Neural Estimation of Submodular Functions with Applications to Differentiable Subset Selection

Anonymous Author(s)
Affiliation
Address
email

Abstract

Submodular functions and variants, through their ability to characterize diversity and coverage, have emerged as a key tool for data selection and summarization. Many recent approaches to learn submodular functions suffer from limited expressiveness. In this work, we propose FLEXSUBNET, a family of flexible neural models for both monotone and non-monotone submodular functions. To fit a latent submodular function from (set, value) observations, FLEXSUBNET applies a concave function on modular functions in a recursive manner. We do not draw the concave function from a restricted family, but rather learn from data using a highly expressive neural network that implements a differentiable quadrature procedure. Such an expressive neural model for concave functions may be of independent interest. Next, we extend this setup to provide a novel characterization of monotone α-submodular functions, a recently introduced notion of approximate submodular functions. We then use this characterization to design a novel neural model for such functions. Finally, we consider learning submodular set functions under distant supervision in the form of (perimeter-set, high-value-subset) pairs. This yields a novel subset selection method based on an order-invariant, yet greedy sampler built around the above neural set functions. Our experiments on synthetic and real data show that FLEXSUBNET outperforms several baselines.

1 Introduction

Owing to their strong characterization of diversity and coverage, submodular functions and their extensions, *viz.*, weak and approximate submodular functions, have emerged as a powerful machinery in data selection tasks [34, 64, 24, 51, 68, 48]. We propose trainable parameterized families of submodular functions under two supervision regimes. In the first setting, the goal is to estimate the submodular function based on (set, value) pairs, where the function outputs the value of an input set. Such a setup may arise in applications like auction design where one may like to learn a player's valuation function based on her bids [5]. In the second setting, the task is to estimate the submodular function under the supervision of (perimeter-set, high-value-subset) pairs, where high-value-subset potentially maximizes the underlying function against all other subsets of perimeter-set. The trained function is expected to extract high-value subsets from freshly-specified perimeter sets. Such scenario has applications in itemset prediction in recommendation [60, 61], data summarization [37, 2, 4, 10, 57], *etc*.

1.1 Our contributions

Driven by the above motivations, we propose (i) a novel family of highly expressive neural models for submodular and α-submodular functions which can be estimated under the supervisions of both (set, value) and (perimeter-set, high-value-subset) pairs; (ii) a novel permutation adversarial training method for differentiable subset selection, which efficiently trains submodular and α-submodular functions based on (perimeter-set, high-value-subset) pairs. We provide more details below.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Neural models for submodular functions. We design FLEXSUBNET, a family of flexible neural models for monotone, non-monotone submodular functions and monotone α-submodular functions.

— *Monotone submodular functions.* We model a monotone submodular function using a recursive neural model which outputs a concave composed submodular function [18, 58, 39] at every step of recursion. Specifically, it first computes a linear combination of the submodular function computed in the previous step and a modular function and, then applies a concave function to the result to output the next submodular function.

— *Monotone α-submodular function.* Our proposed model for submodular function rests on the principle that a concave composed submodular function is always submodular. However, to the best of our knowledge, there is no known result for an α-submodular function. We address this gap by showing that an α-submodular function can be represented by applying a mapping φ on a positive modular set function, where φ satisfies a second-order differential inequality. Subsequently, we provide a neural model representing the universal approximator of φ, which in turn is used for modeling an α-submodular function.

— *Non-monotone submodular functions.* By applying a non-monotone concave function on modular function, we extend our model to non-monotone submodular functions.

Several recent models learn subclasses of submodular functions [40, 59, 61]. Bilmes and Bai [6] present a thorough theoretical characterization of the benefits of network depth with fixed concave functions, in a general framework called deep submodular functions (DSF). DSF leaves open all design choices: the number of layers, their widths, the DAG topology, and the choice of concave functions (no implementation or experimental details are available). All-to-all attention has replaced domain-driven topology design in much of NLP [11]. Set transformers [38] would therefore be a natural alternative to compare against DSF, but need more memory and computation. Here we explore the following third, somewhat extreme trade-off: we restrict the topology to a single recursive chain, thus providing an effectively plug-and-play model with no topology and minimal hyperparameter choices (mainly the length of the chain). However, we compensate with a more expressive, trainable concave function that is shared across all nodes of the chain. Our experiments show that our strategy improves ease of training and predictive accuracy beyond both set transformers and various DSF instantiations with fixed concave functions. Related work is discussed in more detail in Appendix B.

Permutation insensitive differentiable subset selection. It is common [59, 54, 48] to select a subset from a given dataset by sequentially sampling elements using a softmax distribution obtained from the outputs of a set function on various sets of elements. At the time of learning set functions based on (perimeter-set, high-value-subset) pairs, this protocol naturally results in order-sensitivity in the training process for learning set functions. To mitigate this, we propose a novel max-min optimization. Such a formulation sets forth a game between an adversarial permutation generator and the set function learner — where the former generates the worst-case permutations to induce minimum likelihood of training subsets and the latter keeps maximizing the likelihood function until the estimated parameters become permutation-agnostic. To this end, we use a Gumbel-Sinkhorn neural network [43] as a neural surrogate of hard permutations, that expedites the underlying training process and allows us to avoid combinatorial search on large permutation spaces.

Experiments. We first experiment with several submodular set functions and synthetically generated examples, which show that FLEXSUBNET recovers the function more accurately than several baselines. Later experiments with several real datasets on product recommendation reveal that FLEXSUBNET can predict the purchased items more effectively and efficiently than several baselines.

2 Design of FLEXSUBNET

In this section, we first present our notations and then propose a family of flexible neural network models for monotone and non-monotone submodular functions and and α-submodular functions.

2.1 Notation and preliminary results

We denote $V = \{1, 2, .., |V|\}$ as the ground set or universal set of elements and $S, T \subseteq V$ as subsets of V. Each element $s \in V$ may be associated with a feature vector $\boldsymbol{z}_s$. Given a set function $F : 2^V \to \mathbb{R}$, we define the marginal utility $F(s \,|\, S) := F(S \cup \{s\}) - F(S)$. The function F is called *monotone* if $F(s \,|\, S) \geq 0$ whenever $S \subset V$ and $s \in V \backslash S$; F is called *α-submodular* with $\alpha > 0$ if $F(s \,|\, S) \geq \alpha F(s \,|\, T)$ whenever $S \subseteq T$ and $s \in V \backslash T$ [24, 66, 13]. As a special case, F is submodular if $\alpha = 1$ and F is modular if $F(s \,|\, S) = F(s \,|\, T)$. Here, $F(\cdot)$ is called *normalized* if

$F(\varnothing) = 0$. Similarly, a real function $f : \mathbb{R} \to \mathbb{R}$ is *normalized* if $f(0) = 0$. Unless otherwise stated, we only consider normalized set functions in this paper. We quote a key result often used for neural modeling for submodular functions [18, 58, 39].

