Differentiable Submodular Maximization

Sebastian Tschiatschek[1], **Aytunc Sahin**[2], **Andreas Krause**[2],
[1] Microsoft Research Cambridge
[2] ETH Zurich
sebastian.tschiatschek@microsoft.com, aytunc.sahin@inf.ethz.ch, krausea@ethz.ch

Abstract

We consider learning of submodular functions from data. These functions are important in machine learning and have a wide range of applications, e.g. data summarization, feature selection and active learning. Despite their combinatorial nature, submodular functions can be maximized approximately with strong theoretical guarantees in polynomial time. Typically, learning the submodular function and optimization of that function are treated separately, i.e. the function is first learned using a proxy objective and subsequently maximized. In contrast, we show how to perform learning and optimization jointly. By interpreting the output of greedy maximization algorithms as distributions over sequences of items and smoothening these distributions, we obtain a differentiable objective. In this way, we can differentiate through the maximization algorithms and optimize the model to work well with the optimization algorithm. We theoretically characterize the error made by our approach, yielding insights into the tradeoff of smoothness and accuracy. We demonstrate the effectiveness of our approach for jointly learning and optimizing on synthetic maxcut data, and on a real world product recommendation application.

1 Introduction

Many important applications require the prediction of set-valued outputs, e.g. product recommendation in which the output corresponds to a set of items that should be recommended to a user, object detection in images in which the output is a set of bounding boxes [Felzenszwalb *et al.*, 2010], or extractive summarization in which the output is a set of input objects (sentences in text summarization [Lin and Bilmes, 2010], images in image collection summarization [Tschiatschek *et al.*, 2014], short scenes in video summarization [Zhang *et al.*, 2016]). This problem is particularly challenging as the size of the output space (i.e. the number of possible subsets of some ground set, e.g. the set of all possible bounding boxes) increases exponentially with the size of the ground set.

The problem of learning to predict sets is most commonly approached by either specifying suitable set functions by hand or by first learning the set functions for the task at hand from data and then performing inference using the learned set function. To enjoy strong theoretical guarantees for inference via maximization, we are interested in using set functions that are submodular. The submodular set functions are fitted to the given training data (we consider the supervised case in which the training data contains the sets we wish to predict) using, for example, large margin methods [Lin and Bilmes, 2012; Tschiatschek *et al.*, 2014], maximum likelihood estimation [Gillenwater *et al.*, 2014; Tschiatschek *et al.*, 2016] or by minimizing some kind of set-valued loss function [Dolhansky and Bilmes, 2016]. Finally, for inference, some variant of greedy algorithm is used for maximizing the learned set function.

This approach suffers from the problem that training and testing are performed differently which can lead to degraded performance. Furthermore, this approach circumvents end-to-end training of the considered functions. However, end-to-end training is desirable as it often leads to significant performance improvements as demonstrated on various domains recently [Ren *et al.*, 2015; Bahdanau *et al.*, 2014].

We resolve this problem by proposing two differentiable algorithms for maximizing non-negative submodular set functions with cardinality constraints and without constraints. These algorithms can easily be integrated into existing deep learning architectures. The algorithms define a likelihood over sets and are used for *both* training and testing. This enables end-to-end training of functions for predicting sets. Our algorithms are derived from the standard greedy algorithm [Nemhauser *et al.*, 1978] and the double greedy algorithm [Buchbinder *et al.*, 2015] which was originally proposed for maximizing positive non-monotone submodular functions and provides a $\frac{1}{2}$ approximation guarantee. Interestingly, our algorithm for unconstrained maximization also provides approximation guarantees of the form $f(X_G) \geq \frac{1}{2}f(\text{OPT}) - \epsilon(t)$, where $\epsilon(t)$ is a function of the parameter t parameterizing our algorithm, OPT is an optimal solution and X_G is the output of our algorithm.

Our main contributions are:

1. We develop differentiable algorithms for end-to-end training of deep networks with set-valued outputs.
2. We theoretically characterize the additional errors imposed by smoothing used in the derivation of our algo-

rithm for unconstrained maximization.

3. We demonstrate the excellent end-to-end learning performance of our approach on challenging real-world applications, including product recommendation.

