DiPS: Differentiable Policy for Sketching in Recommender Systems

Aritra Ghosh[1], Saayan Mitra[2], Andrew Lan[1]

[1]University of Massachusetts Amherst
[2]Adobe Research
arighosh@cs.umass.edu, smitra@adobe.com, andrewlan@cs.umass.edu

Abstract

In sequential recommender system applications, it is important to develop models that can capture users' evolving interest over time to successfully recommend future items that they are likely to interact with. For users with long histories, typical models based on recurrent neural networks tend to forget important items in the distant past. Recent works have shown that storing a small sketch of past items can improve sequential recommendation tasks. However, these works all rely on static sketching policies, i.e., heuristics to select items to keep in the sketch, which are not necessarily optimal and cannot improve over time with more training data. In this paper, we propose a differentiable policy for sketching (DiPS), a framework that learns a data-driven sketching policy in an end-to-end manner together with the recommender system model to explicitly maximize recommendation quality in the future. We also propose an approximate estimator of the gradient for optimizing the sketching algorithm parameters that is computationally efficient. We verify the effectiveness of DiPS on real-world datasets under various practical settings and show that it requires up to 50% fewer sketch items to reach the same predictive quality than existing sketching policies.

Introduction

Recommender Systems (RSs) have seen great success in matching users with items they are interested in when large-scale user-item interaction datasets are available. Early approaches in RS assume that a user's interest is static over time and uses a RS model to compute their latent *interest* state from historical interactions and predict their ratings on future items (He et al. 2017; Xue et al. 2017). Sequential RS (SRS) is an emerging research topic on how to effectively capture user preference changes over time. The general idea is to keep track of a user's latent interest state *over time*, e.g., using a recurrent neural network (Wu et al. 2017; Chung et al. 2014; Belletti, Chen, and Chi 2019). Since recurrent neural networks are prone to forget interactions in the distant past, in many practical scenarios, sequential RS approaches prioritize on making decisions according to a user's recent interactions in the current *session* (Hidasi et al. 2015; Hidasi and Karatzoglou 2018). However, these approaches are not adept at capturing a user's static interests from their history,

which can also be critical to future recommendations. Recently, (Wang et al. 2018; Guo et al. 2019) found that storing a small set of historical items is highly beneficial in session-based SRS. Therefore, the sketching policy, i.e., how to select *which* items to keep in the sketch, is key to the effectiveness of SRS approaches.

The sketching policy plays an important role in real-world SRS applications to reduce memory consumption and has been extensively studied in the context of problems such as moment finding, k-minimum value, and distinct element counting with probabilistic guarantees (Cormode 2011). Various data structures can be used for sketching (Cormode 2011); in this paper, we will focus on sample-based sketching as used in previous RS literature (Wang et al. 2018; Guo et al. 2019) where each sketch item is a past user-item interaction. This setting is related to problems such as data summarization and coreset construction where influence score and item hardness are often used to select representative samples (Aljundi, Kelchtermans, and Tuytelaars 2019; Koh and Liang 2017). Another related problem is active learning where the label uncertainty of future items is often used to select the next item to query (Settles 2011). A common theme for all these methods is that they define a measure of *informativeness* to select the most informative item(s).

The signature of RS applications is that items come in a *streaming* fashion (Chang et al. 2017); thus, at each time step, we observe a single item, decide whether to store it in the sketch, and decide which item to remove from the sketch if the current item is stored. This streaming nature is in stark contrast to the aforementioned problems where one often have access to the full set of items available to select from. This "one-pass" sketching setup is more challenging than the sketching setup in other problems. In practice, simple sketching strategies such as uniform reservoir sampling are often adopted in RS applications (Guo et al. 2019; Wang et al. 2018). These approaches usually keep a reservoir of random historical items to compute the gradient required for model updates.

Although static sketching policies often work well, two major limitations hinder their further development. First, these methods define a heuristic informativeness measure to select items to add or remove from the sketch that is *not optimized* on the real objective of RS: predictive quality of the user's interaction with future items. Second, these informativeness

metrics are *static* and cannot exploit abundant information that can be extracted from larger and larger training datasets that are made available in recent years. Recently, there have been approaches for differentiable sample selection policy learning in data streams in other application domains (Ghosh and Lan 2021); however, these approaches are only applicable when the prediction objective is computed on a static set of known items. In contrast, in streaming RS sketching, items included in the prediction task are constantly changing.

Contributions. In this paper, we propose **DiPS**, a **Di**fferentiable **P**olicy for **S**ketching framework to *learn* a sketching policy that optimizes the performance on the final recommendation tasks in SRS. The sketching policy is learned in an end-to-end manner together with the base RS model; at each time step, the policy takes the past sketch (of K items) and the recent item (or items) and produces a new sketch (of K items) for use in subsequent time steps. We make three key contributions:

First, we formulate the sketch update and recommendation tasks as a bi-level optimization problem (Franceschi et al. 2018) with a learnable sketching policy. In the outer-level optimization problem, we learn both the base RS model and the sketching policy by explicitly maximizing predictive quality on future recommendations. In the inner-level optimization problem, we adapt the base RS model for the user using the items in the current sketch. The sketching policy is learned in a fully differentiable manner with the sole objective of maximizing performance on future time steps.

Second, we propose an approximate estimator of the true gradient of the sketching policy parameters using a separate queue module that is computationally efficient. Since at any time step, the sketch is dependent on all the prior decisions made by the sketching policy, we need to back-propagate gradient to all the previous time steps. This gradient computation requires a re-computation of the entire sketching process from the start until the current step using the *current policy parameters*, which is computationally intensive. Instead, we show that our approximation effectively alleviates these cumbersome computations.

Third, we verify the effectiveness of DiPS through extensive experiments on five real-world datasets. We observe that the learned sketching policy outperforms existing sketching policies using static informativeness metrics on future recommendation and prediction tasks, requiring up to 50% fewer sketch items to reach the same predictive quality. Our implementation will be publicly available at https://github.com/arghosh/DiPS.

