}_i^* \in [0,1]^{|V|}$ is the probability vector of elements in V_i being included in S_i, $\mathbb{KL}(\cdot||\cdot)$ is the Kullback-Leibler divergence, and $p_{\boldsymbol{\theta}}(\cdot)$ 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 $\boldsymbol{\psi}^*$, 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 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):

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

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

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

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

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

where the function $\boldsymbol{\sigma} : \mathbb{R}^{|V|} \to \mathbb{R}^{|V|}$ is defined as $\boldsymbol{\sigma}(\mathbf{x}) = [\sigma(x_j)]_{j=1}^{|V|}$ and $\sigma : \mathbb{R} \to \mathbb{R}$ 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.

Observing that the stationary condition in Eq. (7) is a fixed point equation, Ou et al. (2022) propose solving it via the

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 $\boldsymbol{\theta}$

$\boldsymbol{\theta} \leftarrow$ initialize
repeat
  sample training data point $(S^*, V) \sim \{(S_i^*, V_i)\}_{i=1}^N$
  initialize the variational parameter $\boldsymbol{\psi}^{(0)} \leftarrow 0.5 * \mathbf{1}$
  repeat
    for $k \leftarrow 1, \ldots, K$ **do**
      for $j \leftarrow 1, \ldots, |V|$ in parallel **do**
        sample m subsets $S_\ell \sim q(S, (\boldsymbol{\psi}^{(k-1)} | \psi_j^{(k-1)} \leftarrow 0))$
        update variational parameter $\psi_j^{(k)} \leftarrow \sigma\left(\frac{1}{m} \sum_{\ell=1}^{m} [F_{\boldsymbol{\theta}}(S_\ell \cup \{j\}) - F_{\boldsymbol{\theta}}(S_\ell)]\right)$
      end for
    end for
  until convergence of $\boldsymbol{\psi}$
  update parameter $\boldsymbol{\theta}$ by unfolding the derivatives of the $K-$layer meta-network resulting from the fixed-point equations given in Eq. (8) during SGD
  $\partial_{\boldsymbol{\theta}} \boldsymbol{\psi}^{(K)}(\boldsymbol{\theta}) \leftarrow$
  $\partial_{\boldsymbol{\theta}} \underbrace{\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}^{(K-1)}} \tilde{F}(\ldots(\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}^{(0)}} \tilde{F}(\boldsymbol{\psi}^{(0)}))\ldots))}_{K \text{nested functions}}$
  $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\psi}^{(K)}, \boldsymbol{\theta}) \leftarrow \nabla_{\boldsymbol{\psi}^{(K)}} \mathcal{L}(\boldsymbol{\psi}^{(K)}(\boldsymbol{\theta})) \cdot \partial_{\boldsymbol{\theta}} \boldsymbol{\psi}^{(K)}(\boldsymbol{\theta})$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\psi}^{(K)}, \boldsymbol{\theta}\right)$
until convergence of $\boldsymbol{\theta}$

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

following fixed-point iterations. Given $\boldsymbol{\theta} \in \mathbb{R}^d$,

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

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 $\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})$ via Monte Carlo sampling (see App. C.2 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 $\boldsymbol{\theta}$, 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 $F_{\boldsymbol{\theta}}(\cdot)$ (one per sample). Subsequently, this extended network is entered in the loss given in Eq. (4), which is minimized w.r.t. $\boldsymbol{\theta}$ via SGD.

They also introduce two variants of this algorithm, regressing also $\boldsymbol{\psi}^{(0)}$ as a function of the item features via an extra recognition network, assuming the latter are independent (terming inference in this setting as $\texttt{EquiVSet}_{\text{ind}}$) or correlated by a Gaussian copula (Sklar 1973; Nelsen 2006) (termed $\texttt{EquiVSet}_{\text{copula}}$). 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, $\boldsymbol{\psi}^{(K)}$, is heavily dependent on the selection of the starting point, $\boldsymbol{\psi}^{(0)}$. Moreover, as these iterations correspond to stacking up layers, each containing multiple copies of $F_{\boldsymbol{\theta}}(\cdot)$ due to sampling, backpropagation 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:

$$\begin{aligned} \underset{\{\boldsymbol{\psi}_i^*\}, \boldsymbol{\theta}}{\text{Min.}} \quad & \mathcal{L}(\{\boldsymbol{\psi}_i^*\}) \approx \\ & \frac{1}{N} \sum_{i=1}^{N} \left(- \sum_{j \in S_i^*} \log \psi_{ij} - \sum_{j \in V_i \setminus S_i^*} \log (1 - \psi_{ij}) \right), \\ \text{subj. to} \quad & \boldsymbol{\psi}_i^* = \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}_i^*, \boldsymbol{\theta})), \text{ for all } i \in \{i, \ldots, n\}. \end{aligned} \tag{9}$$

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; Rechnitzer 2003). We characterize the condition under which the convergence is guaranteed in the following assumption.

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

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

As discussed in Sec. 3, scaling $F_{\boldsymbol{\theta}}(S, V)$ 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 $F_{\boldsymbol{\theta}} : 2^V \to \mathbb{R}$ satisfies Asm. 4.1. Then, the fixed-point given in Eq.* (7) *has a unique solution $\boldsymbol{\psi}^* \in [0,1]^{|V|}$ where $\boldsymbol{\psi}^* = \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}^*, \boldsymbol{\theta}))$. Moreover, starting with an arbitrary point $\boldsymbol{\psi}^{(0)} \in [0,1]^{|V|}$, $\boldsymbol{\psi}^*$ can be found via the fixed-point iterative sequence described in Eq.* (8) *where $\lim_{k \to \infty} \boldsymbol{\psi}^{(k)} = \boldsymbol{\psi}^*$.*

The proof can be found in App. E and relies on the Banach fixed-point theorem (Banach 1922). Thm. 4.2 implies that as long as $\tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})$ 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 $\boldsymbol{\psi}^*(\cdot)$ to be the map $\boldsymbol{\theta} \mapsto \boldsymbol{\psi}^*(\boldsymbol{\theta})$ induced by Eq. (7); equivalently, given $\boldsymbol{\theta}$, $\boldsymbol{\psi}^*(\boldsymbol{\theta})$ is the (unique by Thm. 4.2) limit point of iterations given in Eq. (8). Observe that, by the chain rule:

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

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

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

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

and rewrite Eq. (7) as $G(\boldsymbol{\psi}(\boldsymbol{\theta}), \boldsymbol{\theta}) = 0$. Using the implicit function theorem, given in App. D , we obtain

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

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

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

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

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

The proof is in App. F. Eq. (12) shows that the Jacobian of the fixed-point solution, $\partial_{\boldsymbol{\theta}} \boldsymbol{\psi}^*(\boldsymbol{\theta})$, 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).

4.3 Implicit Differentiable Mean Field Variation

Putting everything together, we propose ***i**mplicitly **Diff**erentiable **M**ean **F**ield 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 $\boldsymbol{\theta}$ in the direction of this gradient. We summarize this process in Alg. 2.

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

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

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

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

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 $\partial_{\boldsymbol{\theta}} \boldsymbol{\psi}^*(\boldsymbol{\theta})$ has a closed form formulation as a result of Thm. 4.3.

Algorithm 2: `iDiffMF`

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

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

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., $\|\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{x},\boldsymbol{\theta})) - \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{y},\boldsymbol{\theta}))\|_2 \leq \omega\|\boldsymbol{x}-\boldsymbol{y}\|_2$ (see App. H 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 for a datasets summary and App. I 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.

`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.

$\texttt{EquiVSet}_{\text{ind}}$ **(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 $\boldsymbol{\psi}$ 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.

$\texttt{EquiVSet}_{\text{copula}}$ **(Ou et al. 2022):** A correlation-aware version of the $\texttt{EquiVSet}_{\text{ind}}$ 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 $\texttt{iDiffMF}_c$. 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. We propose two variants, termed $\texttt{iDiffMF}_2$ and $\texttt{iDiffMF}_*$, using ℓ_2 ($\|\cdot\|_2$) and nuclear ($\|\cdot\|_*$) norms when scaling, respectively.