2 Notation & Problem Setting

Notation. Let $\mathcal{V} = \{e_1, e_2, \ldots\}$ be a ground set of items. We want to perform subset selection from $\mathcal{V}$. In some parts of the paper we consider $\mathcal{V}$ to be fixed while in others $\mathcal{V}$ depends on the context. Note that we assume an arbitrary but fixed enumeration of the items in $\mathcal{V}$. Furthermore, let $f\colon 2^{\mathcal{V}} \to \mathbb{R}$ be a set function assigning a real-valued scalar to every set $S \subseteq \mathcal{V}$. The function f is *submodular* if $f(A \cup e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$ for all $e \in \mathcal{V}, A \subseteq B \subseteq \mathcal{V} \setminus \{e\}$. The function f is *monotone* if $f(A) \leq f(B)$ for all $A \subseteq B$ and *non-monotone* otherwise. We denote the gain of adding an item $e \in \mathcal{V}$ to a set of items S by $\Delta_f^+(e \mid S) = f(S \cup \{e\}) - f(S)$ and the gain of removing the item $e \in S \subseteq \mathcal{V}$ from a set S as $\Delta_f^-(e \mid S) = f(S \setminus \{e\}) - f(S)$. We will omit the subscript f whenever the function is clear from the context. For notational convenience we write $S + e$ instead of $S \cup \{e\}$ and $S - e$ instead of $S \setminus \{e\}$.

Problem setting. We consider the problem of learning an unknown submodular function $f(S \mid \mathcal{V})$ from training data. The training data is given in the form of a collection of maximizers $\mathcal{X}$ (or approximate maximizers) of $f(S \mid \mathcal{V})$ for different ground sets $\mathcal{V}$, i.e. $\mathcal{X} = \{(\mathcal{V}_1, X_1), \ldots, (\mathcal{V}_M, X_M)\}$. We assume that the ground set is not simply only an abstract set of elements but that for every element in the ground set we have additional information, e.g. an associated vector of features. For instance, in the case of image collection summarization, the ground set $\mathcal{V}_i$ corresponds to the images (each element comes in form of an actual image in form of pixel values) in an image collection and the set X_i is a human generated summary for that collection. Note that the training data can contain multiple samples for the same ground set, e.g. in image collection summarization there can be multiple human generated summaries for some particular image collection. The (approximate) maximizers in $\mathcal{X}$ either maximize $f(S \mid \mathcal{V})$ unconstrainedly, i.e. $\forall(\mathcal{V}, X) \in \mathcal{X}$ it holds that $f(X \mid \mathcal{V}) \approx \max_{S \in \mathcal{V}} f(S)$ or cardinality constrainedly, i.e. $f(X) \approx \max_{S \in \mathcal{V}, |S| \leq k} f(S \mid \mathcal{V})$ for some cardinality constraint k. Given an algorithm APPROXMAX that can approximately maximize a submodular function with or without cardinality constraints, our goal is to learn a set function $h_\theta(S \mid \mathcal{V})$ parameterized by θ such that if it is maximized with APPROXMAX, the returned solutions approximately maximize $f(S \mid \mathcal{V})$. For the image collection summarization application, maximizing $h_\theta(S \mid \mathcal{V})$ with APPROXMAX should generalize to new unseen ground sets $\mathcal{V}$, i.e. yield subsets of images that summarize an unseen image collection well, respectively. For other applications, e.g. the product recommendation application, maximizing the function h_θ with APPROXMAX should yield good solutions conditioned on the selection of a subset of items (see the experimental section for details).

Our approach—learning to greedily maximize. We develop two probabilistic approximate maximization algorithms APPROXMAX that take a function $h_\theta(\cdot|\mathcal{V})$ as input and output approximate maximizers of $h_\theta(\cdot|\mathcal{V})$. These algorithms are based on the standard greedy algorithm and the double greedy algorithm and presented in detail in the next sections. Because of the probabilistic nature of the algorithms, they induce a distribution over sets $S \subseteq \mathcal{V}$ as $P_{\text{APPROXMAX}}(S; h_\theta(\cdot|\mathcal{V})) = P(\text{APPROXMAX}(h_\theta(\cdot|\mathcal{V})) = S)$. We can thus equivalently view these probabilistic algorithms APPROXMAX as distributions over sets. To learn the parameters of the function h_θ, we can thus maximize the likelihood $\mathcal{X}$ under the induced distribution, i.e. we aim to find

$$\theta^* = \arg\max_\theta \sum_{(\mathcal{V}, X) \in \mathcal{X}} \log P_{\text{APPROXMAX}}(X; h_\theta(\cdot|\mathcal{V})). \quad (1)$$

The parameters θ^* according to the above equation intimately connect the function h_{θ^*}, the algorithm APPROXMAX, and the function f, i.e. APPROXMAX computes approximate maximizers of f when applied to h_{θ^*}. By ensuring that our developed approximate maximization algorithms have differentiable likelihoods, we can maximize (1) using gradient based optimization techniques. This enables the easy integration of our algorithms into deep learning architectures which are commonly optimized using stochastic gradient descent techniques.

3 Differentiable Unconstrained Maximization

In this section we present our probabilistic algorithm PD^{2}GREEDY for differentiable unconstrained maximization of submodular set functions. The algorithm is derived from the double greedy algorithm [Buchbinder *et al.*, 2015] and presented in Algorithm 1.