Proposition 1. *Given the set function $F : 2^V \to \mathbb{R}^+$ and a real valued function $\phi : \mathbb{R} \to \mathbb{R}$, (i) the set function $\phi(F(\cdot))$ is monotone submodular if F is monotone submodular and ϕ is an increasing concave function; and, (ii) $\phi(F(\cdot))$ is non-monotone submodular if F is positive modular and ϕ is non-monotone.*

2.2 Monotone submodular functions

Overview. Our model for monotone submodular functions consists of a neural network which cascades the underlying functions in a recursive manner for N steps. Specifically, to compute the submodular function $F^{(n)}(\cdot)$ at step n, it first linearly combines the submodular function $F^{(n-1)}(\cdot)$ computed in the previous step and a *trainable* positive modular function $m^{(n)}(\cdot)$ and then, applies a monotone concave activation function ϕ on it.

Recursive model. We model the submodular function $F_\theta(\cdot)$ as follows:

$$F^{(0)}(S) = m_\theta^{(0)}(S);\ F^{(n)}(S) = \phi_\theta\big(\lambda F^{(n-1)}(S) + (1-\lambda) m_\theta^{(n)}(S)\big);\ F_\theta(S) = F^{(N)}(S); \quad (1)$$

where the iterations are indexed by $1 \leq n \leq N$, $\{m_\theta^{(n)}(\cdot)\}$ is a sequence of positive modular functions, driven by a neural network with parameter θ. $\lambda \in [0, 1]$ is a tunable or trained parameter. We apply a linear layer with positive weights on the each feature vector $\boldsymbol{z}_s$ to compute the value of $m_\theta^{(n)}(\cdot)$ and then compute $m_\theta^{(n)}(S) = \sum_{s \in S} m_\theta^{(n)}(s)$. Moreover, ϕ_θ is an increasing concave function which, as we shall see later, is modeled using neural networks. Under these conditions, one can use Proposition 1(i) to easily show that $F_\theta(S)$ is a monotone submodular function (Appendix C).

2.3 Monotone-α-submodular functions

Our characterization for submodular functions in Eq. (1) are based on Proposition 1(i), which implies that a concave composed submodular function is submodular. However, to the best of our knowledge, a similar characterization of α-submodular functions is lacking in the literature. To address this gap, we first introduce a novel characterization of monotone α-submodular functions and then use it to design a recursive model for such functions.

Novel characterization of α-submodular function. In the following, we show how we can characterize an α-submodular function using a differential inequality (proven in Appendix C).

Theorem 2. *Given the function $\varphi : \mathbb{R} \to \mathbb{R}^+$ and a modular function $m : V \to [0, 1]$, the set function $F(S) = \varphi(\sum_{s \in S} m(s))$ is monotone α-submodular for $|S| \leq k$, if $\varphi(x)$ is increasing in x and $\frac{\mathrm{d}^2\varphi(x)}{\mathrm{d}x^2} \leq \frac{1}{k} \log\left(\frac{1}{\alpha}\right) \frac{\mathrm{d}\varphi(x)}{\mathrm{d}x}$.*

The above theorem also implies that given $\alpha = 1$, then $F(S) = \varphi(\sum_{s \in S} m(s))$ is monotone submodular if $\varphi(x)$ is concave in x, which reduces to a particular case of Proposition 1(i). Once we design $F(S)$ by applying φ on a modular function, our next goal is to design more expressive and flexible modeling in a recursive manner similar to Eq. (1). To this end, we extend Proposition 1 to the case for α-submodular functions.

Proposition 3. *Given a monotone α-submodular function $F(\cdot)$, $\phi(F(S))$ is monotone α-submodular, if $\phi(\cdot)$ is an increasing concave function. Here, α remains the same for F and $\phi(F(\cdot))$.*

Note that, when $F(S)$ is submodular, *i.e.*, $\alpha = 1$, the above result reduces to Proposition 1 in the context of concave composed modular functions.

Recursive model. Similar to Eq. (1) for submodular functions, our model for α-submodular functions is also driven by an RNN which maintains an α-submodular function and updates its value recursively for N iterations. However, in contrast to FLEXSUBNET, where the underlying submodular function is initialized with a positive modular function, we initialize the corresponding α-submodular function $F^{(0)}(S)$ with $\varphi_\theta(m_\theta^{(0)}(S))$, where φ_θ is a trainable function satisfying the conditions of Theorem 2. Then we recursively apply a trainable monotone concave function on $F^{(n-1)}(S)$ for $n \in [N]$ to build

a flexible model for α-submodular function. Formally, we have:

$$F^{(0)}(S) = \varphi_\theta(m_\theta^{(0)}(S));\ F^{(n)}(S) = \phi_\theta\left(\lambda F^{(n-1)}(S) + (1-\lambda) m_\theta^{(n)}(S)\right);\ F_\theta(S) = F^{(N)}(S); \tag{2}$$

with $\lambda \in [0, 1]$. Here, m_θ, φ_θ and ϕ_θ are realized using neural networks parameterized by θ. Then, using Proposition 3 and Theorem 2, we can show that $F_\theta(S)$ is α-submodular in S (proven in Proposition 6 in Appendix C).

2.4 Non-monotone submodular functions

In contrast to monotone set functions, non monotone submodular functions can be built by applying concave function on top of *only one modular function* rather than submodular function (Proposition 1 (i) vs. (ii)). To this end, we model a non-monotone submodular function $F_\theta(\cdot)$ as follows:

$$F_\theta(S) = \psi_\theta(m_\theta(S)) \tag{3}$$

where $m_\theta(\cdot)$ is positive modular function but $\psi_\theta(\cdot)$ can be a *non-monotone* concave function. Both m_θ and ψ_θ are trainable functions realized using neural networks parameterized by θ. One can use Proposition 1(ii) to show that $F_\theta(S)$ is a non-monotone submodular function.

2.5 Neural parameterization of $m_\theta, \phi_\theta, \psi_\theta, \varphi_\theta$

We complete the neural parameterization introduced in Sections 2.2–2.4. Each model has two types of component functions: (i) the modular set function m_θ; and, (ii) the concave functions ϕ_θ (Eq. (1) and (2)) and ψ_θ (Eq. (3)) and the non-concave function φ_θ (Eq. (2)). While modeling m_θ is simple and straightforward, designing neural models for the other components, *i.e.*, ϕ_θ, ψ_θ and φ_θ is non-trivial. As mentioned before, because we cannot rely on the structural complexity of our 'network' (which is a simple linear recursion) or a judiciously pre-selected library of concave functions [6], we need to invest more capacity in the concave function, effectively making it universal.

Monotone submodular function. Our model for monotone submodular function described in Eq. (1) consists of two neural components: the sequence of modular functions $\{m_\theta^{(n)}\}$ and ϕ_θ.

— *Parameterization of* m_θ. We model the modular function $m_\theta^{(n)} : 2^V \to \mathbb{R}^+$ in Eq. (1) as $m_\theta^{(n)}(S) = \sum_{s\in S} \theta \cdot \boldsymbol{z}_s$, where $\boldsymbol{z}_s$ is the feature vector for each element $s \in S$ and both $\theta, \boldsymbol{z}_s$ are non-negative.