Methodology

We now detail the DiPS method. We will start with notations and the generic sketching problem setup, followed by details on base RS models, the sketching policy, and how to efficiently learn the sketching policy.

Notation and Problem Setup

We use the notation $[N]$ to denote the set $\{1, \cdots, N\}$ and use shorthand notation $x_{1:t}$ for the set $\{x_1, \cdots, x_t\}$. There are a total of N users, indexed by $i \in [N]$ and M items, indexed

by $j \in [M]$. For notation simplicity, we will only discuss the sketching process for a single user, with a total of T discrete time steps, i.e., interactions, indexed by t.

We consider two commonly studied RS settings. In the **explicit** RS setting, for a user, the sequence of interactions is denoted as $[(x_1, r_{x_1}), \cdots, (x_T, r_{x_T})]$; each element in the sequence is an item-rating pair denoted as a tuple, where $x_t \in [M]$ is the tth item they interacted with and r_{x_t} is the *rating* they gave to the item x_t. The sketching policy keeps a sketch of K pairs. Therefore, the sketch at time t is denoted as $\mathcal{S}_t = \{(x_{(1)}, r_{x_{(1)}}), \cdots (x_{(K)}, r_{x_{(K)}})\}$ where $x_{(k)} \in x_{1:t}$ and $k \in [K]$; for the first K time steps, the sketch $\mathcal{S}_t$ contains all the past history. Our goal is to predict their (real/binary/categorical-valued) rating on the item they interact with at the next time step, $r_{x_{t+1}}$, using the sketch $\mathcal{S}_t$, *given* that we know which item they interact with next. In the **implicit** RS setting, for a user, the sequence of interactions is denoted as $[x_1, \cdots, x_T]$ where $x_t \in [M]$ is the item they interacted with at time t. There are no explicit ratings; items the user interacts with are considered positively rated while items that the user does not interact with are considered to be negatively rated. The sketching policy keeps a sketch of K items. Therefore, the sketch at time t is denoted as $\mathcal{S}_t = \{x_{(1)}, \cdots x_{(K)}\}$ where $x_{(k)} \in x_{1:t}$ and $k \in [K]$. Our goal is to predict the item that they interact with next, x_{t+1}, out of the entire set of items $[M]$, using the sketch $\mathcal{S}_t$.

We consider two sketching policy updating settings depending on how frequently the sketch $\mathcal{S}_t$ is updated. In the **online setting**, we update the sketch at each time step. Specifically, given the sketch at the last time step, $\mathcal{S}_{t-1}$ and the current interaction/item (x_t, r_{x_t}) (or x_t) for the explicit (or implicit) case at time t, the sketching policy decides whether to include the current item in the sketch; if so, it also decides which item to remove from $\mathcal{S}_{t-1}$ to keep the size of the sketch fixed, arriving at the sketch for the current time step, $\mathcal{S}_t$. We use $\hat{\mathcal{S}}_t = \mathcal{S}_{t-1} \cup \{(x_t, r_{x_t})\}$ (or $\{x_t\}$) to denote the *intermediate* sketch of $K + 1$ items and the sketching policy decides which *single* item to remove from $\hat{\mathcal{S}}_t$ to get the new sketch $\mathcal{S}_t$. In the **batch setting**, we update the sketch once every τ time steps; setting $\tau = 1$ results in the online setting. Our method is equally applicable to the case of varying update time periods $\tau_1, \tau_2, \cdots$; for notation simplicity, we will only detail the case of a fixed time period τ in this paper. Specifically, given the sketch at the last time step, $\mathcal{S}_t$, and a batch of current interactions/items $[(x_{t+1}, r_{x_{t+1}}), \cdots, (x_{t+\tau}, r_{x_{t+\tau}})]$ (or $[x_{t+1}, \cdots, x_{t+\tau}]$) for the explicit (or implicit) case at time $t + \tau$, the sketching policy decides whether to include the current items in the sketch and in that case which items to remove, arriving at the sketch for the current time step $\mathcal{S}_{t+\tau}$. Similarly, we use $\hat{\mathcal{S}}_{t+\tau} = \mathcal{S}_t \cup \{(x_{t+1}, r_{x_{t+1}}), \cdots, (x_{t+\tau}, r_{x_{t+\tau}})\}$ (or $\{x_{t+1}, \cdots, x_{t+\tau}\}$) to denote the intermediate sketch of $K + \tau$ items and the sketching policy decides which τ items to remove from $\hat{\mathcal{S}}_{t+\tau}$ to get the new sketch $\mathcal{S}_{t+\tau}$.

Sketching Objective

We solve the following bilevel optimization problem (for one user only for notation simplicity) (Franceschi et al. 2018):

$$\underset{\Theta,\Phi}{\text{minimize}} \sum_{t=0}^{T-1} \ell(r_{x_{t+1}}, g(x_{t+1}; \theta_t^*(\Theta,\Phi))) \triangleq \sum_{t=0}^{T-1} \ell_{t+1}(\theta_t^*) \quad (1)$$

$$\text{s.t. } \theta_t^* = \underset{\theta_t}{\text{argmin}} \sum_{k=1}^{K} \ell(r_{x_{(k)}}, g(x_{(k)}; \theta_t)) + \mathcal{R}(\theta_t; \Theta) \triangleq \mathcal{L}(\mathcal{S}_t; \theta_t)$$
$$(2)$$

$$\text{where } \mathcal{S}_{t+1:t+\tau} = \pi(\mathcal{S}_t, (x_{t+1:t+\tau}, r_{t+1:t+\tau}); \Phi). \quad (3)$$

Here, Θ and Φ are the global RS model and sketching policy parameters, respectively. $g(\cdot)$ is the RS model that takes an item x_t as input and predict its explicit or implicit rating (which we denote as $r_{x_t} = 1$). $\pi(\cdot)$ is the sketching policy that takes as input the sketch at the last time step, $\mathcal{S}_t$, the current items $x_{t+1:t+\tau}$, and outputs the updated sketch $\mathcal{S}_{t+\tau}$.