The algorithm works by iterating through the items in $\mathcal{V}$ in a fixed order. In every iteration it computes the gain a_i for adding the ith item to the set X_{i-1} and the gain b_i for removing the ith item from the set Y_{i-1}. It then compares these two gains and makes a probabilistic decision based on that comparison.

Algorithm 1 PD^{2}GREEDY: Probabilistic diff. double-greedy

Require: Function $h_\theta\colon \mathcal{V} \to \mathbb{R}_{\geq 0}$
  $X_0 \leftarrow \emptyset$
  $Y_0 \leftarrow \mathcal{V}$
  for $i = 1, \ldots, |\mathcal{V}|$ **do**
    $a_i = h_\theta(X_{i-1} + e_i) - h_\theta(X_{i-1})$ ▷ $\Delta^+(e_i \mid X_{i-1})$
    $b_i = h_\theta(Y_{i-1} - e_i) - h_\theta(Y_{i-1})$ ▷ $\Delta^-(e_i \mid Y_{i-1})$
    if $g(a_i, b_i) \geq \mathcal{U}$ **then**
      $X_i = X_{i-1} \cup \{e_i\}$
    else
      $Y_i = Y_{i-1} \setminus \{e_i\}$
    end if
  end for
  return Approximate maximizer $X_{|\mathcal{V}|}$

The deterministic and randomized version of the double greedy algorithm as presented in [Buchbinder *et al.*, 2015] can be obtained from Algorithm 1 by specific choices of the function $g(a_i, b_i)$. In particular, setting $g(a, b) = g_1(a, b) \coloneqq \mathbf{1}_{a>b}$ results in the deterministic double greedy algorithm

with a $\frac{1}{3}$ approximation guarantee for maximizing positive non-monotone submodular functions. Setting $g(a,b) = g_2(a,b) \coloneqq \frac{[a]_+}{[a]_+ + [b]_+}$, where $[x]_+ = \max\{0,x\}$, results in the randomized double greedy algorithm with a $\frac{1}{2}$ approximation guarantee for maximization in expectation.

Note that the algorithm, independent of the particular choice of $g(a_i,b_i)$, induces a distribution over subsets of $\mathcal{V}$. To specify the distribution, let x be a binary vector representing the set X in form of a one-hot encoding for the assumed fixed ordering $e_1, e_2, \ldots$ of the elements in the ground set. Then, the distribution is

$$P_{\text{PD}^2\text{GREEDY}}(X) = \prod_{i=1}^{|\mathcal{V}|} g\big(\Delta^+(e_i \mid \{e_j \mid j<i, x_j = 1\}), \Delta^-(e_i \mid \{e_j \in \mathcal{V} \mid j > i \text{ or } x_j = 1\})\big). \tag{2}$$

Obviously, this distribution depends on the order of the items which we assumed to be fixed. In practice, this order influences the maximization performance as substantiated by an example shortly.

While Equation (2) defines a distribution over sets it is hard to optimize for $g = g_1$ and $g = g_2$ because of the non-smoothness of these functions. Thus we propose the following two natural smooth alternatives for the function g:

- $g(a,b) = g_3(a,b;t) \coloneqq \sigma(t \cdot (a-b))$ is an approximation to the deterministic double greedy algorithm which becomes exact as $t \to \infty$.
- $g(a,b) = g_4(a,b;t) \coloneqq \frac{a'}{a'+b'}$, where $a' = \frac{1}{t}\log(1+\exp(t\cdot a))$ and $b' = \frac{1}{t}\log(1+\exp(t\cdot b))$ is an approximation to the randomized double greedy algorithm which becomes exact as $t \to \infty$.

Interestingly, choosing one of these smooth alternatives for the function $g(a,b)$ still results in an algorithm with constant factor approximation guarantees to the true optimum as substantiated in the following theorems.

Theorem 1. *Let $\epsilon > 0$ and $g(a,b) = g_4(a,b;t)$. For $t > \sqrt{|\mathcal{V}|\log(2)/2\epsilon}$, the output $X_{|\mathcal{V}|}$ of Algorithm 1 satisfies $f(X_{|\mathcal{V}|}) \geq \frac{1}{2}f(\text{OPT}) - \epsilon$ in expectation, where $f(\text{OPT})$ is an optimal solution.*

Similarly, for the approximation to the deterministic algorithm we have the following theorem.

Theorem 2. *Let $\epsilon > 0$ and $g(a,b) = g_3(a,b;t)$. For $t > \frac{|\mathcal{V}|}{3\epsilon}W(\frac{1}{e})$, the output $X_{|\mathcal{V}|}$ of Algorithm 1 satifies $f(X_{|\mathcal{V}|}) \geq \frac{1}{3}f(\text{OPT}) - \epsilon$ in expectation, where $f(\text{OPT})$ is an optimal solution.*

The proofs are omitted due to limited space.