— *Parameterization of* ϕ_θ. Recall that ϕ_θ is an increasing concave function. We model it using the fact that a differentiable function is concave if its second derivative is negative. We focus on capturing the second derivative of the underlying function using a complex neural network of arbitrary capacity, providing a negative output. Hence, we use a positive neural network h_θ to model the second derivative

$$\frac{\mathrm{d}^2\phi_\theta(x)}{\mathrm{d}x^2} = -h_\theta(x) \le 0. \tag{4}$$

Now, since ϕ_θ is increasing, we have:

$$\frac{\mathrm{d}\phi_\theta(x)}{\mathrm{d}x} = \int_{b=x}^{b=\infty} h_\theta(b)\,\mathrm{d}b \ge 0 \implies \phi_\theta(x) = \int_{a=0}^{a=x}\int_{b=a}^{b=\infty} h_\theta(b)\,\mathrm{d}b\,\mathrm{d}a. \tag{5}$$

Here, $\phi_\theta(\cdot)$ is normalized, *i.e.*, $\phi_\theta(0) = 0$, which ensures that $\{F_\theta^{(n)}\}$ in Eq. (1) are also normalized. An offset in Eq. (5) allows a nonzero initial value of $\phi_\theta(\cdot)$, if required. Note that monotonicity and concavity of ϕ_θ can be achieved by the restricting positivity of h_θ. Such a constraint can be ensured by setting $h_\theta(\cdot) = \mathrm{ReLU}(\Lambda_\theta^{(h)}(\cdot))$, where $\Lambda_\theta^{(h)}(\cdot)$ is any complex neural network. Hence, $\phi_\theta(\cdot)$ represents a class of universal approximators of normalized increasing concave functions, if $\Lambda_\theta^{(h)}(\cdot)$ is an universal approximator of continuous functions [25]. For a monotone submodular function of the form of concave composed modular function, we have the following result (Proven in Appendix C).

Proposition 4. *Given an universal set V, a constant $\epsilon > 0$ and a submodular function $F(S) = \phi(\sum_{s\in S} m(\boldsymbol{z}_s))$ where $\boldsymbol{z}_s \in \mathbb{R}^d$, $S \subset V$, $0 \le m(\boldsymbol{z}) < \infty$ for all $\boldsymbol{z} \in \mathbb{R}^d$. Then there exists two fully connected neural networks m_{θ_1} and h_{θ_2} of width $d+4$ and 5 respectively, each with ReLU activation function, such that the following conditions hold:*

$$\left\| F(S) - \int_{a=0}^{a=\sum_{s\in S} m_{\theta_1}(\boldsymbol{z}_s)} \int_{b=a}^{b=\infty} h_{\theta_2}(b)\,\mathrm{d}b\,\mathrm{d}a. \right\| \le \epsilon \quad \forall\, S \subset V \tag{6}$$

Monotone α-submodular model. An α-submodular model described in Eq. (2) has three trainable components: (i) the sequence of modular functions $\{m_\theta^{(n)}(\cdot)\}$, (ii) the concave function $\phi_\theta(\cdot)$ and (iii) $\varphi_\theta(\cdot)$. For the first two components, we reuse the parameterizations used for monotone submodular functions. In the following, we describe our proposed neural parameterization of φ_θ.

— *Parameterization of* $\varphi_\theta(\cdot)$. From Theorem 2, we note that $\varphi_\theta(\cdot)$ is increasing and satisfies $\frac{\mathrm{d}^2\varphi_\theta(x)}{\mathrm{d}x^2} \leq \kappa(\alpha)\frac{\mathrm{d}\varphi_\theta(x)}{\mathrm{d}x}$ where, $\kappa(\alpha) = \frac{1}{k}\log(1/\alpha)$. It implies that

$$e^{-x\kappa(\alpha)}\frac{\mathrm{d}^2\varphi_\theta(x)}{\mathrm{d}x^2} - \kappa(\alpha)e^{-x\kappa(\alpha)}\frac{\mathrm{d}\varphi_\theta(x)}{\mathrm{d}x} \leq 0 \implies \frac{\mathrm{d}}{\mathrm{d}x}\left(e^{-x\kappa(\alpha)}\frac{\mathrm{d}\varphi_\theta(x)}{\mathrm{d}x}\right) \leq 0 \tag{7}$$

Driven by the last inequality, we have

$$e^{-x\kappa(\alpha)}\frac{\mathrm{d}\varphi_\theta(x)}{\mathrm{d}x} = \int_x^\infty g_\theta(b)\,\mathrm{d}b \implies \varphi_\theta(x) = \int_{a=0}^{a=x} e^{a\kappa(\alpha)}\int_{b=a}^{b=\infty} g_\theta(b)\,\mathrm{d}b\,\mathrm{d}a \tag{8}$$

Parameterizing non-monotone submodular model. As suggested by Eq. (3), our model for non-monotone submodular function contains a non-monotone concave function ψ_θ and a modular function m_θ. We model m_θ using the same parameterization used for the monotone set functions. We parameterize the ψ_θ as follows.

— *Parameterization of* ψ_θ. Modeling a generic (possibly non-monotone) submodular function requires a general form of concave function ψ_θ which is not necessarily increasing. The trick is to design $\psi_\theta(\cdot)$ in such a way that its second derivative is negative everywhere, whereas its first derivative can have any sign. For $x \in [0, x_{\max}]$, we have:

$$\psi_\theta(x) = \int_{a=0}^{a=x}\int_{b=a}^{b=\infty} h_\theta(b)\,\mathrm{d}b\,\mathrm{d}a - \int_{a=x_{\max}-x}^{a=x_{\max}}\int_{b=a}^{b=\infty} g_\theta(b)\,\mathrm{d}b\,\mathrm{d}a, \tag{9}$$

where $h_\theta, g_\theta \geq 0$. Moreover, we assume that $\int_0^\infty h_\theta(b)\,\mathrm{d}b$ and $\int_0^\infty g_\theta(b)\,\mathrm{d}b$ are convergent. We use $x_{\max}$ in the upper limit of the second integral to ensure that ψ_θ is normalized, *i.e.*, $\psi_\theta(0) = 0$. Next, we compute the first derivative of $\psi_\theta(x)$ as:

$$\frac{\mathrm{d}\psi_\theta(x)}{\mathrm{d}x} = \int_{b=x}^{b=\infty} h_\theta(b)\,\mathrm{d}b - \int_{b=x_{\max}-x}^{b=\infty} g_\theta(b)\,\mathrm{d}b, \tag{10}$$

which can have any sign, since both integrals are positive. Here, the second derivative of ψ_θ becomes

$$\frac{\mathrm{d}^2\psi_\theta(x)}{\mathrm{d}x^2} = -h_\theta(x) - g_\theta(x_{\max} - x) \leq 0 \tag{11}$$

which implies that ψ_θ is concave. Similar to $h_\theta(\cdot)$, we can model $g_\theta(\cdot) = \mathrm{ReLU}(\Lambda_\theta^g(\cdot))$. Such a representation makes $\psi_\theta(\cdot)$ a universal approximator of normalized concave functions. As suggested by Eq. (3), ψ_θ takes $m_\theta(S)$ as input. Therefore, in practice, we set $x_{\max} = \max_S m_\theta(S)$.