The outer-level optimization problem minimizes the loss, $\ell(r_{x_{t+1}}, g(x_{t+1}; \theta_t^*))$ across all users and all time steps to learn both the global RS model and the sketching policy. The inner-level optimization problem minimizes $\mathcal{L}(\mathcal{S}_t; \theta_t)$, the loss on the sketch for each user at each time step, to adapt the global RS model locally, resulting in a user, time step-specific parameter θ_t^*. $\mathcal{R}(\theta_t; \Theta)$ is a regularization term that penalizes large deviations of the local parameters from global values. Note that θ_t^* is a function of the global parameters Θ and Φ, reflected through both the regularization term in (2) and the items the sketching policy selects for the user in (3).

Recommender System Model

Since our focus in this paper is differentiable sketching policy learning, which is agnostic to the underlying base RS model, we adopt a standard neural collaborative filtering (NCF) model as the base RS model (He et al. 2017) since NCF works well with gradient-based optimization; we use NCF to compute the loss $\ell(r_{x_t}, g(x_t; \theta_t))$ in both the inner and outer optimization problems. We emphasize that our approach is model agnostic and equally applicable to any differentiable RS model; in the experiments, we also use case studies to show that a learned sketching policy under one RS model is still highly effective for another RS model.

The prediction model parameter Θ contains the embedding of a user $\Theta(u)$ and a neural network with parameter $\Theta(p)$ corresponding to the parameters of the items. For simplicity, we will use Θ to denote $\{\Theta(u), \Theta(p)\}$. For the explicit RS setting, given the local parameter θ_t and the next item x_t, we predict the rating r_t as $g(x_t; \theta_t)$. For real-valued ratings, we define a Gaussian likelihood function and use the mean-squared error loss ℓ_{mse}; for binary (or categorical) ratings, we define a logistic (softmax) likelihood function resulting in the binary (or categorical) cross-entropy loss ℓ_{bce} (or ℓ_{cce}). For the implicit RS setting, we predict the next item x_t as $g(x_t; \theta)$ among all the items $[M]$. We define a softmax function over all M items, resulting in a categorical cross-entropy loss. The number of items is often large; therefore, several alternative loss functions such as the bayesian personalized ranking loss or the Top1 loss, together with negative sampling, are often used instead (Rendle et al. 2012; Hidasi et al. 2015). We emphasize that our method is agnostic to the loss function;

for simplicity, we use the standard categorical cross-entropy ℓ_{cce} loss in our experiments.

Sketching Policy

We use a sparse vector $\mathbf{z}_t \in \{0,1\}^M$ to represent the indices of each item in the sketch $\mathcal{S}_t$ at time t, with $\mathbf{z}_{t,j} = 1$ if and only if item index j is present in the current sketch. This vector has a one-to-one correspondence with the sketch $\mathcal{S}_t$. We also use the vector $\mathbf{y} = [r_1, \cdots, r_M] \in \mathbb{R}^M$ to represent the user's ratings of all items. These ratings are real-valued under the explicit RS setting and binary-valued under the implicit RS setting.[1] The ratings on the non-interacted items do not need to be defined; the DiPS algorithm masks ratings on these items. In the online ($\tau = 1$) and batch update settings, using the intermediate sketch we defined above, at time $t + \tau$, we have

$$\hat{\mathbf{z}}_{t+\tau} = \mathbf{z}_t + \sum_{j=t+1}^{t+\tau} \mathbf{e}_{x_j},$$

where $\mathbf{e}_{x_j} \in \{0,1\}^M$ represents the unit vector with a 1 only at index x_j and 0 at all other indices.

The sketching policy π only has access to items in the intermediate sketch. Therefore, we can represent this rating information using the vector $\hat{\mathbf{z}}_{t+\tau} \odot \mathbf{y} \in \mathbb{R}^M$ where $\odot$ denotes element-wise multiplication. The policy $\pi(\hat{\mathbf{z}}_{t+\tau}, \mathbf{y}; \Phi)$ updates the intermediate sketch $\hat{\mathcal{S}}_{t+\tau}$ to $\mathcal{S}_{t+\tau}$. In particular, it outputs a sparse vector $\mathbf{w}_{t+\tau} \in \{0,1\}^M$ that indicates whether each item in the sketch should be kept or removed. In the online setting, the policy outputs the item index to *remove*, $\mathbf{w}_{t+1} \in \{0,1\}^M \cap \Delta^{M-1}$, where Δ^{M-1} is the probability simplex. In the batch setting, the policy outputs the K item indices to *keep*, $\mathbf{w}_{t+\tau} \in \{0,1\}^M \cap \{\mathbf{w} : 1^T\mathbf{w} = K\}$.

The sketching policy π computes a score for each item that is in the intermediate sketch $\hat{\mathbf{z}}_{t+\tau}$ using a neural network $f(\cdot)$ with the observed ratings as $f(\hat{\mathbf{z}}_{t+1}, \mathbf{y}; \Phi) = f(\hat{\mathbf{z}}_{t+1} \odot \mathbf{y}; \Phi)$. In the online setting, we use the softmax distribution $\sigma(f(\hat{\mathbf{z}}_{t+1} \odot \mathbf{y}; \Phi))$ to select the item to remove, $\mathbf{w}_{t+1}(\sigma(f(\hat{\mathbf{z}}_{t+1} \odot \mathbf{y}; \Phi)))$. We can do this either in a deterministic way by selecting the item with the highest score or in a stochastic way by sampling from the softmax probability distribution. The item indices included in the updated sketch $\mathcal{S}_{t+1}$ are then computed as

$$\mathbf{z}_{t+1} = \mathbf{z}_t + \mathbf{e}_{x_{t+1}} - \mathbf{w}_{t+1}(\sigma(f(\hat{\mathbf{z}}_{t+1} \odot \mathbf{y}; \Phi))). \quad (4)$$