[2] https://github.com/neu-spiral/LearnSetsImplicit

	Dataset	$\lvert\Omega\rvert$	$\lvert\mathcal{D}\rvert$	$\lvert V\rvert$	avg($\lvert S^*\rvert$)	min($\lvert S^*\rvert$)	max($\lvert S^*\rvert$)	d_f	Split ratio
	CelebA	202,599	10000	8	2.5	2	3	128	11 : 1
	Gaussian	100	1000	100	10	10	10	2	2 : 1
	Moons	100	1000	100	100	10	10	2	2 : 1
Amazon	apparel	100	4,675	30	4.52	3	19	768	2 : 1
Amazon	bath	100	3,195	30	3.80	3	11	768	2 : 1
Amazon	bedding	100	4,524	30	3.87	3	12	768	2 : 1
Amazon	carseats	34	483	30	3.26	3	6	768	2 : 1
Amazon	diaper	100	6,108	30	4.14	3	15	768	2 : 1
Amazon	feeding	100	8,202	30	4.62	3	2	768	2 : 1
Amazon	furniture	32	280	30	3.18	3	6	768	2 : 1
Amazon	gear	100	4,277	30	3.8	3	10	768	2 : 1
Amazon	health	62	2,995	30	3.69	3	9	768	2 : 1
Amazon	media	58	1,485	30	4.52	3	19	768	2 : 1
Amazon	safety	36	267	30	3.16	3	5	768	2 : 1
Amazon	toys	62	2,421	30	4.09	3	14	768	2 : 1
	BindingDB	52,273	1,200	300	15	15	15	512	11 : 1

Table 1: Summary of the datasets. Ω denotes the universe with all possible options. $\mathcal{D}$ is the dataset with (S^*, V) pairs. $V \subseteq \Omega$ is the ground set of options and S^* is the optimal subset of V. d_f is the size of the feature vector for each item in Ω. The optimal subset, ground set (S^*, V) pair selection/generation is instance specific and we describe these processes in the App. I. Optimal subsets are subject to cardinality constraints.

For all algorithms, we use permutation-invariant NN architectures as introduced by Ou et al., described in App. I.6. 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 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.

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 $F_{\boldsymbol{\theta}}(\cdot)$ is used to produce an optimal subset $\hat{S}_i^*$ given query V_i in the test set in App. I.5.

We also measure the running time and the GPU memory usage of the algorithms. During training, we track the amount of

[3] https://github.com/SubsetSelection/EquiVSet

	Datasets	EquiVSet$_{ind}$		EquiVSet$_{copula}$		DiffMF		iDiffMF$_2$		iDiffMF$_*$	
		Test JC	**Time** (s)	**Test JC**	**Time** (s)	**Test JC**	**Time** (s)	**Test JC**	**Time** (s)	**Test JC**	**Time** (s)
AD	CelebA	55.02 ± 0.20	1151.17 ± 698.13	56.16 ± 0.81	1195.47 ± 731.84	54.42 ± 0.70	1299.13 ± 984.20	56.30 ± 0.58	1880.86 ± 266.84	$\mathbf{56.55 \pm 0.49}$	1827.76 ± 472.78
	Gaussian	90.55 ± 0.06	30.68 ± 3.86	90.94 ± 0.09	39.11 ± 6.09	90.96 ± 0.05	85.75 ± 35.82	90.95 ± 0.18	39.41 ± 3.64	$\mathbf{91.03 \pm 0.09}$	46.05 ± 3.44
	Moons	57.76 ± 0.11	66.99 ± 4.43	58.67 ± 0.18	62.03 ± 6.82	58.45 ± 0.15	58.24 ± 3.01	58.48 ± 0.15	70.26 ± 13.96	$\mathbf{58.97 \pm 0.04}$	53.80 ± 10.79
PR (Amazon)	apparel	68.45 ± 0.96	38.32 ± 6.63	$\mathbf{78.19 \pm 0.89}$	77.14 ± 14.37	70.60 ± 1.35	63.06 ± 16.12	76.13 ± 4.65	110.74 ± 42.12	73.80 ± 5.71	98.43 ± 35.07
	bath	67.51 ± 1.19	34.01 ± 5.89	$\mathbf{77.72 \pm 1.98}$	53.29 ± 6.68	71.87 ± 0.27	61.84 ± 12.73	77.68 ± 0.98	70.90 ± 12.26	76.43 ± 0.81	80.12 ± 14.43
	bedding	66.20 ± 1.10	40.99 ± 3.59	77.26 ± 1.24	67.13 ± 12.78	67.66 ± 0.39	72.69 ± 7.73	$\mathbf{77.88 \pm 0.80}$	103.64 ± 17.76	76.94 ± 1.05	88.56 ± 19.34
	carseats	19.99 ± 1.01	12.38 ± 4.19	20.03 ± 0.15	12.19 ± 2.71	20.15 ± 0.65	10.53 ± 5.01	21.94 ± 1.43	40.98 ± 7.77	$\mathbf{22.42 \pm 1.04}$	45.00 ± 8.46
	diaper	74.26 ± 0.73	60.96 ± 17.79	$\mathbf{83.66 \pm 0.69}$	193.55 ± 80.28	81.74 ± 1.18	95.22 ± 10.54	82.76 ± 0.62	127.09 ± 23.58	82.07 ± 0.90	144.33 ± 30.65
	feeding	71.46 ± 0.43	68.43 ± 26.08	$\mathbf{82.47 \pm 0.19}$	95.18 ± 21.75	77.44 ± 0.46	93.27 ± 18.81	81.93 ± 1.00	145.46 ± 50.13	81.52 ± 1.84	179.65 ± 18.05
	furniture	17.28 ± 0.88	10.98 ± 2.44	17.95 ± 0.80	10.03 ± 3.23	16.84 ± 0.05	9.31 ± 1.79	$\mathbf{19.93 \pm 2.68}$	34.31 ± 6.03	18.69 ± 0.93	34.30 ± 5.96
	gear	65.35 ± 0.91	40.89 ± 3.19	$\mathbf{77.33 \pm 0.90}$	69.44 ± 10.22	66.06 ± 2.86	60.95 ± 10.38	73.90 ± 10.29	92.30 ± 45.44	73.57 ± 6.74	132.10 ± 30.14
	health	63.04 ± 0.41	33.51 ± 5.22	72.03 ± 0.77	60.18 ± 6.31	59.64 ± 0.81	51.66 ± 2.54	$\mathbf{72.55 \pm 1.10}$	78.65 ± 13.01	72.32 ± 1.03	88.71 ± 21.76
	media	$\mathbf{56.60 \pm 0.56}$	37.45 ± 11.06	55.73 ± 1.18	45.02 ± 4.95	51.32 ± 1.11	40.69 ± 4.65	56.39 ± 2.68	65.15 ± 18.41	55.58 ± 1.75	67.83 ± 21.18
	safety	21.99 ± 1.85	10.39 ± 1.87	22.09 ± 3.30	13.14 ± 3.13	24.66 ± 5.56	8.59 ± 1.31	$\mathbf{26.02 \pm 1.68}$	47.66 ± 8.62	25.38 ± 1.88	44.63 ± 6.45
	toys	62.36 ± 1.31	34.06 ± 6.69	$\mathbf{69.08 \pm 1.04}$	47.81 ± 9.46	64.39 ± 1.64	43.96 ± 6.89	68.53 ± 1.35	68.34 ± 17.82	68.91 ± 1.00	80.30 ± 18.76
CS	BindingDB	73.59 ± 0.75	9934.30 ± 2591.36	73.57 ± 2.05	13983.93 ± 4458.52	73.22 ± 1.08	21472.44 ± 3239.73	76.83 ± 0.50	10887.64 ± 1709.79	$\mathbf{77.48 \pm 1.04}$	10612.98 ± 946.64

Table 2: Test Jaccard Coefficient (JC) and training time for set anomaly detection (AD), product recommendation (PR), and compound selection (CS) tasks, across all five algorithms. iDiffMF$_2$ 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.

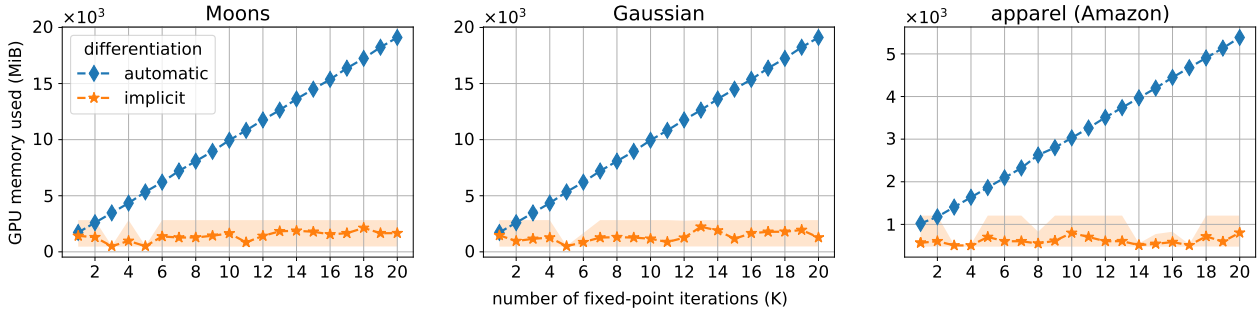

Figure 1: Effects of the choice of differentiation method on the relationship between the allocated GPU memory and the number of fixed-point iterations across different datasets. Blue lines represent automatic differentiation (DiffMF), while the orange lines represent implicit differentiation (iDiffMF).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

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 iDiffMF$_2$ and iDiffMF$_*$ methods against the existing DiffMF method and its variants on Tab. 2, and iDiffMF$_c$ in App. I.7. 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, EquiVSet$_{copula}$, performs the best on some 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.