2.6 Parameter estimation from (set, value) pairs

In this section, our goal is to learn θ from a set of pairs $\{(S_i, y_i) \mid i \in [I]\}$, such that $y_i \approx F_\theta(S_i)$. Hence, our task is to solve the following optimization problem:

$$\min_\theta \textstyle\sum_{i\in[I]}(y_i - F_\theta(S_i))^2 \tag{12}$$

In the following, we discuss methods to solve this problem.

Backpropagation through double integral. Each of our proposed set function models consists of one or more double integrals. Thus computation of the gradients of the loss function in Eq. (12) requires gradients of these integrals. To compute them, we leverage the methods proposed by Wehenkel and Louppe [63], which specifically provide forward and backward procedures for neural networks involving integration. Specifically, we use the Clenshaw-Curtis (CC) quadrature or Trapezoidal Rule for numerical computation of $\phi_\theta(\cdot)$ and $\psi_\theta(\cdot)$. On the other hand, we compute the gradients $\nabla_\theta\,\phi_\theta(\cdot)$ and $\nabla_\theta\,\psi_\theta(\cdot)$ by leveraging Leibniz integral rule [15]. In practice, we replace the upper limit ($b = \infty$) of the inner integral in Eqs. (5), (8) and (9) with $b_{\max} = x_{\max}$ during training.

Decoupling into independent integrals. The aforementioned end-to-end training can be challenging in practice, since the gradients are also integrals which can make loss convergence elusive. Moreover, it is also inefficient, since for each step of CC quadrature of the outer integral, we need to perform numerical integration of the entire inner integral. To tackle this limitation, we decouple the underlying double integral into two single integrals, parameterized using two neural networks.

— *Monotone submodular functions.* During training our monotone submodular function model in Eq. (1), we model ϕ_θ in Eq. (5) as:

$$\phi_\theta(x) = \int_0^x \phi'_\theta(a)\,\mathrm{d}a, \quad \phi'_\theta(x) = \int_x^\infty h_\beta(a)\,\mathrm{d}a \tag{13}$$

In contrast to end-to-end training of θ where ϕ_θ was realized only via neural network of h_θ, here we use two decoupled networks, *i.e.*, ϕ'_θ and h_β. Then, we learn θ and β by minimizing the regularized sum of squared error, *i.e.*,

$$\min_{\theta,\beta} \sum_{i\in[I]} \sum_{n\in[N]} \Big[\rho\Big(\phi'_\theta(G^{(n)}(S_i)) - \int_{G^{(n)}(S_i)}^\infty h_\beta(a)\,\mathrm{d}a\Big)^2 + \big(y_i - F_\theta(S_i)\big)^2\Big]. \tag{14}$$

Here, $G^{(n)}(S) = \lambda F^{(n-1)}(S) + (1-\lambda) m_\theta^{(n)}(S)$ is the input to ϕ_θ in the recursion (1). Recall that N is the number of steps in the recursion. Since ϕ_θ is monotone, we need $\phi'_\theta \geq 0$ which is ensured by $\phi'_\theta(x) = \mathrm{ReLU}(\Lambda_\theta^{\phi'}(x))$. In principle, we would like to have $\phi'_\theta(x) = \int_x^\infty h_\beta(a)\,\mathrm{d}a$ for all $x \in \mathbb{R}$. In practice, we approximate this by penalizing the regularizer values in the domain of interest—the values which are fed as input to ϕ_θ.

While there are potential hazards to such an approximation, in our experiments, the benefits outweighed the risks. The above parameterization involves only single integrals. The use of an auxiliary network h_β allows more flexibility, leading to improved robustness during training optimization. The above minimization task achieves approximate concavity of ϕ via training, whereas backpropagating through double integrals enforces concavity by design. Appendix D extends this method for α-submodular and non-monotone submodular functions.

3 Differentiable subset selection

In Section 2.6, we tackled the problem of learning F_θ from (set, value) pairs. In this section, we consider learning F_θ from a given set of (perimeter-set, high-value-subset) pair instances.

3.1 Learning F_θ from (perimeter-set, high-value-subset)

Let us assume that the training set $\mathcal{U}$ consists of $\{(V, S)\}$ pairs, where V is some arbitrary subset of the universe of element, and $S \subseteq V$ is a high-value subset of V. Given a set function model F_θ, our goal is to estimate θ so that, across all possible cardinality constrained subsets in $S' \subseteq V$ with $|S'| = |S|$, $F_\theta(\cdot)$ attains its maximum value at S. Formally, we wish to estimate θ so that, for all possible $(V, S) \in \mathcal{U}$, we have:

$$S = \mathrm{argmax}_{S'\subset V}\, F_\theta(S'), \text{ subject to } |S'| = |S|. \tag{15}$$

3.2 Probabilistic greedy model

The problems of maximizing both monotone submodular and monotone α-submodular functions rely on a greedy heuristic [45]. It sequentially chooses elements maximizing the marginal gain and hence, cannot directly support backpropagation. Tschiatschek et al. [61] tackle this challenge with a probabilistic model which greedily samples elements from a softmax distribution with the marginal gains as input. Having chosen the first j elements of high-value-subset from perimeter-set V, it draws the $(j+1)^{\text{th}}$ element from the remaining candidates in V with probability proportional to the marginal utility of the candidate. If the elements of S under permutation π are written as $\pi(S)_j$ for $j \in [|S|]$, then the probability of selecting the elements of the set S in the sequence $\pi(S)$ is given by

$$\Pr{}_\theta(\pi(S) \,|\, V) = \prod_{j=0}^{|S|-1} \frac{\exp(\tau F_\theta(\pi(S)_{j+1} \,|\, \pi(S)_{\leq j}))}{\sum_{s\in V\setminus\pi(S)_{\leq j}} \exp(\tau F_\theta(s \,|\, \pi(S)_{\leq j}))} \tag{16}$$

Here, F_θ is the underlying submodular function, τ is a temperature parameter and $S_0 = \emptyset$.

Bottleneck in estimating θ. Note that, the above model (16) generates the elements in a sequential manner and is sensitive to π. Tschiatschek et al. [61] attempt to remove this sensitivity by maximizing the sum of probability (16) over all possible π:

$$\theta^* = \mathrm{argmax}_\theta \sum_{(V,S)\in\mathcal{U}} \log \sum_{\pi\in\Pi_{|S|}} \Pr{}_\theta(\pi(S) \,|\, V). \tag{17}$$

However, enumerating all such permutations for even a medium-sized subset is expensive both in terms of time and space. They attempt to tackle this problem by approximating the mode and the mean of $\Pr(\cdot)$ over the permutation space $\Pi_{|S|}$. But, it still require searching over the entire permutation space, which is extremely time consuming.

3.3 Proposed approach

Here, we describe our proposed method of permutation adversarial parameter estimation that avoids enumerating all possible permutations of the subset elements, while ensuring that the learned parameter θ^* remains permutation invariant.