In the batch setting, we need to select K items from $K + \tau$ items. We employ the Top-K projection layer (Amos, Koltun, and Kolter 2019) defined as

$$\mu(f(\hat{\mathbf{z}}_{t+\tau} \odot \mathbf{y}; \Phi)) = \underset{0 < \mathbf{u} < 1}{\text{argmin}} -f(\hat{\mathbf{z}}_{t+\tau} \odot \mathbf{y}; \Phi)^T\mathbf{u} - H(\mathbf{u})$$
$$\text{s.t. } 1^T\mathbf{u} = K, \quad (5)$$

where $H(\mathbf{u})$ is the binary cross entropy function and $f(\hat{\mathbf{z}}_{t+\tau} \odot \mathbf{y}; \Phi)$ is the score for the M items. Similarly, we

---

[1]Our framework allows multiple interactions with the same item; for notation simplicity, we detail our method in the case where a user interacts with each item at most once.
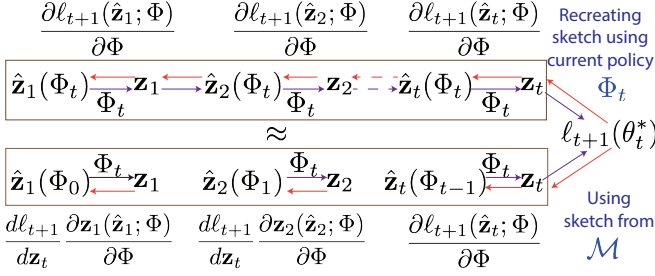
Figure 1: Top/bottom: true/approximate gradient computation at time step t. The approximate gradient calculated using intermediate sketches, obtained from past parameters $\Phi_{1:t-1}$, is close to the true gradient when the learning rate is small.

can sample the K points to keep, $\mathbf{w}_{t+\tau}(\mu(f(\hat{\mathbf{z}}_{t+\tau}, y; \Phi)))$, in either a deterministic way or a stochastic way. The sketch at time $t + \tau$ is given by

$$\mathbf{z}_{t+\tau} = \mathbf{w}_{t+\tau}(\mu(f(\hat{\mathbf{z}}_{t+\tau} \odot \mathbf{y}; \Phi))). \quad (6)$$

In both cases, the sketching policy output is only defined over items in the intermediate sketch. This constraint can be satisfied by adding $\log \hat{\mathbf{z}}_{t+\tau}$ as input to the final softmax or Top-K projection layer of the sketching policy network.

Optimization

At the inner-level, we adapt the user parameter θ_t^* from the global parameter Θ using the sketched items $\mathcal{S}_t$ at each time step. In practice, we keep item-specific neural network parameters $\Theta(p)$ fixed and adapt only the user embedding $\Theta(u)$ to minimize the loss on the K items in the sketch. Following the model agnostic meta learning approach (Finn, Abbeel, and Levine 2017), we set $\theta_t(u), \theta_t(p) \leftarrow \Theta(u), \Theta(p)$ and take a fixed number of gradient descent (GD) steps as

$$\theta_t(u) \leftarrow \theta_t(u) - \alpha \nabla_{\theta(u)} \mathcal{L}(\mathcal{S}_t; \theta)|_{\theta=\theta_t}. \quad (7)$$

A fixed number GD steps in Eq. 7 is equivalent to implicit regularization (Grant et al. 2018); thus, we do not impose any explicit regularization in the inner optimization problem. Since θ_t^* is a function of Θ, computing the gradient w.r.t. Θ in the outer optimization objective (1) requires us to compute the gradient w.r.t. the gradient in (7), i.e., the meta gradient, which can be computed using automatic differentiation (Paszke et al. 2017). Similarly, to learn the sketching policy parameters Φ, we need to compute the gradient of the outer optimization objective w.r.t. Φ through the user parameters $\theta_t^*(\Theta, \Phi)$ in (2). However, the discrete item indices to remove from the intermediate sketch are non-differentiable. Therefore, we need to develop a method to approximate this gradient, which we detail next.

Sketching Policy Optimization. The inner-level optimization in (2) uses $\mathbf{z}_t$, the vector version of the sketch $\mathcal{S}_t$, to compute the inner-level loss, which is used to adapt the user specific parameter θ_t^*. This loss is computed on all items, regardless of whether they are part of the sketch, and multiplied with the weight vector $\mathbf{z}_t$ before taking gradient steps. Therefore, we can still compute the gradient w.r.t. to the weight of all the items $\frac{d\ell_{t+1}}{d\mathbf{z}_t}$ even if their corresponding weight is

zero. We start with the online setting and denote the outer optimization objective at time $t + 1$ as $\ell_{t+1}(\theta_t^*)$. Thus, we need to compute $\frac{d\ell_{t+1}}{d\Phi}$. Note that ℓ_{t+1} is a function of θ_t^*, which is a function of Φ (from (2) and (3)) as

$$\theta_t^* = \underset{\theta_t}{\operatorname{argmin}} \sum_{j=1}^{M} \mathbf{z}_{t,j}(\Phi)\ell(r_j, g(j, \theta_t)) + \mathcal{R}(\theta_t; \Theta). \quad (8)$$

We can re-write the gradient using the chain rule as

$$\frac{d\ell_{t+1}}{d\Phi} = \frac{d\ell_{t+1}}{d\theta_t^*}\frac{d\theta_t^*}{d\Phi} = \frac{d\ell_{t+1}}{d\theta_t^*}\frac{d\theta_t^*}{d\mathbf{z}_t}\frac{d\mathbf{z}_t}{d\Phi}.$$