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 interval 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

Anderson, D. G. 1965. Iterative Procedures for Nonlinear Integral Equations. *J. ACM*, 12(4): 547–560.

Apostol, T. 1974. *Mathematical Analysis*. Addison-Wesley series in mathematics. Addison-Wesley. ISBN 9780201002881.

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*.

Bhhatarai, B.; Walters, W. P.; Hop, C. E.; Lanza, G.; and Ekins, S. 2019. Opportunities and challenges using artificial intelligence in ADME/Tox. *Nature materials*, 18(5): 418–422.

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.

Burke, J. V. 2014. Nonlinear optimization. *Lecture Notes, Math*, 408: 80.

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*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *North American Chapter of the Association for Computational Linguistics*.

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*.

Gillenwater, J. A.; Kulesza, A.; Fox, E.; and Taskar, B. 2014. Expectation-maximization for learning determinantal point processes. *NeurIPS*.

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.

Hestenes, M. R.; and Stiefel, E. 1952. *Methods of conjugate gradients for solving linear systems*. NBS Washington, DC.

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. $(\text{Implicit})^2$: 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 Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

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 Equilibirum 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*.

Liu, Z.; Li, Y.; Han, L.; Li, J.; Liu, J.; Zhao, Z.; Nie, W.; Liu, Y.; and Wang, R. 2015a. PDB-wide collection of binding data: current status of the PDBbind database. *Bioinformatics*, 31(3): 405–412.

Liu, Z.; Luo, P.; Wang, X.; and Tang, X. 2015b. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

Lorraine, J.; Vicol, P.; and Duvenaud, D. 2020. Optimizing millions of hyperparameters by implicit differentiation. In *AISTATS*.

Mašková, M.; Zorek, M.; Pevnỳ, 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*.

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*.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Rechnitzer, A. 2003. Fixed Points - Summary [Lecture Notes]. Dynamical Systems and Chaos — 620341.

Rudin, W. 1976. *Principles of Mathematical Analysis*. International series in pure and applied mathematics. McGraw-Hill. ISBN 9780070856134.

Saad, Y.; and Schultz, M. H. 1986. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3): 856–869.

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.

Singh, S.; Gupta, H.; Sharma, P.; and Sahi, S. 2024. Advances in Artificial Intelligence (AI)-assisted approaches in drug screening. *Artificial Intelligence Chemistry*, 2(1): 100039.

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.

Appendix

A Permutation Invariance

In this section, we formally define permutation invariant functions and state the relationship between sum-decomposable and permutation-invariant functions following the works of Zaheer et al. (2017) and Wagstaff et al. (2019). We use these definitions to explain how we also enforce the property of permutation invariance in this work.

Definition A.1. (Zaheer et al. 2017, Property 1) A function $f(\mathbf{x})$ is *permutation-invariant* if $f(x_1, \ldots, x_M) = f(x_{\pi(1)}, \ldots, x_{\pi(M)})$ for all π permutations.

Definition A.2. (Wagstaff et al. 2019, Definition 2.2) A set function f is *sum-decomposable* if there are functions ρ and ϕ such that

$$f(S) = \rho\left(\sum_{s \in S} \phi(s)\right).$$

Theorem A.3. *(Zaheer et al. 2017, Theorem 2) (Wagstaff et al. 2019, Theorem 2.8) Let $f : 2^S \to \mathbb{R}$ where S is countable. Then, f is permutation-invariant if and only if it is sum-decomposable via $\mathbb{R}$.*

In Ou et al. (2022), ϕ is a dataset specific initial layer that takes set elements, s, as inputs and transforms them into some representation $\phi(s)$. These representations are added up and go through ρ, a fully connected feed forward neural network. We use the same architectures for ϕ and ρ (see App. I.6). As a result, our model satisfies the permutation invariance property.

B Proof of Equation (5)

Proof. Starting from the definition of the KL divergence, we get:

$$\begin{aligned}
\mathbb{KL}(q(S, \boldsymbol{\psi}) || p_{\boldsymbol{\theta}}(S)) &= \sum_{S \subseteq V} q(S, \boldsymbol{\psi}) \log \frac{q(S, \boldsymbol{\psi})}{p_{\boldsymbol{\theta}}(S)} \\
&= \sum_{S \subseteq V} q(S, \boldsymbol{\psi}) \left(\log q(S, \boldsymbol{\psi}) - \log p_{\boldsymbol{\theta}}(S)\right) \\
&= \sum_{S \subseteq V} \left(q(S, \boldsymbol{\psi}) \log q(S, \boldsymbol{\psi}) - q(S, \boldsymbol{\psi}) \log p_{\boldsymbol{\theta}}(S)\right) \\
&= \sum_{S \subseteq V} q(S, \boldsymbol{\psi}) \log q(S, \boldsymbol{\psi}) - \sum_{S \subseteq V} q(S, \boldsymbol{\psi}) \log p_{\boldsymbol{\theta}}(S) \\
&= -\mathbb{H}(q(S, \boldsymbol{\psi})) - \mathbb{E}_{q(S, \boldsymbol{\psi})}[\log p_{\boldsymbol{\theta}}(S)].
\end{aligned}$$

Observe that, by Eq. (2):

$$\mathbb{E}_{q(S, \boldsymbol{\psi})}[\log p_{\boldsymbol{\theta}}(S)] = \mathbb{E}_{q(S, \boldsymbol{\psi})}[F_{\boldsymbol{\theta}}(S)] - Z = \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}) - Z,$$

where $Z \equiv \sum_{S' \subseteq V} \exp\left(F_{\boldsymbol{\theta}}(S', V)\right)$ does not depend on $\boldsymbol{\psi}$ and thus can be dropped, and $\tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})$ is the multilinear relaxation. Therefore, minimizing the KL divergence w.r.t. $\boldsymbol{\psi}$ is equivalent to maximizing $\tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}) + \mathbb{H}(q(S, \boldsymbol{\psi}))$. In summary,

$$\min_{\boldsymbol{\psi}} \mathbb{KL}(q(S, \boldsymbol{\psi}) || p_{\boldsymbol{\theta}}(S)) \iff \max_{\boldsymbol{\psi}} \underbrace{\tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}) + \mathbb{H}(q(S, \boldsymbol{\psi}))}_{\text{ELBO}}. \tag{14}$$

□

C Derivations for Sec. 3

C.1 Derivation of the Fixed-Point

Rewriting the ELBO by plugging in the definition of entropy,

$$\tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}) + \mathbb{H}(q(S, \boldsymbol{\psi})) = \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}) - \sum_{i=1}^{|V|} \left[\psi_i \log \psi_i + (1 - \psi_i) \log(1 - \psi_i)\right]. \tag{15}$$

Taking the partial derivative of this expression with respect to the i^{th} coordinate and setting it to zero, yields

$$\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_i} - \log \frac{\psi_i}{1-\psi_i} = 0,$$
$$\exp \frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_i} = \frac{\psi_i}{1-\psi_i},$$
$$\exp \frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_i} - \psi_i \exp\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_i}\right) = \psi_i,$$
$$\exp\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_i}\right) = \psi_i \left(1 + \exp\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_i}\right)\right),$$
$$\psi_i = \frac{\exp\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_i}\right)}{1 + \exp\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_i}\right)} = \frac{1}{1 + \exp\left(-\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_i}\right)} = \sigma\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_i}\right),$$

where $\sigma(x) = (1 + \exp(-x))^{-1}$ is the sigmoid function.