Theorems 1 and 2 characterize a tradeoff between accuracy of the algorithm and smoothness of the approximation. High accuracy (small ϵ) requires low smoothness (large t) and vice versa.

Speeding up computations. Naively computing the likelihood in Equation (2) requires $4|\mathcal{V}|$ function evaluations. By keeping track of the function values $f(X_i)$ and $f(Y_i)$ this can immediately be reduced to $2|\mathcal{V}|$ function evaluations. However, for large $|\mathcal{V}|$ this may still be prohibitive. Fortunately, many submodular functions allow for fast computation of all a_i and b_i values needed for evaluating (2). For example, for the modular function $f(S) = \sum_{e\in S} s_e$, where $s_e \in \mathbb{R}$, the gains for computing $\text{PD}^2\text{GREEDY}(X)$ are $a_i = s_{e_i}$ and $b_i = -s_{e_i}$. Another example is the facility location function $f(S) = \max_{e\in S} w_e$, where $w_e \in \mathbb{R}_{\geq 0}$. In that case, a_i can be iteratively computed as $a_i = \max\{c_{i-1}, w_{e_i}\}$ where $c_i = \max\{c_{i-1}, w_{e_i}\}$ if $e_i \in X$ and $c_i = c_{i-1}$ otherwise (setting $c_0 = 0$ for initialization).

Non non-negative functions. Some of the functions we are learning in the experiments in Section 5 are not guaranteed to be non-negative over their whole domain. This seems problematic because the above guarantees only hold for non-negative functions. Note that any set function $\hat{f}(S)$ can be transformed into a non-negative set function $f(S)$ by defining $f(S) = \hat{f}(S) - \min_{S'\in\mathcal{V}} \hat{f}(S')$. This transformation does not affect the gains a_i and b_i computed in Algorithm 1, however it changes the values of the maxima.

Ordering of the items. The ordering of the items in the ground set influences the induced distributions over sets. Furthermore it can have a significant impact on the achieved maximization performance, although the theoretical guarantees are independent of this ordering. To see this, consider the following toy example and PD^2GREEDY with $g = g_1$: Let $\mathcal{V} = \{e_1, e_2\}$, $w_{e_1} = 2, w_{e_2} = 1$ and $f(S) = 2 + \max_{i\in S} w_i - |S|^2$. This function is non-negative submodular with $f(\emptyset) = 2, f(\{e_1\}) = 3, f(\{e_2\}) = 2, f(\mathcal{V}) = 0$. If the items in PD^2GREEDY were considered in the the order (e_1, e_2), the algorithm would return $\{e_2\}$ with a function value of 2 while it would return $\{e_1\}$ with a function value of 3 if the items were considered in the reversed order (e_2, e_1). Note that similar observations hold for all choices of g.

Connection to probabilistic submodular models. Algorithm PD^2GREEDY defines a distribution over sets and there is an interesting connection to probabilistic submodular models [Djolonga and Krause, 2014] Specifically, for the case that $f(S)$ is a modular function, $g = g_3$ and $t = \frac{1}{2}$, the induced distribution of PD^2GREEDY corresponds to that of a log-modular distribution, i.e. items e_i are contained in the output of PD^2GREEDY with probability $\sigma(f(\{e_i\}))$.

4 Diff. Cardinality Constrained Maximization

In this section we present our algorithm for differentiable cardinality constrained maximization of submodular functions. We assume that the cardinality constraint is k, i.e. $\forall_{(\mathcal{V},X)\in\mathcal{X}}\colon f(X) \approx \max_{S\subseteq\mathcal{V},|S|=k} f(S)$.

Our algorithm PGREEDY is presented in Algorithm 2. The algorithm iteratively builds up a solution of k elements by adding one element in every iteration, starting from the empty set. The item added to the interim solution in iteration i is selected randomly from all not yet selected items $\mathcal{V} \setminus X_{i-1}$, where the probability $p_{e|X_{i-1}}$ of selecting item e depends on the gain of adding e to X_{i-1}. The selection probabilities are controlled by the temperature t. Assuming no ties, for $t \to 0$, PGREEDY is equivalent to the standard greedy algorithm, deterministically selecting the element with highest marginal

gain in every iteration.
The algorithm naturally induces a differentiable distribution over sequences of items $\sigma = (e_1, \ldots, e_k)$ of length k at temperature t such that

$$P(\sigma) = \prod_{i=1}^{k} \frac{\exp(\frac{1}{t}\Delta^+_{h_\theta}(\sigma_i \mid X_{i-1}))}{\sum_{e' \in \mathcal{V} \setminus X_{i-1}} \exp(\frac{1}{t}\Delta^+_{h_\theta}(e' \mid X_{i-1}))}, \quad (3)$$

where $X_{i-1} = \{\sigma_1, \ldots, \sigma_{i-1}\}$. From this distribution, we derive the probability of a set S by summing over all sequences σ of items consistent with S, i.e.

$$P(S) = \sum_{\sigma \in \Sigma(S)} P(\sigma) \quad (4)$$

where $\Sigma(S)$ is the set of permutations of the elements in S.