We also note that the sketch item indices $\mathbf{z}_t$ at time t is a function of $\{\mathbf{z}_{t-1}, \cdots, \mathbf{z}_1\}$ which are themselves a function of Φ. We can write the total derivative of $\mathbf{z}_t$ w.r.t. Φ in terms of the partial derivative as

$$\frac{d\mathbf{z}_t}{d\Phi} = \frac{\partial \mathbf{z}_t(\hat{\mathbf{z}}_t \odot \mathbf{y}; \Phi)}{\partial \Phi} + \sum_{j=1}^{t-1}\frac{\partial \mathbf{z}_j(\hat{\mathbf{z}}_j \odot \mathbf{y}; \Phi)}{\partial \Phi}\Big(\prod_{l=j}^{t-1}\frac{d\mathbf{z}_{l+1}}{d\mathbf{z}_l}\Big), \quad (9)$$

where the partial derivatives $\frac{\partial \mathbf{z}_j(\hat{\mathbf{z}}_j; \Phi)}{\partial \Phi}$ w.r.t. Φ are computed by keeping the input $\hat{\mathbf{z}}_j$ constant. The main challenge here is that in order to compute the gradient for the loss on item x_{t+1}, we need to re-generate the computation graph from $\mathbf{z}_1$ to $\mathbf{z}_t$, i.e., the entire sketching history, using the *current* policy parameter Φ (at time step t), which cannot be computed in previous time steps with past policy parameters (with multiple SGD steps in between). This regeneration is often infeasible due to its high computational cost. An alternative is to run the sketching with the current policy parameter and solve the inner optimization for each time step at once; however, that leads to enormous memory requirements for the backward gradient propagation even for a few time steps.

We propose to approximate the total derivative in (9) without recomputing the entire sketching process at every time step. For each iteration, we take a mini-batch of users (with multiple time steps for multiple interactions) for stochastic gradient descent (SGD) optimization. We represent the policy parameter Φ at time t as Φ_t in the training iteration. Note that in (9), every past $\mathbf{z}_j$ (and $\hat{\mathbf{z}}_j$) correspond to the sketch indices obtained using the current parameter Φ_t. However, we can use a queue $\mathcal{M}$ storing the intermediate sketch indices $\mathcal{M} = [\hat{\mathbf{z}}_1, \cdots, \hat{\mathbf{z}}_t]$ computed from old policy parameters, $\Phi_0, \Phi_1, \cdots, \Phi_{-1}$ respectively. If the learning rate is small enough in the SGD steps, we can assume that the past sketches stored in queue, which were computed from the old sketching policy parameters, to be close to that computed from the new parameters Φ_t. At time step t, we can then run the sketching policy with the current parameter Φ_t on the stored intermediate sketch indices $\mathcal{M}$ in parallel to obtain $\mathbf{z}_{1:t-1}$ and compute $\frac{\partial \mathbf{z}_j(\hat{\mathbf{z}}_j \odot \mathbf{y}; \Phi)}{\partial \Phi}|_{\Phi=\Phi_t}$ efficiently. We can approximate the Jacobian $\frac{d\mathbf{z}_{l+1}}{d\mathbf{z}_l}$ with the identity matrix since they are additive in (4), which does not need to be explicitly generated in (9). We can compute the gradient $\mathbf{v} = \frac{d\ell_{t+1}}{d\mathbf{z}_t}$ and obtain the vector-Jacobian product $\frac{d\ell_{t+1}}{d\mathbf{z}_t}\frac{d\mathbf{z}_t}{d\Phi}$ efficiently as

$$\frac{d\ell_{t+1}}{d\Phi} \approx \frac{\partial \ell_{t+1}(\hat{\mathbf{z}}_t; \Phi)}{\partial \Phi} + \frac{\partial}{\partial \Phi}\Big(\mathbf{v}^T \sum_{j=1}^{t-1}(\mathbf{z}_j(\hat{\mathbf{z}}_j; \Phi))\Big), \quad (10)$$

where $\mathbf{v}$ is fixed and only $\mathbf{z}_j$'s are a function of Φ for computing the partial derivatives. We note that in (10), there is no sequential dependency; all the terms can be computed in parallel. We use a fixed-size queue $\mathcal{M}$ (with size $Q \sim 50 - 100$) where we remove the oldest sketch indices when the queue gets full. We update the queue after every τ time steps. This approximate gradient computation process is visualized in Figure 1. In the supplementary material, we show that this approximate gradient remains close to the true gradient.

Since the sketch item indices $\mathbf{z}_t(\hat{\mathbf{z}}_t, \mathbf{y}, \Phi)$ are sampled from the Softmax or Top-K projection layer, they are not differentiable w.r.t the policy parameters Φ. Thus, we need to approximate the partial derivative $\frac{\partial \mathbf{z}_t(\hat{\mathbf{z}}_t, \mathbf{y}, \Phi)}{\partial \Phi}$, which we can re-write in the online setting as

$$\frac{\partial \mathbf{z}_t(\hat{\mathbf{z}}_t, \mathbf{y}; \Phi)}{\partial \Phi} = \frac{d\mathbf{z}_t}{d\mathbf{w}_t} \frac{d\mathbf{w}_t}{d\sigma(f(\cdot))} \frac{\partial \sigma(f(\hat{\mathbf{z}}_t \odot \mathbf{y}; \Phi))}{\partial \Phi},$$

where σ is the softmax layer and $\mathbf{w}_t$ contains the sketch indices after sampling in (4). We need to approximate $\frac{d\mathbf{w}_t}{\sigma(f(\cdot))}$ since they are non-differentiable; we can leverage the approximation $\mathbf{w}_t \approx \sigma(f(\cdot))$ since it holds if the item to be removed has almost all the probability mass. This approximation is known as the straight-through (ST) estimator and it is often found to have lower empirical variance than the REINFORCE gradient estimator (Williams 1992; Bengio, Léonard, and Courville 2013). In general, one can test other differentiable approximations, such as ST-Gumbel softmax estimator (Jang, Gu, and Poole 2017), for the sampling operation in (4) and (6), which we leave for future work. The final term $\frac{\partial \sigma(f(\hat{\mathbf{z}}_t \odot \mathbf{y}; \Phi))}{d\Phi}$ can be easily computed as the gradient of the softmax layer w.r.t. the policy parameters. In the batch setting, the softmax layer (σ) is replaced by the Top-K projection layer (μ) that scores top K items close to 1 and other items close to 0. We can approximate the second term as $\mathbf{w}_t \approx \mu(f(\cdot))$ when the top K items that are selected have the highest scores among all items. We can further use the KKT conditions and the implicit function theorem to compute the gradient $\frac{\partial \mu(f(\hat{\mathbf{z}}_t \odot \mathbf{y}; \Phi))}{d\Phi}$; for details, refer to (Lee et al. 2019; Amos, Koltun, and Kolter 2019).