Max-min optimization problem. We first set up a max-min game between a permutation generator and the maximum likelihood estimator (MLE) of θ. Here, for each subset S, the permutation generator produces an adversarial permutation $\pi \in \Pi_{|S|}$ which induces a low value of the underlying likelihood. On the other hand, MLE learns θ in the face of these adversarial permutations. Hence, we have the following optimization problem:

$$\max_{\theta} \min_{\pi \in \Pi_{|S|}} \sum_{(V,S)\in\mathcal{U}} \log \Pr{}_{\theta}(\pi(S) \mid V) \tag{18}$$

Differentiable surrogate for permutations. Solving the inner minimization in (18) requires searching for π over $\Pi_{|S|}$, which appears to confer no benefit beyond (17). To sidestep this limitation, we relax the inner optimization problem by using a doubly stochastic matrix $\boldsymbol{P} \in \mathcal{P}_{|S|}$ as an approximation for the corresponding permutation π. Suppose $\boldsymbol{Z}_S = [\boldsymbol{z}_s]_{s\in S}$ is the feature matrix where each row corresponds to an element of S. Then $\boldsymbol{Z}_{\pi(S)} \approx \boldsymbol{P}\boldsymbol{Z}_S$. Thus, $\Pr_\theta(\pi(S)|V)$ can be evaluated by iterating down the rows of $\boldsymbol{P}\boldsymbol{Z}_S$ and, therefore, written in the form $\Pr_\theta(\boldsymbol{P}, S)$. Thus, we get a continuous approximation to (18):

$$\max_{\theta} \min_{\boldsymbol{P} \in \mathcal{P}_{|S|}} \sum_{(V,S)\in\mathcal{U}} \log \Pr{}_{\theta}(\boldsymbol{P}, S \mid V) \tag{19}$$

so that we can learn θ by continuous optimization. We generate the soft permutation matrices $\boldsymbol{P}$ using a Gumbel-Sinkhorn network [43]. Given a subset S, it takes a seed matrix $\boldsymbol{B}_S$ as input and generates $\boldsymbol{P}^S$ in a recursive manner:

$$\boldsymbol{P}^0 = \exp(\boldsymbol{B}_S/t); \quad \boldsymbol{P}^k = \mathbb{D}_c\left(\mathbb{D}_r\left(\boldsymbol{P}^{(k-1)}\right)\right) \tag{20}$$

Here, t is a temperature parameter; and, $\mathbb{D}_c$ and $\mathbb{D}_r$ provide column-wise and row-wise normalization. Thanks to these two operations, $\boldsymbol{P}^k$ tends to a doubly stochastic matrix for sufficiently large k. Denoting $\boldsymbol{P}^\infty = \lim_{k\to\infty} \boldsymbol{P}^k$, one can show [43] that

$$\boldsymbol{P}^\infty = \operatorname*{argmax}_{\boldsymbol{P}\in\mathcal{P}_{|S|}} \operatorname{Tr}(\boldsymbol{P}^\top \boldsymbol{B}_S) - t\sum_{i,j} \boldsymbol{P}(i,j)\log \boldsymbol{P}(i,j)$$

where the temperature t controls the 'hardness' of the "soft permutation" $\boldsymbol{P}$. As $t \to 0$, $\boldsymbol{P}$ tends to be a hard permutation matrix [8]. The above procedure demands different seeds $\boldsymbol{B}_S$ across different pairs of $(V, S) \in \mathcal{U}$, which makes the training computationally expensive in terms of both time and space. Therefore, we model $\boldsymbol{B}_S$ by feeding the feature matrix $\boldsymbol{Z}_S$ into an additional neural network G_ω which is shared across different (V, S) pairs. Then the optimization (19) reduces to $\max_\theta \min_\omega \sum_{(V,S)\in\mathcal{U}} \log \Pr_\theta(\omega, S)$, with ω taking the place of $\boldsymbol{P}$.

4 Experiments

In this section, we first evaluate FLEXSUBNET using a variety of synthesized set functions and show that it is able to fit them under the supervision of $(set, value)$ pairs more accurately than the state-of-the-art methods. These experiments are consistent with recent work on learning submodular functions, except that our planted functions are more challenging. Next, we use datasets gathered from Amazon baby registry [19] to show that FLEXSUBNET can learn to select data from (perimeter-set, high-value-subset) pairs more effectively than several baselines.

4.1 Training by (set, value) pairs

Dataset generation. We generate $|V|{=}10^4$ samples, where we draw the feature vector $\boldsymbol{z}_s$ for each sample $s{\in}V$ uniformly at random, *i.e.*, $\boldsymbol{z}_s{\in}\text{Unif}[0,1]^d$ with $d{=}10$. Then, we generate subsets S of different sizes by randomly gathering elements from the set V. Finally, we compute the values of $F(S)$ for different submodular functions F. Specifically, we consider seven planted set functions: (i) **Log**: $F(S){=}\log(\sum_{s\in S} \mathbf{1}^\top \boldsymbol{z}_s)$, (ii) **LogDet**: $F(S) = \log\det(\mathbb{I} + \sum_{s\in S} \boldsymbol{z}_s\boldsymbol{z}_s^\top)$ [56, 7], (iii) **Facility location (FL)**: $F(S) = \sum_{s'\in V} \max_{s\in S} \boldsymbol{z}_s^\top \boldsymbol{z}_{s'}/(||\boldsymbol{z}_s||||\boldsymbol{z}_{s'}||)$ [44, 16, 12, 46], (iv) **Monotone graph cut (Gcut$_{\sigma=0.1}$)**: In general, Gcut$_\sigma$ is computed using $F(S) := \sum_{u\in V, v\in S} \boldsymbol{z}_u^\top \boldsymbol{z}_v - \sigma \sum_{u,v\in S} \boldsymbol{z}_u^\top \boldsymbol{z}_v$. It measures the weighted cut across $(S, V\backslash S)$ when the weight of the edge (u, v) is computed as $\boldsymbol{z}_u^\top \boldsymbol{z}_v$ [26, 55, 28, 29, 30]. Here, σ trades off between diversity and representation. We set $\sigma = 0.1$. Note that Gcut$_\sigma$ is monotone (non-monotone) submodular when

	Log	LogDet	FL	$\text{Gcut}_{\sigma=0.1}$	Log×Sqrt	Log×LogDet	$\text{Gcut}_{\sigma=0.8}$
FLEXSUBNET	**0.015 ± 0.000**	**0.013 ± 0.000**	**0.022 ± 0.000**	**0.004 ± 0.000**	**0.032 ± 0.000**	**0.025 ± 0.000**	**0.068 ± 0.001**
Set-transformer	0.060 ± 0.001	0.029 ± 0.000	0.063 ± 0.001	0.014 ± 0.000	0.037 ± 0.000	0.051 ± 0.001	0.171 ± 0.002
Deep-set	0.113 ± 0.002	0.044 ± 0.000	0.179 ± 0.003	0.058 ± 0.001	0.079 ± 0.001	0.070 ± 0.001	0.075 ± 0.001
DSF	0.258 ± 0.003	0.684 ± 0.007	0.189 ± 0.003	0.778 ± 0.009	0.240 ± 0.003	0.274 ± 0.003	0.970 ± 0.010
SubMix	0.148 ± 0.002	0.063 ± 0.001	0.172 ± 0.002	0.018 ± 0.000	0.158 ± 0.002	0.154 ± 0.002	1.722 ± 0.015