Algorithm 2 PGREEDY: Probabilistic differentiable greedy

Require: Function $h_\theta \colon \mathcal{V} \to \mathbb{R}_{\geq 0}$, cardinality constraint k
  $X_0 \leftarrow \emptyset$
  for $i = 1, \ldots, k$ **do**
    $\mathcal{C} \leftarrow \mathcal{V} \setminus X_{i-1}$
    $\forall e \in \mathcal{C} \colon p_{e|X_{i-1}} \leftarrow \frac{\exp\left(\frac{1}{t}\Delta^+_{h_\theta}(e|X_{i-1})\right)}{\sum_{e' \in \mathcal{C}} \exp\left(\frac{1}{t}\Delta^+_{h_\theta}(e'|X_{i-1})\right)}$
    $e^* \leftarrow$ sample e from $\mathcal{C}$ with probability $p_{e|X_{i-1}}$
    $X_i = X_{i-1} \cup \{e\}$
  end for
  return Approximate maximizer X_k

As already briefly mentioned, the probabilities $P(S \mid \sigma)$ depend on the temperature t. For $t = 0$ there is at most one permutation for which $P(S \mid \sigma)$ is non-zero (assuming that there are no ties). In contrast, for $t \to \infty$, $P(S \mid \sigma)$ is constant for all permutations. Similarly to the case of PD^{2}GREEDY, the choice of the temperature t allows to trade-off accuracy (the standard greedy algorithm has a $(1 - 1/e)$ approximation guarantee for cardinality constraint maximization of non-negative monotone functions) and smoothness. The higher the temperature, the smoother is the induced distribution but the *further* are the made decisions from the greedy choice.

Learning with PGREEDY. Given training data $\mathcal{X}$, we can optimize the parameters of the function h_θ by maximizing the likelihood of $\mathcal{X}$ under the distribution (4). However, for large k computing the summation over $\sigma \in \Sigma(S)$ is infeasible. We propose two approximations for this case:

- If t is *small*, $P(S)$ can be accurately approximated by $P(\sigma^*(S))$ where $\sigma^*(S) = \arg\max_{\sigma \in \Sigma(S)} P(\sigma)$. However, to find σ^* we would still have to search over all possible permutations of S. Hence, we propose to approximate $\sigma^*(S)$ by the permutation $\sigma^G(S)$ induced by the greedy order of the elements in S.
- Assuming there is no clear preference on the order of the items in S, i.e. $P(\sigma_i) \approx P(\sigma_j)$ for all $\sigma_i, \sigma_j \in \Sigma(S)$, we can approximate $P(S)$ as $P(S) \approx k! P(\sigma^R)$, where σ^R is a random permutation of the items in S.

Testing. If k is known at test time, e.g. we aim to compute a summary of fixed size in the image collection summarization application, we can simply execute the standard greedy algorithm using the learned function h_θ. This returns an approximate MAP solution for $P(S)$. However, we can also generate approximate maximizers of h_θ by invoking Algorithm 2. This can be beneficial in cases where we want to propose several candidate summaries to a user to choose from.

5 Experiments

5.1 Maximum Cut

Let $G(V, E, w)$ be a weighted undirected graph, where V denotes the set of vertices, E is the set of edges and w denotes the set of non-negative edge weights $w_{ij}, \forall (i,j) \in E$. The maximum cut problem is to find a subset of nodes S such that the cut value $f(S) = \sum_{i \in S} \sum_{j \in V \setminus S} w_{ij}$ is maximized. The cut function $f(S)$ is a non-negative and non-monotone submodular function and has, for instance, been used in semi-supervised learning [Wang *et al.*, 2013].

We generate synthetic data for our maximum cut experiment as follows. We first generate n vectors $x_1, \ldots, x_n$ whose components are sampled from $\mathcal{U}{\sim}(0, 1)$ in $\mathbb{R}^d$ and arrange them in a $n \times d$ matrix $\mathbf{X}$. Using a projection matrix $\mathbf{P}$, we project each x_i to the lower dimensional space $\mathbb{R}^k$. The weights of the graph w are generated using an RBF kernel $\mathbf{K}$ on these transformed points. For the resulting graph instance $G(V, E, w)$ we compute the maximum cut $S^* \subseteq V$ using MIP (Mixed Integer Programming).

We sample $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_m$ as described above and, using the *same* projection matrix $\mathbf{P}$ and RBF kernel $\mathbf{K}$, we generate weights w^i for m graphs and find the maximum cut S^*_i by MIP for each of these graphs. From this data we compose our training set as $\mathcal{X} = \{(\mathbf{X}_1, S^*_1), \ldots, (\mathbf{X}_m, S^*_m)\}$. The test set is composed using the same strategy.