Connection to Influence Function. In (8), we can compute the gradient of θ_t^* w.r.t. $\mathbf{z}_{t,j}, \forall j \in x_{1:t}$ using the implicit function theorem (Cook and Weisberg 1982) as

$$\frac{d\theta_t^*}{d\mathbf{z}_{t,j}} = -(\nabla_{\theta_t}^2 \mathcal{L}(\mathcal{S}_t; \theta_t))^{-1} \nabla_{\theta_t} \ell(r_j, g(j, \theta_t))|_{\theta_t = \theta_t^*}. \quad (11)$$

The gradient for loss on the next rating prediction is given by

$$\frac{d\ell_{t+1}}{d\mathbf{z}_{t,j}(\Phi)} = -(\nabla_{\theta_t} \ell(r_{x_{t+1}}; g(x_{t+1}; \theta_t)))(\nabla_{\theta_t}^2 \mathcal{L}(\mathcal{S}_t; \theta_t))^{-1}$$

$$\nabla_{\theta_t} \ell(r_j, g(j, \theta_t))|_{\theta_t = \theta_t^*} := \mathcal{I}_{t+1}(j),$$

where $\mathcal{I}_{t+1}(j)$, the influence function (Koh and Liang 2017) score of item j, computes the change in the loss on the next time step under small perturbations in the weight of this item, $\mathbf{z}_{t,j}$ in (8). Intuitively, we would want to keep items that have gradients similar to that for the future items in sketch, i.e., those that are the most informative of future recommendations. Therefore, in the online setting, the sketching policy

---

Algorithm 1: Training of DiPS

1: Initialize global parameters Θ, Φ, learning rates η (outer level), α (inner level) sketch size K, queue size Q.
2: **while** not converged **do**
3:     Randomly sample a mini-batch of n users.
4:     For each user, initialize empty queue of past sketch indices $\mathcal{M} \leftarrow \phi$, sketch $\mathcal{S}_0$ and sketch indices $\mathbf{z}_0 \in \{0, 1\}^M$, encode ratings into vector $\mathbf{y} \in \mathbb{R}^M$.
5:     **for** $t \in 1 \cdots (T - 1)$ **do**
6:         For each user, optimize θ_t^* on the sketch $\mathcal{S}_{t-1}$.
7:         Compute loss ℓ_{t+1} on item $(x_{t+1}, r_{x_{t+1}})$ using θ_t^*.
8:         Compute $\nabla_\Theta \ell_{t+1}$ update Θ: $\Theta \leftarrow \Theta - \eta \nabla_\Theta \ell_{t+1}$.
9:         Computed intermediate indices $\hat{\mathbf{z}}_t \leftarrow \mathbf{z}_{t-1} + \mathbf{e}_{x_t}$.
10:         **if** $t > K$ **then**
11:             Compute $\mathbf{z}_j$ for $j \in \{1, \cdots, t - 1\}$ in parallel from stored $\hat{\mathbf{z}}_j$ in queue $\mathcal{M}$ using $\pi(\cdot; \Phi)$.
12:             Compute $\nabla_\Phi \ell_{t+1}$ using Eq.10 and update Φ: $\Phi \leftarrow \Phi - \eta \nabla_\Phi \ell_{t+1}$.
13:             Append $\hat{\mathbf{z}}_t$ into queue $\mathcal{M}$, remove oldest if full.
14:         **end if**
15:         Compute $\mathcal{S}_t$ and $\mathbf{z}_t$ using policy π or set $\mathbf{z}_t \leftarrow \hat{\mathbf{z}}_t$.
16:     **end for**
17: **end while**

Dataset	Movielens 1M	Movielens 10M	Netflix	Book	Forusquare
Users	6K	70K	430K	22K	52K
Items	3.7K	11K	18K	24K	37K
Interactions	1M	10M	100M	1.1M	2.3M

Table 1: Dataset Statistics

will tend to select items that are the least informative (to replace from the sketch) and in the batch setting, it will tend to select items that are most informative (to keep in the sketch).

Experimental Results

Datasets and Evaluation Metric. We use five publicly available benchmark datasets: the Movielens 1M[2] and 10M[3] datasets (Harper and Konstan 2015) and the Netflix Prize dataset [4] for explicit RSs and the Amazon Book[5] and Foursquare[6] datasets for implicit RSs. The Movielens datasets contain at least 20 ratings for each user; for the Netflix dataset, we filter out users with less than 20 ratings. We use 20-core settings for the Foursquare and the Amazon book dataset; thus, all users (items) interact with at least 20 items (users). For Amazon Book dataset, we keep reviews with ratings more than 3.5 (from 1-5 scale) as the implicit positively rated items (He and McAuley 2016). The foursquare dataset contains global user check-in datasets on the Foursquare platform from Apr. 2012 to Jan. 2014 (Yang et al. 2019).

---

[2]https://grouplens.org/datasets/movielens/1m/

[3]https://grouplens.org/datasets/movielens/10m/

[4]https://www.kaggle.com/netflix-inc/netflix-prize-data,https://www.netflixprize.com/

[5]https://jmcauley.ucsd.edu/data/amazon/

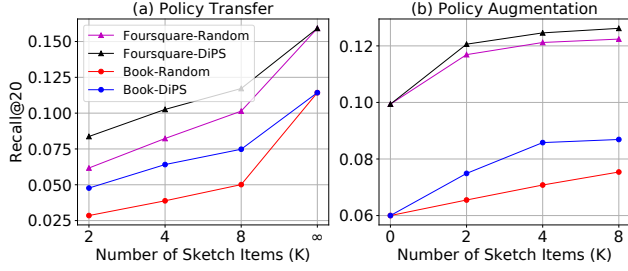[6]https://sites.google.com/site/yangdingqi/home/foursquare-dataset

Figure 2: On implicit RS datasets, (a) Recall@20 with a GRU4Rec RS model using sketching policies learned by DiPS (with NCF as the base RS model) and (b) Recall@20 with a session-based GRU4Rec model augmented with K historical sketch items.