C.2 Gradient Computation via Sampling

Lemma C.1. *Given a set function $F : 2^V \to \mathbb{R}$ and a vector of probabilities $\boldsymbol{\psi} \in [0,1]^{|V|}$ where $\psi_i = \mathbb{P}[i \in S]$, the gradient of the multilinear relaxation of the set function $F(S)$ is*

$$\frac{\partial \tilde{F}(\boldsymbol{\psi})}{\partial \psi_i} = \mathbb{E}_{S \sim \boldsymbol{\psi} | \psi_i \leftarrow 0}[F(S+i) - F(S)].$$

Proof.

$$\begin{aligned}
\frac{\partial \tilde{F}(\boldsymbol{\psi})}{\partial \psi_i} &= \frac{\partial}{\partial \psi_i} \sum_{S \subseteq V} F(S) \prod_{i \in S} \psi_i \prod_{i \notin S} (1 - \psi_i), \\
&= \mathbb{E}_{S \sim \boldsymbol{\psi} | \psi_i \leftarrow 1}[F(S)] - \mathbb{E}_{S \sim \boldsymbol{\psi} | \psi_i \leftarrow 0}[F(S)], \\
&= \sum_{S \subseteq V,\, i \in S} F(S) \prod_{j \in S \setminus \{i\}} \psi_j \prod_{j \notin S} (1 - \psi_j) - \sum_{S \subseteq V \setminus \{i\}} F(S) \prod_{j \in S} \psi_j \prod_{j \notin S,\, j \neq i} (1 - \psi_j), \\
&= \sum_{S \subseteq V \setminus \{i\}} [F(S+i) - F(S)] \prod_{j \in S} \psi_j \prod_{j \notin S} (1 - \psi_j), \\
&= \mathbb{E}_{S \sim \boldsymbol{\psi} | \psi_i \leftarrow 0}[F(S+i) - F(S)].
\end{aligned}$$

□

In the proof above, the fourth line holds because on the first summation of the right-hand side, i is included in all instances of S. This is equivalent to iterating over all S that exclude i and then adding i to these sets. In the expectation $\mathbb{E}_{S \sim \boldsymbol{\psi} | \psi_i \leftarrow 0}[F(S+i) - F(S)]$, we are sampling a set S based on $\boldsymbol{\psi}$ where $\psi_i = 0$. Then, we are adding the element i to S.

Corollary C.2. *Knowing Lemma C.1, the gradient of the multilinear relaxation $\tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})$, is defined as follows*

$$\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_i} = \tilde{F}([\boldsymbol{\psi}]_{+i}, \boldsymbol{\theta}) - \tilde{F}([\boldsymbol{\psi}]_{-i}, \boldsymbol{\theta}), \tag{16}$$

where $[\boldsymbol{\psi}]_{+i}$, $[\boldsymbol{\psi}]_{-i}$ are operands setting $\psi_i = 1$ and $\psi_i = 0$, respectively.

Note that this derivation is a classic (Calinescu et al. 2011) and Eq. (16) can be computed by producing random samples of S.

D Technical Preliminaries

Theorem D.1. *(Multivariate Mean Value Theorem (Apostol 1974; Rudin 1976; Burke 2014)) If $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ is continuously differentiable, then for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, there exists a $\boldsymbol{z} \in [\boldsymbol{x}, \boldsymbol{y}]$, such that*

$$\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{y})\|_2 \leq \sup_{\boldsymbol{z} \in [\boldsymbol{x}, \boldsymbol{y}]} \|\partial \boldsymbol{f}(\boldsymbol{z})\|_F \|\boldsymbol{x} - \boldsymbol{y}\|_2,$$

where $\|\cdot\|_2$ is the L_2 norm and $\|\cdot\|_F$ is the Frobenius norm.

Definition D.2. A mapping $T : X \to X$ is called a *contraction* on X if there exists a constant $\epsilon \in [0, 1)$ such that for all $x, y \in X$,

$$\|T(x) - T(y)\|_2 \leq \epsilon \|x - y\|_2.$$

Theorem D.3. *(Banach's Fixed Point Theorem (Banach 1922)) Let $T : X \to X$ be a contraction on X. Then T has a unique fixed point $x^* \in X$ where $T(x^*) = x^*$.*

Theorem D.4. *(Implicit Function Theorem (Krantz and Parks 2002; Blondel et al. 2022)) Given a continuously differentiable function $G : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^n$, an implicitly defined function $\boldsymbol{x}^* : \mathbb{R}^d \to \mathbb{R}^n$ of $\boldsymbol{\theta} \in \mathbb{R}^d$, and an optimal solution $\boldsymbol{x}^*(\boldsymbol{\theta})$; let*

$$G(\boldsymbol{x}^*(\boldsymbol{\theta}), \boldsymbol{\theta}) = 0. \tag{17}$$

For $(\boldsymbol{x}_0, \boldsymbol{\theta}_0)$ satisfying $G(\boldsymbol{x}_0, \boldsymbol{\theta}_0) = 0$, if the Jacobian $\partial_{\boldsymbol{x}} G$ evaluated at $(\boldsymbol{x}_0, \boldsymbol{\theta}_0)$ is a square invertible matrix, then there exists a function $\boldsymbol{x}^*(\cdot)$ defined on a neighborhood of $\boldsymbol{\theta}_0$ such that $\boldsymbol{x}^*(\boldsymbol{\theta}_0) = x_0$. Furthermore, for all $\boldsymbol{\theta}$ in this neighborhood, we have that $G(x^*(\boldsymbol{\theta}), \boldsymbol{\theta}) = 0$ and $\partial x^*(\boldsymbol{\theta})$ exists. According to the chain rule, the Jacobian $\partial x^*(\boldsymbol{\theta})$ satisfies

$$\partial_{\boldsymbol{x}} G(\boldsymbol{x}^*(\boldsymbol{\theta}), \boldsymbol{\theta}) \partial \boldsymbol{x}^*(\boldsymbol{\theta}) + \partial_{\boldsymbol{\theta}} G(\boldsymbol{x}^*(\boldsymbol{\theta}), \boldsymbol{\theta}) = 0. \tag{18}$$

Therefore, computing $\partial \boldsymbol{x}^*(\boldsymbol{\theta})$ becomes the equivalent of solving the following linear system of equations

$$\underbrace{-\partial_{\boldsymbol{x}} G(\boldsymbol{x}^*(\boldsymbol{\theta}), \boldsymbol{\theta})}_{A \in \mathbb{R}^{n \times n}} \underbrace{\partial \boldsymbol{x}^*(\boldsymbol{\theta})}_{J \in \mathbb{R}^{n \times d}} = \underbrace{\partial_{\boldsymbol{\theta}} G(\boldsymbol{x}^*(\boldsymbol{\theta}), \boldsymbol{\theta})}_{B \in \mathbb{R}^{n \times d}}. \tag{19}$$

E Proof of Theorem 4.2

Before stating our proof we need to state the following corollary:

Corollary E.1. *Knowing Corollary C.2, we can write the* Hessian *of the multilinear relaxation as*

$$\begin{aligned} \frac{\partial^2 \tilde{F}(\boldsymbol{\psi}, \theta)}{\partial \psi_i \partial \psi_j} &= \left(\tilde{F}([\boldsymbol{\psi}]_{+i,+j}, \boldsymbol{\theta})] - \tilde{F}([\boldsymbol{\psi}]_{-i,+j}, \boldsymbol{\theta})] \right) - \left(\tilde{F}([\boldsymbol{\psi}]_{+i,-j}, \boldsymbol{\theta})] - \tilde{F}([\boldsymbol{\psi}]_{-i,-j}, \boldsymbol{\theta})] \right), \\ &= \tilde{F}([\boldsymbol{\psi}]_{+i,+j}, \boldsymbol{\theta})] - \tilde{F}([\boldsymbol{\psi}]_{-i,+j}, \boldsymbol{\theta}) - \tilde{F}([\boldsymbol{\psi}]_{+i,-j}, \boldsymbol{\theta}) + \tilde{F}([\boldsymbol{\psi}]_{-i,-j}, \boldsymbol{\theta}), \end{aligned} \tag{20}$$

if $i \neq j$, otherwise $\frac{\partial^2 \tilde{F}(\boldsymbol{\psi}, \theta)}{\partial \boldsymbol{\psi}_i \partial \boldsymbol{\psi}_j} = 0$.

We proof the following lemma using this corollary:

Lemma E.2. *Elements of the Hessian given in Eq.* (20) *are bounded with $4 \sup_{\boldsymbol{\psi} \in [\mathbf{0}, \mathbf{1}]} \left| \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}) \right|$, i.e., $\sup_{\psi_i, \psi_j \in [0,1]} \left| \frac{\partial^2 \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial_{\psi_i} \partial_{\psi_j}} \right| \leq 4 \sup_{\boldsymbol{\psi} \in [\mathbf{0}, \mathbf{1}]} \left| \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}) \right|$.*

Proof. By Corollary E.1, we have

$$\left| \frac{\partial^2 \tilde{F}(\boldsymbol{\psi}, \theta)}{\partial \psi_i \partial \psi_j} \right| = \left| \left(\tilde{F}([\boldsymbol{\psi}]_{+i,+j}, \boldsymbol{\theta})] - \tilde{F}([\boldsymbol{\psi}]_{-i,+j}, \boldsymbol{\theta})] \right) - \left(\tilde{F}([\boldsymbol{\psi}]_{+i,-j}, \boldsymbol{\theta})] - \tilde{F}([\boldsymbol{\psi}]_{-i,-j}, \boldsymbol{\theta})] \right) \right|.$$