In this experiment, our aim is to learn a projection matrix $\hat{\mathbf{P}}$ such that PD^{2}GREEDY, if executed on the graph induced by the points $\hat{\mathbf{P}}\mathbf{X}_i$, returns S^*_i. Note that our goal is not necessarily to recover the original projection matrix $\mathbf{P}$—we are only interested in a projection matrix that allows us to find high value cuts through PD^{2}GREEDY. For each element of the test set, we compute the corresponding graph using $\hat{\mathbf{P}}$ and find the maximum cut using PD^{2}GREEDY and call the set as S^l_{DG}. We compute the corresponding graph using $\mathbf{P}$ and a random projection matrix, which acts as a baseline, $\mathbf{P}_r$ and find the maxcut using PD^{2}GREEDY; we refer to these cuts as S^o_{DG} and S^r_{DG} respectively. When the cut value is calculated using MIP, we call it as S^o_{MIP}. We compare the ratio of the each cut value calculated by PD^{2}GREEDY to the cut value calculated by MIP.

Results are shown in Table 1. We observe that cuts computed with PD^{2}GREEDY using the learned projection matrix outperform cuts computed via random projection matrices and the original projection matrix. The number of nodes in the graph is n, the temperature of the PD^{2}GREEDY is t and 0.3 is used as a bandwith parameter for the RBF kernel. The original points x_i are in $\mathbb{R}^{10}$ and they are projected into $\mathbb{R}^5$ using the first 5 coordinates. Our training and test set consists of 800 and 200 elements, respectively. During training and testing, we used $g_4(a, b; t)$ to define the likelihood of sets as in Equation (2). For optimization, we used Adam [Kingma and

Ba, 2014] with batch size of 20 and initial learning rate 0.01. In every case, we benefit from learning the projection matrix.

Table 1: Performance of different cuts for varying number of nodes n and temperature t in PD^{2}GREEDY. The cuts S^l_{DG} computed from PD^{2}GREEDY using the learned projection matrix outperforms cuts computed via random projection matrices and the original projection matrix. This indicates that the learned projection matrix is optimized to accommodate our probabilistic greedy algorithm.

Parameters	S^l_{DG}	S^o_{DG}	S^r_{DG}
n=20, t=1	$\mathbf{0.74 \pm 0.008}$	0.70 ± 0.010	0.61 ± 0.012
n=20, t=2	$\mathbf{0.80 \pm 0.007}$	0.76 ± 0.009	0.61 ± 0.012
n=30, t=1	$\mathbf{0.76 \pm 0.006}$	0.73 ± 0.007	0.66 ± 0.008
n=30, t=2	$\mathbf{0.80 \pm 0.006}$	0.79 ± 0.006	0.68 ± 0.007

5.2 Product recommendation

We consider the Amazon baby registry data [Gillenwater *et al.*, 2014]. The data consists of baby registry data collected from Amazon and split into several datasets according to product categories, e.g. safety, strollers, etc. The datasets contain 5,500 to 13,300 registries of users. The data is processed using 10-fold cross-validation. As for PD^{2}GREEDY the order of items matters, we permute the items according to their empirical frequencies for each dataset.

For every dataset and fold, we fit a facility location diversity (FLID) type model [Tschiatschek *et al.*, 2016], i.e.

$$f(S) = \sum_{i \in S} u_i + \underbrace{\sum_{d=1}^{D} \big(\max_{i \in S} w_{i,d} - \sum_{i \in S} w_{i,d} \big)}_{\mathrm{div}(S)},$$

where $u_i \in \mathbb{R}$ encodes the utility of product i and $w_{i,d} \in \mathbb{R}_{\geq 0}$ encodes the dth latent property of item i. The intuitive idea behind the term $\mathrm{div}(S)$ is to encode repulsive dependencies between items that all have the dth latent property.

We train FLID models by optimizing the likelihood of the training data in Equation (2) for $t = 1$ and the sigmoid approximation (FLID$^{\mathrm{D}}$) and by optimizing the likelihood in Equation (4) (FLID$^{\mathrm{G}}$) for $t = 10$. For optimization we used Adam with a batch size of 1 for FLID$^{\mathrm{D}}$ and with a batch size of 100 for FLID$^{\mathrm{G}}$. We decayed the initial learning rate of 0.01 with a decay factor of 0.9 per epoch. We trained all models for 20 epochs. The number of latent dimensions D is chosen 10 for ground sets with at most 40 items and 20 for larger ground sets.