Table 1: Performance measured in terms of RMSE on synthetically generated examples using seven set functions, *viz.*, Log, LogDet, Facility Location (FL) and Graph cut ($\text{Gcut}_{\sigma=0.1}$), Log×Sqrt, Log×LogDet and $\text{Gcut}_{\sigma=0.8}$ for FLEXSUBNET, Set-transformer [38], Deep-set [65], deep submodular function (DSF) [6] and mixture submodular function (SubMix) [59]. Among them, the first four datasets are monotone submodular functions, Log×Sqrt and Log×LogDet are α-submodular functions and $\text{Gcut}_{\sigma=0.8}$ is a non-monotone submodular function. Number in **bold** font (underline) indicate the best (second best) performers.

σ<0.5 (σ>0.5). (v) **Log×Sqrt**: $F(S) = [\log(\sum_{s\in S} \mathbf{1}^\top \boldsymbol{z}_s)]\cdot[\sum_{s\in S} \mathbf{1}^\top \boldsymbol{z}_s]^{1/2}$, (vi) **Log×LogDet**: $F(S) = [\log(\sum_{s\in S} \mathbf{1}^\top \boldsymbol{z}_s)] \cdot [\log\det(\mathbb{I} + \sum_{s\in S} \boldsymbol{z}_s \boldsymbol{z}_s^\top)]$ and (vii) **Non monotone graph cut ($\text{Gcut}_{\sigma=0.8}$)**: It is the graph cut function in (iv) with $\sigma = 0.8$. Among above set functions, (i)–(iv) are monotone submodular functions, (v)–(vi) are monotone α-submodular functions and (vii) is a non-monotone submodular function. We set the number of steps in the recursions (1), (2) as $N = 2$ using cross validation.

Evaluation protocol. We sample $|V|$=10000 (set,value) instances as described above and split them into train, dev and test folds of equal size. We present the train and dev folds to the set function model and measure the performance in terms of RMSE on test fold instances. In the first four datasets, we used our monotone submodular model described in Eq. (1); in case of Log×Sqrt and Log×LogDet, we used our α-submodular model described in Eq. (2); and, for $\text{Gcut}_{\sigma=0.8}$, we used our non-monotone submodular model in Eq. (3). For α-submodular model, we tuned α using cross validation. Appendix E provides hyperparameter tuning details for all methods.

Baselines. We compare FLEXSUBNET against four state-of-the-art models, *viz.*, Set-transformer [38], Deep-set [65], deep submodular function (DSF) [6] and mixture submodular function (SubMix) [59]. Appendix E contains more details about the baselines.

Results. We compare the performance of FLEXSUBNET against the above four baselines in terms of RMSE on the test fold. Table 1 summarizes the results. We make the following observations. (1) FLEXSUBNET outperforms the baselines by a substantial margin across all datasets. (2) Set–transformer outperforms all the other baselines. While both Deep-set and Set-transformer ensure permutation invariant aggregation of the feature vectors $\{\boldsymbol{z}_s \mid s \in S\}$, Set-transformer allows explicit interactions between set elements via all-to-all attention layers. As a result, it outperforms all other baselines even without explicitly using the knowledge of submodularity in its network architecture. Note that, like Deep-set, FLEXSUBNET does not directly model any interaction between the elements. However, it explicitly models submodularity or α-submodularity in the network architecture, which helps it outperform the baselines. (3) While, in principle, DSF function family contains that of FLEXSUBNET, standard multilayer instantations of DSF with fixed concave functions cannot learn well from (set,value) training.

Next, we compare different variants of our approach: (i) FLEXSUBNET, trained by decoupling the double integral into independent integrals (Eq. (14)); (ii) FLEXSUBNET, trained using end-to-end training via backpropagation through double integrals (Section 2.6); and (iii) FLEXSUBNET with $N = 1$, where N is the number of steps of the recursions (1). Table 2 summarizes the results which reveal the following observations. (1) We achieve substantial performance gain via decoupling into independent integrals. Although decoupling integrals is an approximation of end-to-end training, it also provides a more smooth loss surface than the loss on the double integral. (2) Running FLEXSUBNET for only a single step significantly deteriorates performance.

	Log	LogDet	FL
Decoupling	**0.015 ± 0.000**	**0.013 ± 0.000**	**0.022 ± 0.000**
End-end	0.078 ± 0.001	0.073 ± 0.001	0.078 ± 0.001
Our ($N = 1$)	0.089 ± 0.001	0.075 ± 0.001	0.089 ± 0.001

Table 2: Variants of our approach.

4.2 Training by (perimeter-set, high-value-subset)

Datasets. We use the Amazon baby registry dataset [19] which contains 17 product categories. Among them, we only consider those categories where $|V| > 50$, where V is the total number of items in the universal set. These categories are: (i) Gear, (ii) Bath, (iii) Health, (iv) Diaper, (v) Toys, (vi) Bedding, (vii) Feeding, (viii) Apparel and (ix) Media. They are also summarized in Appendix F.

	Mean Jaccard Coefficient (MJC)						Mean NDCG@10					
	FLEXSUBNET	DSF	SubMix	FL	DPP	DisMin	FLEXSUBNET	DSF	SubMix	FL	DPP	DisMin
Gear	**0.101**	0.099	0.028	0.019	0.014	0.013	**0.539**	0.538	0.449	0.433	0.425	0.426
Bath	**0.091**	0.087	0.038	0.020	0.012	0.010	**0.520**	0.500	0.447	0.433	0.427	0.422
Health	**0.153**	0.142	0.022	0.084	0.011	0.015	**0.597**	0.549	0.449	0.540	0.425	0.435
Diaper	**0.134**	0.115	0.023	0.018	0.013	0.012	**0.562**	0.546	0.447	0.440	0.435	0.435
Toys	**0.157**	0.150	0.025	0.064	0.029	0.029	**0.591**	0.577	0.446	0.472	0.448	0.449
Bedding	**0.203**	0.191	0.028	0.015	0.043	0.047	**0.643**	0.623	0.437	0.438	0.456	0.461
Feeding	**0.100**	0.091	0.026	0.023	0.020	0.019	**0.550**	0.547	0.459	0.453	0.454	0.452
Apparel	**0.101**	0.093	0.036	0.022	0.016	0.016	**0.558**	0.550	0.459	0.452	0.446	0.444
Media	**0.135**	0.130	0.029	0.035	0.029	0.025	**0.578**	**0.578**	0.474	0.470	0.461	0.461

Table 3: Prediction of subsets in product recommendation task. Performance is measured in terms of Jaccard Coefficient (JC) and Mean NDCG@10 for nine categories from Amazon baby registry records, for FLEXSUBNET, Deep submodular function (DSF), mixture of submodular functions (SubMix), Facility location (FL), Determinantal point process (DPP) and Disparity Min (DisMin). Numbers in **bold** (underline) indicate best (second best) performer.