See Table 1 for detailed statistics. For explicit RSs, we use root mean square error (RMSE) as the evaluation metric. For implicit RSs, we use Recall@20 ($= \mathbb{E}\mathbb{1}_{\text{rank} \leq K}$) as the evaluation metric where rank is computed among all possible items; we also provide additional results with Mean Reciprocal Rank (MRR)@20 as the evaluation metric where MRR@K $= \mathbb{E}\frac{\mathbb{1}_{\text{rank} \leq K}}{\text{rank}}$. We randomly split 60-20-20% of the users in the datasets into training-validation-testing sets. We run all experiments five times with different splits and report the average and standard deviation (std) numbers across all five runs.

Methods and Baselines. We compare our method, **DiPS**, against various baselines including reservoir sampling (Vitter 1985) that has been primarily used in RS applications, which keeps items with uniform probability (Guo et al. 2019; Wang et al. 2018). We dub this heuristic sketching policy as **Random**. There are several other heuristic sketching policies in streaming settings used in various applications. The **Hardest** sample heuristic keeps the hardest data point to classify in the sketch and has been highly successful in continual learning and active learning tasks (Aljundi, Kelchtermans, and Tuytelaars 2019). For binary classification, it is equivalent to uncertainty sampling. Another closely related method is to construct a *coreset* in online and batch RS settings. We use an **Influence** function-based score to construct the sketch by selecting the most representative K data points from the $K + \tau$ intermediate sketch items (Borsos, Mutnỳ, and Krause 2020). In contrast, our bi-level optimization framework explicitly minimizes the loss incurred on predicting *future items*. We also experiment with a simpler version of our method, dubbed as **DiPS@1**, where we do not keep the queue of past intermediate sketches and flow gradient only for the current items, i.e., the first term on the right-hand side of (10). We test different sketch sizes as $K \in \{2, 4, 8\}$. For the batch setting, we set the sketch update period as $\tau = 4$ to cover three cases: the update period is less than, equal to, or larger than the sketch size. Model details and parameter settings can be found in the supplementary material.

Results and Discussion

In Table 2, we list the mean RMSE and std numbers across all runs for all methods on all explicit RS datasets under the

online setting ($\tau = 1$). On all datasets, for all values of the sketch size K, DiPS significantly outperforms other methods, followed by DiPS@1. On all datasets, DiPS reaches similar predictive quality to that of static informativeness-based policies using up to 50% fewer sketch items. DiPS@1 does not perform as well as DiPS, which suggests that storing past sketch steps in queue $\mathcal{M}$ for more than one time step is beneficial to obtaining a more accurate gradient approximation and better predictive quality. We also observe that reservoir sampling slightly outperforms the Hardest and Influence heuristics on the Movielens1M and Netflix datasets while the Hardest heuristic slightly outperforms the other two on the Movielens10M dataset. Somewhat surprisingly, reservoir sampling performs well in many cases, without using any informativeness measures. We postulate that the reason behind this observation is that the Hardest and Influence heuristics operate *locally* since they make decisions only in the context of the local sketch; this restriction means that they favor items that are more informative to the most *recent* user interactions over those that are representative of longer-term history. Re-weighting these heuristics based on time difference can be beneficial and is left for future work.

In Table 2, we list the mean RMSE across all runs for all methods on all explicit RS datasets under the batch setting ($\tau = 4$). On all datasets, DiPS and DiPS@1 significantly outperform other informativeness-based policies. On the smaller datasets (Movielens 1M), DiPS@1 slightly outperforms DiPS for larger sketch sizes $K \in \{4, 8\}$. We postulate that the reason behind this observation is that the policy is not frequently updated (only once every $\tau = 4$ time steps), which reduces the benefit of more accurate gradient approximation by keeping the past sketches. Moreover, the fact that Movielens 1M is significantly smaller than the other two datasets might also contribute to this observation.

In Table 3, we list the mean Recall@20 metric across all runs for all the methods on all implicit RS datasets under the online setting ($\tau = 1$). On all datasets, DiPS and DiPS@1 significantly outperform other informativeness-based policies. On the Book dataset, DiPS with the smallest sketch size of $K = 2$ outperforms static informativeness-based policies with the largest sketch size $K = 8$ by at least 30%. On the foursquare dataset, DiPS reaches similar predictive quality to that of static policies using up to 50% fewer sketch items. We observe that DiPS@1 slightly outperforms DiPS on the smaller Book dataset. Combined with a similar observation in the explicit RS case, this observation suggests that storing past sketches is more beneficial on the larger datasets. We also observe that the Influence policy works better than other static policies on implicit RS datasets. This observation suggests that recent context might be more important under the implicit RS setting. In Table 3, we list the mean Recall@20 metric across all runs for all the methods on all implicit datasets under the batch setting ($\tau = 4$). On all datasets, DiPS significantly outperforms other informativeness-based policies while DiPS@1 slightly outperforms DiPS on the smaller Book dataset. These observations fall in line with those in the online setting. In Table 4, we list the mean and standard deviation of MRR@20 scores for all methods on all implicit datasets under the online setting ($\tau = 1$) and batch