Using the triangular inequality twice, we get

$$\begin{aligned} \left| \frac{\partial^2 \tilde{F}(\boldsymbol{\psi}, \theta)}{\partial \psi_i \partial \psi_j} \right| &\leq \left| \tilde{F}([\boldsymbol{\psi}]_{+i,+j}, \boldsymbol{\theta})] - \tilde{F}([\boldsymbol{\psi}]_{-i,+j}, \boldsymbol{\theta})] \right| + \left| \tilde{F}([\boldsymbol{\psi}]_{+i,-j}, \boldsymbol{\theta})] - \tilde{F}([\boldsymbol{\psi}]_{-i,-j}, \boldsymbol{\theta})] \right|, \\ &\leq \left| \tilde{F}([\boldsymbol{\psi}]_{+i,+j}, \boldsymbol{\theta})] \right| + \left| \tilde{F}([\boldsymbol{\psi}]_{-i,+j}, \boldsymbol{\theta})] \right| + \left| \tilde{F}([\boldsymbol{\psi}]_{+i,-j}, \boldsymbol{\theta})] \right| + \left| \tilde{F}([\boldsymbol{\psi}]_{-i,-j}, \boldsymbol{\theta})] \right|, \\ &\leq 4 \sup_{\boldsymbol{\psi} \in [\mathbf{0}, \mathbf{1}]} \left| \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}) \right|. \end{aligned}$$

□

Equipped with this lemma, we are ready to proof Thm. 4.2:

Proof. For simplicity, define a mapping $T_{\boldsymbol{\theta}} : [0, 1]^{|V|} \to [0, 1]^{|V|}$ where $T_{\boldsymbol{\theta}}(\boldsymbol{\psi}) = \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}))$. Given $\tilde{F}$ is a polynomial w.r.t. $\boldsymbol{\psi}$ and the sigmoid is a smooth function, $T_{\boldsymbol{\theta}}$ is continuously differentiable w.r.t. $\boldsymbol{\psi}$ in $[0, 1]^{|V|}$. By the multivariate equivalent of the mean-value theorem (see Thm. D.1 in App. D), for every $\boldsymbol{x}, \boldsymbol{y} \in |0, 1|^{|V|}$, there exists a $\boldsymbol{\psi} \in |0, 1|^{|V|}$, such that

$$\begin{aligned} \|T_{\boldsymbol{\theta}}(\boldsymbol{x}) - T_{\boldsymbol{\theta}}(\boldsymbol{y})\|_2 &\leq \sup_{\boldsymbol{\psi} \in [\mathbf{0}, \mathbf{1}]} \|\partial T_{\boldsymbol{\theta}}(\boldsymbol{\psi})\|_F \|\boldsymbol{x} - \boldsymbol{y}\|_2, \\ \|\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{x}, \boldsymbol{\theta})) - \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{y}, \boldsymbol{\theta}))\|_2 &\leq \sup_{\boldsymbol{\psi} \in [\mathbf{0}, \mathbf{1}]} \|\partial_{\boldsymbol{\psi}} \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}))\|_F \|\boldsymbol{x} - \boldsymbol{y}\|_2. \end{aligned} \tag{21}$$

From Eq. (25), we know that $\partial_{\boldsymbol{\psi}}\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})) = \left[\sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}}\right)\frac{\partial^2\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}\partial_{\psi_j}}\right]_{1\le i,j\le|V|}$ where $\sigma'(x) = (1+\exp(-x))^{-2}\cdot\exp(-x)$. Then,

$$\begin{aligned}
\sup_{\boldsymbol{\psi}\in[\mathbf{0},\mathbf{1}]}\|\partial_{\boldsymbol{\psi}}\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta}))\|_F &= \sup_{\boldsymbol{\psi}\in[\mathbf{0},\mathbf{1}]}\sqrt{\sum_i^{|V|}\sum_j^{|V|}\left|\sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}}\right)\frac{\partial^2\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}\partial_{\psi_j}}\right|^2},\\
&\le\sqrt{\sum_i^{|V|}\sum_j^{|V|}\left(\sup_{\psi_i,\psi_j\in[0,1]}\left|\sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}}\right)\frac{\partial^2\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}\partial_{\psi_j}}\right|\right)^2},\\
&= |V|\sup_{\psi_i,\psi_j\in[0,1]}\left|\sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}}\right)\frac{\partial^2\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}\partial_{\psi_j}}\right|,\\
&\le |V|\sup_{\psi_i\in[0,1]}\left|\sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}}\right)\right|\sup_{\psi_i,\psi_j\in[0,1]}\left|\frac{\partial^2\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}\partial_{\psi_j}}\right|.
\end{aligned}$$

Since $\arg\max_{x\in\mathbb{R}}\sigma'(x) = 0$, we know that $\sup_{\psi_i\in[0,1]}\left|\sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}}\right)\right| \le \frac{1}{4}$. Moreover, Lemma E.2 gives us a bound for the elements of the Hessian matrix where $\sup_{\psi_i,\psi_j\in[0,1]}\left|\frac{\partial^2\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial_{\psi_i}\partial_{\psi_j}}\right| \le 4\sup_{\boldsymbol{\psi}\in[\mathbf{0},\mathbf{1}]}\left|\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})\right|$. As a result,

$$\sup_{\boldsymbol{\psi}\in[\mathbf{0},\mathbf{1}]}\|\partial_{\boldsymbol{\psi}}\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta}))\|_F \le |V|\sup_{\boldsymbol{\psi}\in[\mathbf{0},\mathbf{1}]}\left|\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})\right|. \tag{22}$$

According to Asm. 4.1, we know that $\sup_{\boldsymbol{\psi}\in[\mathbf{0},\mathbf{1}]}\left|\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})\right| < \frac{1}{|V|}$. Therefore,

$$\sup_{\boldsymbol{\psi}\in[\mathbf{0},\mathbf{1}]}\|\partial_{\boldsymbol{\psi}}\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta}))\|_F < 1. \tag{23}$$

Plugging this in Eq. (21) above, we get

$$\|\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{x},\boldsymbol{\theta})) - \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{y},\boldsymbol{\theta}))\|_2 < \|\boldsymbol{x}-\boldsymbol{y}\|_2. \tag{24}$$

This means Eq. (7) is a contraction on $[0,1]^{|V|}$ (see Definition D.2 in App. D). Thus, according to Banach fixed-point theorem (see Thm. D.3 in App. D) the iterations given in Eq. (8) are bound to converge to a unique solution. □

F Proof of Theorem 4.3

Proof. For $n = |V|$,

$$\begin{aligned}
A &= I - \partial_{\boldsymbol{\psi}}\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})) = I - \begin{bmatrix}\frac{\partial\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta}))}{\partial\psi_1} & \dots & \frac{\partial\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta}))}{\partial\psi_n}\end{bmatrix},\\
&= I - \begin{bmatrix}\frac{\partial\sigma_1(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta}))}{\partial\psi_1} & \dots & \frac{\partial\sigma_1(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta}))}{\partial\psi_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial\sigma_n(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta}))}{\partial\psi_1} & \dots & \frac{\partial\sigma_n(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta}))}{\partial\psi_n}\end{bmatrix} = I - \begin{bmatrix}\frac{\partial\sigma\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_1}\right)}{\partial\psi_1} & \dots & \frac{\partial\sigma\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_1}\right)}{\partial\psi_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial\sigma\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_n}\right)}{\partial\psi_1} & \dots & \frac{\partial\sigma\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_n}\right)}{\partial\psi_n}\end{bmatrix},\\
&= I - \begin{bmatrix}\sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_1}\right)\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial^2\psi_1} & \dots & \sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_1}\right)\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_1\partial\psi_n}\\ \vdots & \ddots & \vdots\\ \sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_n}\right)\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_n\partial\psi_1} & \dots & \sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_n}\right)\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial^2\psi_n}\end{bmatrix},\\
&= I - \begin{bmatrix}\sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_1}\right) & \dots & 0\\ \vdots & \ddots & \vdots\\ 0 & \dots & \sigma'\left(\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_n}\right)\end{bmatrix}\begin{bmatrix}\frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial^2\psi_1} & \dots & \frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_1\partial\psi_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial\psi_n\partial\psi_1} & \dots & \frac{\partial\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta})}{\partial^2\psi_n}\end{bmatrix},\\
&= I - \Sigma'(\nabla_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta}))\cdot\nabla^2_{\boldsymbol{\psi}}\tilde{F}(\boldsymbol{\psi},\boldsymbol{\theta}),
\end{aligned} \tag{25}$$

where the function $\Sigma' : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ is defined as

$$\Sigma'(\mathbf{x}) = \begin{bmatrix} \sigma'(x_1) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma'(x_n) \end{bmatrix}$$