Performance evaluation. To assess the performance of our trained models we compute the following performance measures and compare them to the performance of *modular* models (fully factorized models independently predicting the inclusion of any element in a set without considering correlations) and determinantal point process (DPPs) fitted by expectation maximization [Gillenwater *et al.*, 2014]. Let $\mathcal{D} = \{R_1, \ldots, R_N\}$ be a set of registries and let $\bar{\mathcal{D}} = \{R \in \mathcal{D} \mid |R| > 1\}$ be the set of registries in $\mathcal{D}$ with at least 2 items.

- *Relative improvement in likelihood* RLL. This measure quantifies the increase in likelihood for FLID and DPPs over a modular model. The RLL is computed as

$$\mathrm{RLL} = \frac{|\sum_{R \in \mathcal{D}} \log P(R) - \mathrm{LL}_{\mathrm{mod}}|}{\mathrm{LL}_{\mathrm{mod}}},$$

  where $P(R)$ is the probability of R for FLID/DPP and $\mathrm{LL}_{\mathrm{mod}} = \sum_{R \in \mathcal{D}} \log P_{\mathrm{modular}}(R) \log$ is the likelihood of the fully factorized model.
- *Fill-in accuracy* ACC. We test how accurately the considered models can predict items removed from a baby registry. We compute the fill-in accuracy as

$$\mathrm{ACC}(\mathcal{D}; \hat{f}) = \frac{1}{|\bar{\mathcal{D}}|} \sum_{R = \{r_1, \ldots, r_k\} \in \bar{\mathcal{D}}} \sum_{i=1}^{k} \mathbf{1}_{\hat{f}(R - r_i) = r_i}$$

  where $\hat{f}$ is the function used for prediction. For FLID and modular functions, $\hat{f}(S) = \arg\max_{e \in \mathcal{V} \setminus S} P(S + e)$. In the case of DPPs, $\hat{f}$ returns the element with largest marginal conditioned on S.
- *Mean reciprocal rank* MRR. Instead of only accounting correct predictions, this measure considers the order in which the models would predict items. Formally, MRR is defined as

$$\mathrm{MRR}(\mathcal{D}; \hat{f}) = \frac{1}{|\bar{\mathcal{D}}|} \sum_{R = \{r_1, \ldots, r_k\} \in \bar{\mathcal{D}}} \sum_{i=1}^{k} \frac{1}{r(r_i; R - r_i)},$$

  where $r(r_i; R - r_i) = k$ if the item r_i would be predicted as the kth item.

The results are shown in Table 2. We observe that FLID in most cases outperforms the modular model. The RLL of DPPs is in general higher than that of FLID$^{\mathrm{D}}$. In many cases FLID$^{\mathrm{G}}$ significantly outperforms DPPs and FLID$^{\mathrm{D}}$ in terms of fill-in accuracy and MRR.

6 Related Work

There is a growing literature on learning parameters of the set functions with deep neural nets. Traditionally, deep neural nets are used to learn parameters for a fixed size vector. [Rezatofighi *et al.*, 2016] considers learning the parameters of the likelihood of a set using deep neural nets and [Zaheer *et al.*, 2017] designs specific layers to create permutation invariant and equivariant functions for set prediction.

[Balcan and Harvey, 2011] considers learning submodular functions in a PAC-style framework and provides lower bound on their learnability. [Lin and Bilmes, 2012; Tschiatschek *et al.*, 2014] learn mixtures of submodular functions by optimizing the weights using a large margin structured prediction framework. Since only the weights, not the components, are learned, the learning process is highly dependent on the representativeness of the functions used. [Dolhansky and Bilmes, 2016] develop a new class of submodular functions, similar to deep neural nets, and learn them in a max margin setting.

Recently, there has been some applications which combine deep learning and reinforcement learning to develop heuristics for combinatorial optimization problems. [Dai *et al.*, 2017] uses graph embedding and reinforcement learning to learn heuristics for graph combinatorial optimization problems such as maximum cut.

Table 2: Performance of different models on the Amazon baby registry dataset. Performance of log-modular models fitted with maximum likelihood (modular), DPPs trained with the EM algorithm (DPP), FLID trained with PD^{2}GREEDY (FLID$^{\text{D}}$), and FLID trained with PGREEDY (FLID$^{\text{G}}$). FLID$^{\text{D}}$, FLID$^{\text{G}}$and DPPs clearly outperform the modular model in most cases. FLID$^{\text{G}}$outperforms FLID$^{\text{D}}$and DPPs in many cases in terms of fill-in accuracy and MRR. RLL is omitted for FLID$^{\text{G}}$as it would require approximation due to the intractability of (4) for large sets. A RLL of 0.00 indicates a negative likelihood improvement.