Evaluation protocol. Each dataset contains a universal set V and a set of subsets $\mathcal{S} = \{S\}$. Each item s is characterized by a short textual description such as "bath: Skip Hop Moby Bathtub Elbow Rest, Blue : Bathtub Side Bumpers : Baby". From this text, we compute $\boldsymbol{z}_s$ using BERT [11]. We split $\mathcal{S}$ into equal-sized training ($\mathcal{S}_{\text{train}}$), dev ($\mathcal{S}_{\text{dev}}$) and test ($\mathcal{S}_{\text{test}}$) folds. We disclose the training and dev sets to the submodular function models, which are trained using our proposed permutation adversarial approach (Section 3). Then the trained model is used to sample item sequence S' with prefix $S'_{\leq K} = \{s_1, \ldots, s_K\}$. Here, s_i is the item selected at the ith step of the greedy algorithm. We assess the quality of the sampled sequence using two metrics.

Mean Jaccard coefficient (MJC). Given a set T in the test set, we first measure the overlap between T and the first $|T|$ elements of S' using Jaccard coefficient, *i.e.*, $JC(T) = |S'_{\leq|T|} \cap T| / |S'_{\leq|T|} \cup T|$ and then average over all subsets in the test set, *i.e.*, $T \in \mathcal{S}_{\text{test}}$ to compute mean Jaccard coefficient [27].

Mean NDCG@10. The greedy sampler outputs item sequence S'. For each $T \in \mathcal{S}_{\text{test}}$, we compute the NDCG given by the order of first 10 elements of S', where $i \in S'$ is assigned a gold relevance label 1, if $i \in T$ and 0, otherwise. Finally, we average over all test subsets to compute Mean NDCG@10.

Our method vs. baselines. We compare the subset selection ability of FLEXSUBNET against several baselines which include two trainable submodular models: (i) Deep submodular function (DSF) and (ii) Mixture of submodular functions (SubMix) described in Section 4.1; two popular non-trainable submodular functions which include (iii) Facility location (FL) [44, 16], (iv) Determinantal point process (DPP) [7]; and, a non-submodular function (v) Disparity Min (DisMin) which is often used in data summarization [10]. Here, we use the monotone submodular model of FLEXSUBNET. Appendix F contains more details about the baselines. We did not consider general purpose set functions, *e.g.*, Set-transformer, Deep-set, etc., because they cannot be maximized using greedy-like algorithms and therefore, we cannot apply our proposed method in Section 3.

Results. Table 3 provides a comparative analysis across all candidate set functions, which shows that: (i) FLEXSUBNET outperforms all the baselines across all the datasets; (ii) DSF is the second best performer across all datasets; and, (iii) the performance of non-trainable set functions is poor, as they are *not trained to mimic* the set selection process. Recall our key motivation was to train parameters invariant to the order of the elements of the input subset. Tschiatschek et al. [61] achieve this goal using a Monte Carlo average of the likelihood over many samples. In Figure 4, we compare our method with theirs for a representative dataset (Gear), which shows that our method is $\geq 4\times$ faster.

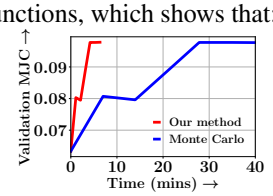

Figure 4: Efficiency.

5 Conclusion

We introduced FLEXSUBNET: a family of submodular functions, represented by neural networks that implement quadrature-based numerical integration, and supports end-to-end backpropagation through these integration operators. We designed a permutation adversarial subset selection method, which ensures that the estimated parameters are independent of greedy item selection order. On both synthetic and real datasets, FLEXSUBNET improves upon recent competitive formulations. Our work opens up several avenues of future work. One can extend our work for γ-weakly submodular functions [14]. Another extension of our work is to leverage other connections to convexity, *e.g.*, the Lovasz extension [41, 1] similar to Karalias et al. [31].

References

[1] Francis Bach. Learning with submodular functions: A convex optimization perspective. *arXiv preprint arXiv:1111.6453*, 2011.

[2] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 671–680, 2014.

[3] Wenruo Bai, William S Noble, and Jeff A Bilmes. Submodular maximization via gradient ascent: the case of deep submodular functions. *Advances in neural information processing systems*, 2018:7989, 2018.

[4] Ramakrishna Bairi, Rishabh Iyer, Ganesh Ramakrishnan, and Jeff Bilmes. Summarization of multi-document topic hierarchies using submodular mixtures. In *ACL*, pages 553–563, 2015.

[5] Maria-Florina Balcan and Nicholas JA Harvey. Learning submodular functions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 793–802, 2011.

[6] Jeffrey Bilmes and Wenruo Bai. Deep submodular functions. *arXiv preprint arXiv:1701.08939*, 2017.

[7] Alexei Borodin. Determinantal point processes. *arXiv preprint arXiv:0911.1153*, 2009.

[8] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26:2292–2300, 2013.

[9] Abhimanyu Das and David Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. *arXiv preprint arXiv:1102.3975*, 2011.

[10] Anirban Dasgupta, Ravi Kumar, and Sujith Ravi. Summarization through submodularity and dispersion. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1014–1022, 2013.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] Donglei Du, Ruixing Lu, and Dachuan Xu. A primal-dual approximation algorithm for the facility location problem with submodular penalties. *Algorithmica*, 63(1):191–200, 2012.

[13] Marwa El Halabi and Stefanie Jegelka. Optimal approximation for unconstrained non-submodular minimization. In *International Conference on Machine Learning*, pages 3961–3972. PMLR, 2020.

[14] Ethan R Elenberg, Rajiv Khanna, Alexandros G Dimakis, and Sahand Negahban. Restricted strong convexity implies weak submodularity. *arXiv preprint arXiv:1612.00804*, 2016.

[15] Harley Flanders. Differentiation under the integral sign. *The American Mathematical Monthly*, 80(6):615–627, 1973.

[16] Alan M Frieze. A cost function property for plant location problems. *Mathematical Programming*, 7(1):245–248, 1974.

[17] Satoru Fujishige. *Submodular functions and optimization*. Elsevier, 2005.

[18] Satoru Fujishige and Satoru Iwata. Minimizing a submodular function arising from a concave function. *Discrete applied mathematics*, 92(2-3):211–215, 1999.

[19] Jennifer A Gillenwater, Alex Kulesza, Emily Fox, and Ben Taskar. Expectation-maximization for learning determinantal point processes. *Advances in Neural Information Processing Systems*, 27:3149–3157, 2014.

[20] Akhil Gupta, Lavanya Marla, Ruoyu Sun, Naman Shukla, and Arinbjörn Kolbeinsson. Pender: Incorporating shape constraints via penalized derivatives. In *AAAI*, 2021. URL https://www.aaai.org/AAAI21Papers/AAAI-9099.GuptaA.pdf.