and $\sigma' : \mathbb{R} \to \mathbb{R}$ is

$$\sigma'(x) = (1 + \exp(-x))^{-2} \cdot \exp(-x).$$

Similarly,

$$\begin{aligned}
B = \partial_\theta \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})) &= \begin{bmatrix} \frac{\partial \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}))}{\partial \theta_1} & \dots & \frac{\partial \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}))}{\partial \theta_d} \end{bmatrix}, \\
&= \begin{bmatrix} \frac{\partial \sigma_1(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}))}{\partial \theta_1} & \dots & \frac{\partial \sigma_1(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}))}{\partial \theta_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial \sigma_n(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}))}{\partial \theta_1} & \dots & \frac{\partial \sigma_n(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}))}{\partial \theta_d} \end{bmatrix} = \begin{bmatrix} \frac{\partial \sigma\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_1}\right)}{\partial \theta_1} & \dots & \frac{\partial \sigma\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_1}\right)}{\partial \theta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \sigma\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_n}\right)}{\partial \theta_1} & \dots & \frac{\partial \sigma\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_n}\right)}{\partial \theta_n} \end{bmatrix}, \\
&= \begin{bmatrix} \sigma'\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_1}\right) \frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_1 \partial \theta_1} & \dots & \sigma'\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_1}\right) \frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_1 \partial \theta_d} \\ \vdots & \ddots & \vdots \\ \sigma'\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_n}\right) \frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_n \partial \theta_1} & \dots & \sigma'\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_n}\right) \frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_n \partial \theta_d} \end{bmatrix}, \\
&= \begin{bmatrix} \sigma'\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_1}\right) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma'\left(\frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_n}\right) \end{bmatrix} \begin{bmatrix} \frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_1 \partial \theta_1} & \dots & \frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_1 \partial \theta_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_n \partial \theta_d} & \dots & \frac{\partial \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{\partial \psi_n \partial \theta_d} \end{bmatrix}, \\
&= \Sigma'(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})) \cdot \partial_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}).
\end{aligned} \tag{26}$$

□

G Hessian Inverse Computation During Implicit Differentiation

Note first that the Hessian in question can be computed efficiently through Cor. E.1, using only function oracle calls: this is a well-known property of the multilinear relaxation.

Subsequently, we never have to invert matrices appearing in Eq. (12) to compute the Jacobian J: we need to solve a linear system. Better yet, as highlighted by Blondel et al. (2022), it is not necessary to form the entire Jacobian matrix: all of the readily implemented algorithms are so-called "matrix-free". It is sufficient to left-multiply or right-multiply the Jacobian by $\partial_{\boldsymbol{\psi}} G$ and $\partial_{\boldsymbol{\theta}} G$ in our notation (see Eq. (12): these are respectively called vector-Jacobian product (VJP) and Jacobian-vector product (JVP), which can be computed efficiently *without computing the full Jacobian*. JVPs in turn show up in forward passes while VJPs show up in backward passes of standard back-prop. In particular, left-multiplication (VJP) of $v^\top = \nabla_{\boldsymbol{\psi}^*} \mathcal{L}(\boldsymbol{\psi}^*(\boldsymbol{\theta}))$ with J, can be computed by first solving $A^\top u = v$ for u, where A as in Eq. (12). Then, $v^\top J$ can be obtained by $v^\top J = u^\top A J = u^\top B$.

Finally, we solve these linear systems via the readily implemented conjugate gradient method (Hestenes and Stiefel 1952) (`normalcg`) or via GMRES (Saad and Schultz 1986) (`gmres`). We treat the choice of backward solver selection as a hyperparameter and optimize w.r.t. different combinations using cross-validation. They are both indirect solvers and iteratively solve the linear system up to a given precision.

H Time Complexity of Finding the Root of the Fixed-Point in Equation (7)

In App. E, we base our proof on the fact that $\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}))$ is a contraction mapping when $\sup_{\boldsymbol{\psi} \in [\mathbf{0}, \mathbf{1}]} \|\partial_{\boldsymbol{\psi}} \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}))\|_F < 1$. Substituting $q = \sup_{\boldsymbol{\psi} \in [\mathbf{0}, \mathbf{1}]} \|\partial_{\boldsymbol{\psi}} \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta}))\|_F < 1$ back in Eq. (21), we obtain the Lipschitz constant for the fixed point as

$$\|\boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{x}, \boldsymbol{\theta})) - \boldsymbol{\sigma}(\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{y}, \boldsymbol{\theta}))\|_2 \leq q \|\boldsymbol{x} - \boldsymbol{y}\|_2.$$

For the sequence defined in Eq. (8), it holds that

$$\|\boldsymbol{\psi}^* - \boldsymbol{\psi}^{(K)}\|_2 \leq \frac{q^K}{1-q} \|\boldsymbol{\psi}^{(1)} - \boldsymbol{\psi}^{(0)}\|_2.$$

We wish to stop the iterations when $\frac{q^K}{1-q} \|\boldsymbol{\psi}^{(1)} - \boldsymbol{\psi}^{(0)}\|_2 \leq \epsilon$ and we know that $\boldsymbol{\psi}$ in $[0, 1]^{|V|}$, this means we need to run the iterative sequence given in Eq. (8) for $K \leq \frac{\log(\epsilon(1-q)/\sqrt{|V|})}{\log q}$ iterations.

I Additional Experiment Details and Results

I.1 Datasets

Moons and Gaussian. There are two classes in the synthetic datasets, whose labels are Bernoulli sampled with $p = 0.5$. Based on this label, the optimal subset and ground set pairs are constructed as follows: 1) sampling 10 points within the class as S^*; and 2) sampling 90 points from the other class as $V \backslash S^*$. This process is repeated until $|\mathcal{D}_{\text{validation + training}}| = 2000$ and $|\mathcal{D}_{\text{test}}| = 1000$.Following the experimental procedure of Ou et al. (2022), we use the SCIKIT-LEARN package (Pedregosa et al. 2011) to generate the Moons dataset, consisting of two interleaving moons with some noise with variance $\sigma^2 = 0.1$. For the Gaussian dataset, we sample data from a mixture of Gaussians $\frac{1}{2}\mathcal{N}(\mu_0, \mathbf{\Sigma}) + \frac{1}{2}\mathcal{N}(\mu_1, \mathbf{\Sigma})$, where $\mu_0 = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$, $\mu_1 = -\mu_0$, and $\mathbf{\Sigma} = \frac{1}{4}\mathbf{I}$.

CelebA. The CelebFaces Attributes dataset (CelebA) (Liu et al. 2015b) is a large-scale face dataset used extensively in computer vision research, particularly for tasks such as face detection, face attribute recognition, etc. The dataset contains $202,599$ face images of $10,177$ celebrities with various poses and backgrounds. Besides, each image is annotated with 40 binary attributes, describing facial features and properties (e.g., having a mustache, wearing a hat or glasses, etc.). Following previous work, we select 2 attributes at random and construct the set V with a size of 8 and the oracle set S^* with a size of 2 or 3, where neither attribute is present (e.g. not wearing glasses and hats).

Amazon. The Amazon Baby Registry dataset (Gillenwater et al. 2014) includes various subsets of baby registry products chosen by customers. These products are then organized into 18 distinct categories. From these, 12 categories are selected. Each product in the dataset is described by a textual description, which has been transformed into a 768-dimensional vector using a pre-trained BERT model (Devlin et al. 2019). For each category, the (V, S^*) pairs are sampled using the following process. First, we exclude subsets chosen by customers that contain only one item or more than 30 items. Next, we divide the remaining subsets into training, validation, and test sets equally. For each oracle subset $S^* \in S$, we randomly sample $30 - |S^*|$ additional products from the same category to ensure that V contains exactly 30 products. This method constructs a data point (V, S^*) for each customer, simulating a real-world scenario where V represents 30 products shown to the customer, and S^* represents the subset of products the customer is interested in.

BindingDB. BindingDB is a dataset designed to facilitate target selection in drug discovery. Given a set of compounds, the goal is to select the target to which a drug can bind to treat the disease effectively (Liu et al. 2015a). An ideal target should possess certain properties such as high biological activity, diversity, absorption, distribution, metabolism, excretion, and toxicology (ADME/Tox) (Bhhatarai et al. 2019; Singh et al. 2024). Thus, one needs to go through multiple filters sequentially, requiring intermediate signals that can be very expensive or impossible due to the privacy policy (Ou et al. 2022). However, by approaching the target selection from a set function perspective, we only require the compound sets and optimal target pairs. Here, we follow the steps in (Ou et al. 2022): a) choose two filters: high bioactivity and diversity. b) construct ground set with size $|V| = 300$. c) filter out one-third of the compounds with the highest bioactivity via a distance matrix based on the fingerprint similarity of molecules. d) ensure diversity by generating OS oracle S^* according to the centers of clusters which are presented by applying the affinity propagation algorithm. e) split the datasets so that the training, validation, and test set are 1,000, 100, and 100.