	RLL [%]		**ACC [%]**				**MRR [–]**			
Dataset	DPP	FLID$^{\text{D}}$	modular	DPP	FLID$^{\text{D}}$	FLID$^{\text{G}}$	modular	DPP	FLID$^{\text{D}}$	FLID$^{\text{G}}$
safety	11.60	**12.47**	16.06	16.20	16.67	**16.72**	29.55	30.02	30.34	**30.36**
carseats	8.77	**8.78**	13.76	14.72	15.03	**16.18**	28.75	30.14	30.56	**31.33**
strollers	8.54	**9.58**	14.42	16.79	19.69	**21.75**	27.36	29.90	32.95	**34.49**
furniture	10.53	**10.59**	15.82	16.08	15.93	**17.21**	30.11	30.73	30.35	**31.21**
health	**2.89**	1.83	9.94	10.32	10.30	**12.07**	21.42	22.31	21.87	**23.26**
bath	**2.66**	1.65	7.30	8.68	8.24	**9.90**	16.06	17.32	17.13	**18.88**
media	**2.35**	1.19	9.31	10.07	10.04	**14.31**	20.29	21.70	21.86	**26.14**
toys	**2.22**	1.04	9.85	11.65	11.59	**14.78**	21.93	23.40	23.64	**26.23**
bedding	**1.55**	0.24	17.17	17.43	16.87	**17.53**	27.59	28.04	27.59	28.01
apparel	**0.93**	0.00	**13.47**	13.30	13.31	**13.28**	20.88	21.19	21.16	**22.00**
diaper	**0.90**	0.25	10.13	10.13	10.24	**14.55**	18.03	18.56	19.10	**23.78**
gear	**1.68**	0.56	6.23	6.34	6.10	**6.46**	14.80	**15.12**	14.66	14.71
feeding	**0.17**	0.00	6.38	6.77	6.49	**9.24**	14.53	15.22	15.02	**18.00**

7 Conclusions

We considered learning of submodular functions which are used for subset selection through greedy maximization. To this end, we proposed variants of two types of greedy algorithms that allow for approximate maximization of submodular set functions such that their output can be interpreted as a distribution over sets. This distribution is differentiable, can be used within deep learning frameworks and enables gradient based learning. Furthermore, we theoretically characterized the tradeoff of smoothness and accuracy of some of the considered algorithms. We demonstrated the effectiveness of our approach on different applications, including max-cuts and product recommendation—observing good empirical performance.

References

[Bahdanau *et al.*, 2014] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[Balcan and Harvey, 2011] Maria-Florina Balcan and Nicholas J. A. Harvey. Learning submodular functions. In *Symposium on Theory of Computing (STOC)*, pages 793–802, 2011.

[Buchbinder *et al.*, 2015] Niv Buchbinder, Moran Feldman, Joseph Seffi, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM Journal on Computing*, 44(5):1384–1402, 2015.

[Dai *et al.*, 2017] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *CoRR*, abs/1704.01665, 2017.

[Djolonga and Krause, 2014] Josip Djolonga and Andreas Krause. From map to marginals: Variational inference in bayesian submodular models. In *Advances in Neural Information Processing Systems (NIPS)*, pages 244–252. 2014.

[Dolhansky and Bilmes, 2016] Brian W Dolhansky and Jeff A Bilmes. Deep submodular functions: Definitions and learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3404–3412, 2016.

[Felzenszwalb *et al.*, 2010] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *TPAMI*, 32(9):1627–1645, 2010.

[Gillenwater *et al.*, 2014] Jennifer A Gillenwater, Alex Kulesza, Emily Fox, and Ben Taskar. Expectation-maximization for learning determinantal point processes. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3149–3157. 2014.

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

[Lin and Bilmes, 2010] Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920. Association for Computational Linguistics, 2010.

[Lin and Bilmes, 2012] Hui Lin and Jeff A. Bilmes. Learning mixtures of submodular shells with application to document summarization. *CoRR*, abs/1210.4871, 2012.

[Nemhauser *et al.*, 1978] 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.

[Ren *et al.*, 2015] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 91–99. Curran Associates, Inc., 2015.

[Rezatofighi *et al.*, 2016] Seyed Hamid Rezatofighi, Anton Milan, Ehsan Abbasnejad, Anthony Dick, Ian Reid, et al. Deepsetnet: Predicting sets with deep neural networks. *arXiv preprint arXiv:1611.08998*, 2016.

[Tschiatschek *et al.*, 2014] 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 (NIPS)*, pages 1413–1421, 2014.

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

[Wang *et al.*, 2013] Jun Wang, Tony Jebara, and Shih-Fu Chang. Semi-supervised learning using greedy max-cut. *Journal of Machine Learning Research*, 14(1):771–800, 2013.

[Zaheer *et al.*, 2017] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. *CoRR*, abs/1703.06114, 2017.

[Zhang *et al.*, 2016] Ke Zhang, Wei-Lun Chao, Fei Sha, and Kristen Grauman. Video summarization with long short-term memory. In *European Conference on Computer Vision*, pages 766–782. Springer, 2016.