[21] Maya R. Gupta, Erez Louidor Ilan, Olexander Mangylov, Nobuyuki Morioka, Taman Narayan, and Sen Zhao. Multidimensional shape constraints. In *ICML*, 2020. URL http://proceedings.mlr.press/v119/gupta20b/gupta20b.pdf.

[22] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[23] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300, 2004.

[24] Abolfazl Hashemi, Mahsa Ghasemi, Haris Vikalo, and Ufuk Topcu. Submodular observation selection and information gathering for quadratic models. *arXiv preprint arXiv:1905.09919*, 2019.

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

[26] Rishabh Iyer, Ninad Khargoankar, Jeff Bilmes, and Himanshu Asanani. Submodular combinatorial information measures with applications in machine learning. In *Algorithmic Learning Theory*, pages 722–754. PMLR, 2021.

[27] Paul Jaccard. 'E comparative study of floral distribution in a portion of the alps and jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.

[28] Stefanie Jegelka and Jeff Bilmes. Notes on graph cuts with submodular edge weights. In *NIPS 2009 Workshop on Discrete Optimization in Machine Learning: Submodularity, Sparsity Polyhedra (DISCML)*, pages 1–6, 2009.

[29] Stefanie Jegelka and Jeff Bilmes. Cooperative cuts: Graph cuts with submodular edge weights. 2010.

[30] Stefanie Jegelka and Jeff Bilmes. Submodularity beyond submodular energies: coupling edges in graph cuts. In *CVPR 2011*, pages 1897–1904. IEEE, 2011.

[31] Nikolaos Karalias, Joshua David Robinson, Andreas Loukas, and Stefanie Jegelka. Neural extensions: Training neural networks with set functions. 2021.

[32] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

[33] Rajiv Khanna, Ethan Elenberg, Alexandros G Dimakis, Sahand Negahban, and Joydeep Ghosh. Scalable greedy feature selection via weak submodularity. *arXiv preprint arXiv:1703.02723*, 2017.

[34] Krishnateja Killamsetty, Durga Subramanian, Ganesh Ramakrishnan, and Rishabh Iyer. Glister: A generalization based data selection framework for efficient and robust learning. *In AAAI*, 2021.

[35] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[36] Ivan Kobyzev, Simon Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43:3964–3979, 2021. URL http://arxiv.org/pdf/1908.09257.

[37] Suraj Kothawade, Jiten Girdhar, Chandrashekhar Lavania, and Rishabh Iyer. Deep submodular networks for extractive data summarization. *arXiv preprint arXiv:2010.08593*, 2020.

[38] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *ICML*, 2019.

[39] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 510–520, 2011.

[40] Hui Lin and Jeff A Bilmes. Learning mixtures of submodular shells with application to document summarization. *arXiv preprint arXiv:1210.4871*, 2012.

[41] László Lovász. Submodular functions and convexity. In *Mathematical programming the state of the art*, pages 235–257. Springer, 1983.

[42] Piyushi Manupriya, Tarun Ram Menta, Sakethanath N. Jagarlapudi, and Vineeth N. Balasubramanian. Improving attribution methods by learning submodular functions. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th*

International Conference on Artificial Intelligence and Statistics, volume 151 of *Proceedings of Machine Learning Research*, pages 2173–2190. PMLR, 28–30 Mar 2022. URL https://proceedings.mlr.press/v151/manupriya22a.html.

[43] Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018.

[44] Pitu B Mirchandani and Richard L Francis. *Discrete location theory*. 1990.

[45] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

[46] Camilo Ortiz-Astorquiza, Ivan Contreras, and Gilbert Laporte. Multi-level facility location as the maximization of a submodular set function. *European Journal of Operational Research*, 247(3):1013–1016, 2015.

[47] Chirag Pabbaraju and Prateek Jain. Learning functions over sets via permutation adversarial networks. *arXiv preprint arXiv:1907.05638*, 2019. URL https://arxiv.org/pdf/1907.05638.

[48] Thomas Powers, Rasool Fakoor, Siamak Shakeri, Abhinav Sethy, Amanjit Kainth, Abdelrahman Mohamed, and Ruhi Sarikaya. Differentiable greedy networks. *arXiv preprint arXiv:1810.12464*, 2018.

[49] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[50] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Deep learning with sets and point clouds. *arXiv preprint arXiv:1611.04500*, 2016.

[51] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *International Conference on Machine Learning*, pages 4334–4343, 2018.

[52] Hamid Rezatofighi, Roman Kaskman, Farbod T Motlagh, Qinfeng Shi, Anton Milan, Daniel Cremers, Laura Leal-Taixé, and Ian Reid. Learn to predict sets using feed-forward neural networks. *arXiv preprint arXiv:2001.11845*, 2020.

[53] Indradyumna Roy, Abir De, and Soumen Chakrabarti. Adversarial permutation guided node representations for link prediction. In *arXiv:2012.08974*, 2021.

[54] Shinsaku Sakaue. Differentiable greedy algorithm for monotone submodular maximization: Guarantees, gradient estimators, and applications. In *International Conference on Artificial Intelligence and Statistics*, pages 28–36. PMLR, 2021.

[55] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[56] Manohar Shamaiah, Siddhartha Banerjee, and Haris Vikalo. Greedy sensor selection: Leveraging submodularity. In *49th IEEE conference on decision and control (CDC)*, pages 2572–2577. IEEE, 2010.

[57] Adish Singla, Sebastian Tschiatschek, and Andreas Krause. Noisy submodular maximization via adaptive sampling with applications to crowdsourced image collection summarization. In *AAAI*, 2016.

[58] Peter Stobbe and Andreas Krause. Efficient minimization of decomposable submodular functions. *arXiv preprint arXiv:1010.5511*, 2010.

[59] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in neural information processing systems*, pages 1413–1421, 2014.

[60] Sebastian Tschiatschek, Josip Djolonga, and Andreas Krause. Learning probabilistic submodular diversity models via noise contrastive estimation. In *Artificial Intelligence and Statistics*, pages 770–779. PMLR, 2016.

[61] Sebastian Tschiatschek, Aytunc Sahin, and Andreas Krause. Differentiable submodular maximization. *arXiv preprint arXiv:1803.01785*, 2018.

[62] Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A. Osborne. On the limitations of representing functions on sets. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[63] Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks. *NeurIPS*, 2019.

[64] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *International Conference on Machine Learning*, pages 1954–1963, 2015.

[65] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. *arXiv preprint arXiv:1703.06114*, 2017.

[66] Haifeng Zhang and Yevgeniy Vorobeychik. Submodular optimization with routing constraints. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[67] Yan Zhang, Jonathon Hare, and Adam Prugel-Bennett. Deep set prediction networks. *Advances in Neural Information Processing Systems*, 32:3212–3222, 2019.

[68] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in neural information processing systems*, pages 8778–8788, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Appendix A in supplemental material
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 2
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix C in supplemental material.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code is included as part of supplemental material. Further details are in Appendix E and F in supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4, Appendix E and F in supplemental material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Table 1 and Appendix G in supplemental material.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix F in supplemental material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See Appendix F in supplemental material.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provided code as a part of supplemental material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No such information present in the data used in this paper.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]