I.2 Hyperparameters

Both baseline and the proposed models are trained with an Adam optimizer (Kingma and Ba 2015) with a learning rate $\eta \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ and a batch size of 4 for BindingDB and 128 for the remaining datasets. We explore the number of layers, denoted as L, as a hyperparameter $L \in \{2, 3\}$.[4] For our proposed algorithms, we experiment with different root finding methods, i.e., forward solvers $\in \{\texttt{fpi}, \texttt{anderson}\}$ where `fpi` corresponds to fixed-point iterations in Eq. (8) and `anderson` corresponds to Anderson acceleration (Anderson 1965). For these root finding methods, we set the tolerance threshold to 10^{-6}, a very small value, to ensure convergence of fixed-point iterations. This is feasible as convergence is fast. We also try different methods to solve the linear system of equations (see Thm. 4.3) that show up during implicit differentiation, i.e., backward solvers in $\in \{\texttt{normalcg}, \texttt{gmres}\}$ where they stand for the conjugate gradient (Hestenes and Stiefel 1952) and GMRES (Saad and Schultz 1986) methods, respectively.

I.3 Normalizing the Gradient

Scaling the gradient of the multilinear relaxation as discussed in Sec. 5.1, modifies the fixed-point equation given in Eq. (7). In particular, when we scale $\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})$; we, in effect, solve the following fixed-point equation:

$$\boldsymbol{\psi} = \boldsymbol{\sigma}\left(\frac{2\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})}{|V|Q}\right). \tag{27}$$

[4] The BindingDB dataset requires larger memory for training and, thus, is only tested for 2 layers due to computational limits.

Set Function ($F_{\boldsymbol{\theta}}$)
InitLayer
SumPooling
FC(256, 500, ReLU)
FC(500, 500, ReLU)
FC(256, 1, $-$)

Table 3: Summary of the architecture of $F_{\boldsymbol{\theta}}$. InitLayer is dataset specific. FC(d_{input}, d_{output}, f) is a fully connected layer with an input dimension of d_{input}, output dimension of d_{output}, and activation function f.

CelebA	Gaussian	Moons	Amazon	BindingDB	
				Drug	**Target**
Conv(32, 3, 2, ReLU)				Conv(32, 4, 1, ReLU)	Conv(32, 4, 1, ReLU)
Conv(64, 4, 2, ReLU)				Conv(64, 6, 1, ReLU)	Conv(64, 8, 1, ReLU)
Conv(128, 5, 2, ReLU)	FC(2, 256, $-$)	FC(2, 256, $-$)	FC(768, 256, $-$)	Conv(96, 8, 1, ReLU)	Conv(96, 12, 1, ReLU)
MaxPooling				MaxPooling	MaxPooling
FC(128, 256, $-$)				FC(96, 256, ReLU)	FC(96, 256, ReLU)
				Concatenation	
				FC(512, 256, $-$)	

Table 4: Summary of the initial layer architectures for $F_{\boldsymbol{\theta}}$.

We choose $Q = c$ to be a constant for $\texttt{iDiffMF}_c$ and treat c as a hyperparameter. We report the results in Tab. 6. For $\texttt{iDiffMF}_2$ and $\texttt{iDiffMF}_*$, we choose $Q = \|\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})\|_2$ and $Q = \|\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})\|_*$, respectively. We note that when computing the gradient through implicit differentiation, we take this change into account.

I.4 Software and Hardware

We conduct experiments on a DGX Station A100 equipped with four A100 (80GB). The operating system is Ubuntu 22.04.4 LTS with x86-64 architecture, powered by an AMD EPYC 7742 64-Core Processor(4TB of DDR4 memory and a 256MB L3 cache). All experiments are executed using Python 3.9, with PyTorch 2.2.1 and JAX 0.4.26 as the primary software packages. We provide an environment file in the supplement illustrating the version of all other packages.

I.5 Inference Details

After training, the learned objective $F_{\boldsymbol{\theta}}(\cdot)$ is used to produce an optimal subset $\hat{S}_i^*$ given query V_i in the test set. In particular, during inference in the test time, $\boldsymbol{\psi}^{(0)}$ is set to $0.5 * \mathbf{1}$ and one step of the fixed-point iterations is applied to $\boldsymbol{\psi}^{(0)}$ with the learned $F_{\boldsymbol{\theta}}$ to obtain $\boldsymbol{\psi}^*$ except for the `DiffMF` algorithm. For the `DiffMF` algorithm, $K = 5$ steps of fixed-point iterations is applied. This is in accordance with the implementation of Ou et al. (2022). We also experiment with letting fixed-point iterations converge during inference. We report these results in Tab. 7. Then, the corresponding prediction, $\hat{S}_i^*$, is found by applying topN rounding to $\boldsymbol{\psi}^*$, i.e., top $|S_i^*|$ elements are chosen to be in the prediction set $\hat{S}_i^*$.

The mean JC is obtained by computing the Jaccard coefficient between these prediction sets and the ground truth sets S_i^* in the test set. The Jaccard Coefficient of two sets is defined as the average of the size of their intersection divided by the size of their union. Mathematically, for two sets A and B, the Jaccard Coefficient $J(A, B)$ is given by $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$. Then, the mean JC overall optimal oracle subset pairs are $MJC(S_i^*, \hat{S}_i^*) = \frac{1}{N} \sum_{i=1}^{N} J(S_i^*, \hat{S}_i^*)$.

I.6 Architectures

We use the same permutation invariant architecture for the set function $F_{\boldsymbol{\theta}}$ as in Ou et al. (2022). For completeness, we also include the architecture information here. All datasets share the same architecture except for their initial layer which encodes the set objects into vector representations. The summary of the set function architecture is given below.

I.7 Additional Results

We report the hyperparameter settings that perform best on the validation sets for each dataset and method for reproducibility purposes. Please see Tab. 5 and Tab. 6 for details.

	Datasets	EquiVSet$_{\text{ind}}$		EquiVSet$_{\text{copula}}$		DiffMF		iDiffMF$_2$				iDiffMF$_*$			
		η	L	η	L	η	L	η	L	forward solver	backward solver	η	L	forward solver	backward solver
AD	CelebA	0.001	3	0.001	3	0.001	3	0.01	3	fpi	normal cg	0.01	3	fpi	normal cg
	Gaussian	0.0001	3	0.0001	3	0.0001	3	0.001	2	fpi	normal cg	0.0001	2	fpi	gmres
	Moons	0.00001	2	0.00001	2	0.0001	3	0.001	3	anderson	gmres	0.00001	2	fpi	gmres
PR (Amazon)	apparel	0.001	3	0.0001	3	0.0001	3	0.0001	3	anderson	normal cg	0.0001	3	fpi	normal cg
	bath	0.0001	3	0.0001	3	0.0001	3	0.0001	2	anderson	normal cg	0.00001	3	fpi	normal cg
	bedding	0.0001	3	0.0001	3	0.0001	2	0.0001	2	anderson	normal cg	0.0001	2	fpi	normal cg
	carseats	0.001	2	0.0001	2	0.001	3	0.0001	3	fpi	normal cg	0.0001	3	anderson	normal cg
	diaper	0.001	2	0.0001	2	0.0001	2	0.0001	3	fpi	normal cg	0.0001	3	fpi	normal cg
	feeding	0.0001	3	0.0001	3	0.0001	3	0.001	3	anderson	normal cg	0.001	3	anderson	gmres
	furniture	0.001	2	0.001	2	0.001	3	0.0001	2	fpi	gmres	0.0001	3	fpi	normal cg
	gear	0.0001	2	0.0001	2	0.0001	2	0.0001	3	fpi	normal cg	0.0001	3	fpi	normal cg
	health	0.0001	2	0.0001	3	0.0001	3	0.0001	3	anderson	normal cg	0.0001	3	fpi	normal cg
	media	0.0001	3	0.0001	3	0.001	3	0.0001	3	fpi	gmres	0.0001	3	fpi	normal cg
	safety	0.0001	3	0.001	3	0.0001	3	0.0001	3	fpi	gmres	0.0001	3	fpi	normal cg
	toys	0.001	3	0.0001	3	0.0001	3	0.0001	3	anderson	normal cg	0.0001	3	anderson	normal cg
CS	BindingDB	0.0001	2	0.0001	2	0.0001	2	0.0001	2	fpi	normal cg	0.0001	2	fpi	normal cg

Table 5: Best performing hyperparameters on average across all folds during cross-validation. We obtain the results on Tab. 2 based on the hyperparameter combinations we report on this table. We perform grid search on the Cartesian product of $\eta \in \{0.00001, 0.0001, 0.001, 0.01\}$ and $L \in \{2, 3\}$ for the EquiVSet$_{\text{ind}}$, EquiVSet$_{\text{copula}}$, and DiffMF algorithms, where we perform grid search on the combinations of $\eta \in \{0.00001, 0.0001, 0.001, 0.01\}$, $L \in \{2, 3\}$, forward solver $\in \{\texttt{fpi}, \texttt{anderson}\}$, and backward solver $\in \{\texttt{normalcg}, \texttt{gmres}\}$ for the iDiffMF$_2$ and iDiffMF$_*$ algorithms on all datasets except for BindingDB. Due to its size, we only explore the same range of learning rates for fixed L, forward and backward solver choices.

	Datasets	iDiffMF$_c$						
		Test JC	**Time** (s)	η	L	forward solver	backward solver	c
AD	CelebA	53.93 ± 0.63	2211.87 ± 140.61	0.001	2	anderson	normal cg	10000
	Gaussian	90.94 ± 0.08	60.61 ± 10.19	0.01	2	anderson	normal cg	1000
	Moons	58.37 ± 0.32	87.62 ± 20.65	0.001	2	anderson	normal cg	100
PR (Amazon)	apparel	48.08 ± 0.95	77.97 ± 4.07	0.001	2	fpi	normal cg	100
	bath	50.17 ± 0.97	42.46 ± 2.11	0.00001	2	anderson	gmres	100
	bedding	47.93 ± 1.29	132.59 ± 14.31	0.001	2	anderson	normal cg	10000
	carseats	20.60 ± 1.39	38.00 ± 11.92	0.01	2	fpi	normal cg	1000
	diaper	57.13 ± 5.37	100.37 ± 31.09	0.0001	2	anderson	gmres	100
	feeding	49.15 ± 7.20	124.59 ± 45.12	0.0001	2	anderson	gmres	100
	furniture	17.65 ± 0.70	30.77 ± 3.35	0.01	2	fpi	gmres	100
	gear	42.39 ± 1.00	52.27 ± 1.15	0.001	2	anderson	normal cg	100
	health	39.82 ± 0.34	57.10 ± 2.88	0.001	2	fpi	normal cg	1000
	media	41.32 ± 1.35	46.02 ± 4.25	0.01	2	anderson	normal cg	100
	safety	20.50 ± 2.00	36.76 ± 6.84	0.01	2	fpi	gmres	100
	toys	40.87 ± 2.96	40.00 ± 3.23	0.01	2	fpi	gmres	100
CS	BindingDB	74.72 ± 1.73	11121.13 ± 2553.24	0.001	2	anderson	normal cg	100

Table 6: Test Jaccard coefficient and training time for set anomaly detection (AD), product recommendation (PR) and compound selection (CS) tasks for the iDiffMF$_c$ algorithm, i.e., the version of iDiffMF after scaling the objective with a constant c. We construct this algorithm by multiplying $\nabla_{\boldsymbol{\psi}} \tilde{F}(\boldsymbol{\psi}, \boldsymbol{\theta})$ with $2/(|V|c)$.

	Datasets	EquiVSet$_{\text{ind}}$		EquiVSet$_{\text{copula}}$		DiffMF		iDiffMF$_2$		iDiffMF$_*$	
		Test JC	**Time** (s)	**Test JC**	**Time** (s)	**Test JC**	**Time** (s)	**Test JC**	**Time** (s)	**Test JC**	**Time** (s)
AD	CelebA	55.02 ± 0.20	1151.17 ± 698.13	56.16 ± 0.81	1195.47 ± 731.84	54.42 ± 0.70	1299.13 ± 984.20	55.18 ± 0.77	1955.56 ± 260.31	56.33 ± 0.73	1670.36 ± 259.22
	Gaussian	90.55 ± 0.06	30.68 ± 3.86	90.94 ± 0.09	39.11 ± 6.09	90.96 ± 0.05	85.75 ± 35.82	$\mathbf{91.0 \pm 0.02}$	46.0 ± 5.23	$\underline{90.95 \pm 0.06}$	55.76 ± 14.16
	Moons	57.76 ± 0.11	66.99 ± 4.43	$\underline{58.67 \pm 0.18}$	62.03 ± 6.82	58.45 ± 0.15	58.24 ± 3.01	58.45 ± 0.32	77.56 ± 16.27	$\mathbf{58.95 \pm 0.1}$	57.33 ± 10.38
PR (Amazon)	apparel	68.45 ± 0.96	38.32 ± 6.63	$\mathbf{78.19 \pm 0.89}$	77.14 ± 14.37	70.60 ± 1.35	63.06 ± 16.12	$\underline{76.06 \pm 4.56}$	121.7 ± 52.21	73.73 ± 5.91	139.77 ± 58.75
	bath	67.51 ± 1.19	34.01 ± 5.89	$\mathbf{77.72 \pm 1.98}$	53.29 ± 6.68	71.87 ± 0.27	61.84 ± 12.73	$\underline{77.49 \pm 1.19}$	79.65 ± 11.73	75.51 ± 0.45	151.23 ± 22.02
	bedding	66.20 ± 1.10	40.99 ± 3.59	$\mathbf{77.26 \pm 1.24}$	67.13 ± 12.78	67.66 ± 0.39	72.69 ± 7.73	$\underline{77.07 \pm 2.02}$	118.84 ± 40.73	76.93 ± 1.05	93.63 ± 18.62
	carseats	19.99 ± 1.01	12.38 ± 4.19	20.03 ± 0.15	12.19 ± 2.71	20.15 ± 0.65	10.53 ± 5.01	$\underline{22.24 \pm 1.44}$	45.82 ± 8.58	$\mathbf{22.42 \pm 1.04}$	54.50 ± 12.11
	diaper	74.26 ± 0.73	60.96 ± 17.79	$\mathbf{83.66 \pm 0.69}$	193.55 ± 80.28	81.74 ± 1.18	95.22 ± 10.54	$\underline{82.75 \pm 0.63}$	154.05 ± 27.68	81.65 ± 0.74	184.36 ± 57.70
	feeding	71.46 ± 0.43	68.43 ± 26.08	$\mathbf{82.47 \pm 0.19}$	95.18 ± 21.75	77.44 ± 0.46	93.27 ± 18.81	$\underline{81.54 \pm 1.80}$	165.30 ± 45.35	81.52 ± 1.84	245.12 ± 55.21
	furniture	17.28 ± 0.88	10.98 ± 2.44	17.95 ± 0.80	10.03 ± 3.23	16.84 ± 0.05	9.31 ± 1.79	$\mathbf{20.29 \pm 2.50}$	42.42 ± 6.82	$\underline{18.69 \pm 0.93}$	36.90 ± 6.05
	gear	65.35 ± 0.91	40.89 ± 3.19	$\mathbf{77.33 \pm 0.90}$	69.44 ± 10.22	66.06 ± 2.86	60.95 ± 10.38	$\underline{73.91 \pm 10.30}$	112.19 ± 56.94	73.47 ± 10.81	127.13 ± 56.82
	health	63.04 ± 0.41	33.51 ± 5.22	72.03 ± 0.77	60.18 ± 6.31	59.64 ± 0.81	51.66 ± 2.54	$\underline{72.14 \pm 1.39}$	95.60 ± 25.53	$\mathbf{72.31 \pm 1.04}$	116.89 ± 32.01
	media	$\mathbf{56.60 \pm 0.56}$	37.45 ± 11.06	55.73 ± 1.18	45.02 ± 4.95	51.32 ± 1.11	40.69 ± 4.65	$\underline{56.41 \pm 2.63}$	73.1 ± 21.26	55.20 ± 1.62	67.3 ± 6.66
	safety	21.99 ± 1.85	10.39 ± 1.87	22.09 ± 3.30	13.14 ± 3.13	24.66 ± 5.56	8.59 ± 1.31	$\mathbf{25.98 \pm 1.73}$	51.08 ± 7.64	$\underline{25.73 \pm 2.33}$	47.96 ± 6.39
	toys	62.36 ± 1.31	34.06 ± 6.69	$\mathbf{69.08 \pm 1.04}$	47.81 ± 9.46	64.39 ± 1.64	43.96 ± 6.89	68.55 ± 1.13	79.46 ± 23.40	$\underline{68.70 \pm 1.01}$	109.88 ± 34.23
CS	BindingDB	73.59 ± 0.75	9934.30 ± 2591.36	73.57 ± 2.05	13983.93 ± 4458.52	73.22 ± 1.08	21472.44 ± 3239.73	$\underline{76.97 \pm 0.74}$	13004.75 ± 5611.39	$\mathbf{77.42 \pm 0.64}$	12097.14 ± 1985.58

Table 7: Test Jaccard Coefficient (JC) and training time for set anomaly detection (AD), product recommendation (PR), and compound selection (CS) tasks, across all five algorithms. iDiffMF$_2$ and iDiffMF$_*$ correspond to our algorithm with Frobenius and nuclear norm scaling. This table differs from Tab. 2 in its iDiffMF columns. In this table, fixed-point iterations are run until convergence during inference. **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.