Settings (τ)	Dataset	K	Random	Hardest	Influence	DiPS@1	DiPS
Online (1)	Movielens 1M	2	0.9701±0.0015	0.9747±0.0026	0.9747±0.0016	0.9615±0.002	**0.9543± 0.0015**
		4	0.955±0.0023	0.9606±0.0025	0.9718±0.0008	0.949±0.0014	**0.9418± 0.0018**
		8	0.9387±0.0017	0.9435±0.0024	0.9662±0.0016	0.9354±0.0017	**0.93± 0.002**
	Movielens 10M	2	0.9232±0.0011	0.9221±0.0013	0.9147±0.0015	0.9174±0.001	**0.9008± 0.0011**
		4	0.9065±0.0011	0.903±0.0014	0.9054±0.0017	0.9009±0.0011	**0.8874± 0.0011**
		8	0.8853±0.001	0.8808±0.0013	0.8948±0.0016	0.8812±0.0011	**0.8726± 0.0008**
	Netflix	2	0.9898±0.0006	0.9946±0.0008	0.9826±0.001	0.9807±0.0009	**0.9646± 0.0005**
		4	0.9708±0.0006	0.9786±0.0008	0.9726±0.0012	0.9631±0.0009	**0.9532± 0.0006**
		8	0.9474±0.0006	0.9564±0.0007	0.9598±0.0013	0.9411±0.0007	**0.9351± 0.0006**
Batch (4)	Movielens 1M	2	0.9741±0.0022	0.9965±0.0024	0.997±0.0016	0.9658±0.0023	**0.9651± 0.0027**
		4	0.9611±0.0025	0.9848±0.0021	0.9867±0.0016	**0.9537± 0.0027**	0.9592±0.0017
		8	0.9455±0.0017	0.9678±0.0017	0.9753±0.0017	**0.9418± 0.002**	0.9476±0.0018
	Movielens 10M	2	0.9221±0.0012	0.9271±0.0015	0.9127±0.0013	0.9145±0.0011	**0.9086± 0.0011**
		4	0.9055±0.0014	0.9098±0.0013	0.9016±0.0014	0.9001±0.0013	**0.8788± 0.001**
		8	0.8839±0.0014	0.888±0.0014	0.8907±0.0015	0.8801±0.0015	**0.8647± 0.001**
	Netflix	2	0.9825±0.0006	0.9925±0.0008	0.9715±0.0007	0.9732±0.0008	**0.9632± 0.0007**
		4	0.9625±0.0005	0.9766±0.0007	0.9618±0.0007	0.955±0.0005	**0.919± 0.0008**
		8	0.9387±0.0005	0.9543±0.0007	0.9497±0.0007	0.9327±0.0005	**0.909± 0.0005**

Table 2: Mean and std RMSE for all methods under the online ($\tau = 1$) and batch setting ($\tau = 4$) on all explicit RS datasets.

Settings (τ)	Dataset	K	Random	Hardest	Influence	DiPS@1	DiPS
Online (1)	Book	2	0.0672±0.0002	0.0682±0.0004	0.074±0.0005	**0.1244± 0.0017**	0.1163±0.0006
		4	0.0769±0.0006	0.0787±0.0007	0.0828±0.0004	**0.1349± 0.0005**	0.1262±0.0007
		8	0.0845±0.0005	0.0876±0.0008	0.0877±0.0005	**0.1325± 0.0007**	0.1275±0.0007
	Foursquare	2	0.1329±0.0002	0.1294±0.0003	0.1316±0.0003	0.1396±0.0001	**0.1406± 0.0003**
		4	0.1416±0.0002	0.1368±0.0003	0.141±0.0003	0.1512±0.0001	**0.1513± 0.0002**
		8	0.1508±0.0001	0.1456±0.0002	0.1489±0.0003	0.1591±0.0002	**0.1601± 0.0001**
Batch (4)	Book	2	0.0564±0.0004	0.0404±0.0003	0.068±0.0004	**0.0859± 0.001**	0.0751±0.0004
		4	0.0647±0.0006	0.0514±0.0004	0.0787±0.0004	**0.1048± 0.0008**	0.0987±0.0008
		8	0.0714±0.0005	0.0631±0.0006	0.0844±0.0006	**0.1046± 0.0006**	0.1011±0.0006
	Foursquare	2	0.1203±0.0001	0.1059±0.0003	0.1188±0.0002	0.1204±0.0003	**0.121± 0.0002**
		4	0.1283±0.0001	0.1147±0.0001	0.1277±0.0003	**0.1344± 0.0001**	0.1339±0.0001
		8	0.1362±0.0002	0.1254±0.0002	0.1355±0.0002	0.1406±0.0002	**0.1421± 0.0002**

Table 3: Mean and std Recall@20 for all methods under the online ($\tau = 1$) and batch setting ($\tau = 4$) on all implicit RS datasets.

setting ($\tau = 4$). We observe similar trends for the MRR@20 metric as the Recall@20 metric.

Policy Transfer. We perform additional experiments to show that the sketching policy learned using one base RS model, NCF in our case, can be effective even if used in conjunction with a different base RS model. In particular, we train a sequential GRU4Rec model (Hidasi et al. 2015) on the Book and foursquare datasets, where at each time step t, all history $\mathbf{x}_{1:t-1}$ is used to recommend the next item $\mathbf{x}_t$. We also train three DiPS models ($K \in \{2, 4, 8\}$) with a base NCF RS model (on the same training set of users) and only retain the learned sketching policies π. We test how this GRU4Rec model would perform when we keep a sketch of *only* K items $\mathbf{x}_{(1):(K)}$ on the test users to recommend the next item $\mathbf{x}_t$. In Figure 2(a), we plot the performance of the GRU4Rec model under different values of K for both the DiPS policy and the reservoir sampling policy. These policies are identical at $K = \infty$ when the entire history is available. We see that the DiPS policy requires up to 50% less

sketch items to reach the same recommendation quality than reservoir sampling. This observation suggests that sketching policies learned with a particular base RS model can potentially be transferred to other base RS models effectively. We note that the DiPS sketching policy exploits items that are highly predictive of future items, making them amenable to other base RS models.

Augmented Session-based SRS. We perform additional experiments to show that a few historical sketch items can effectively augment base RS models to improve session-based SRS. In particular, we split the user's history into non-overlapping sessions of four items. At each step, the model has access to items from the current session (0-3 items) and a sketch of $K \in \{2, 4, 8\}$ items from the full history. We train a modified GRU4Rec model that computes hidden states using items from the current session plus the sketch, and concatenate the two hidden states for the final prediction layer. We use uniform reservoir sampling and the DiPS sketching policy (trained with NCF) to build the sketch;

Settings (τ)	Dataset	K	Random	Hardest	Influence	DiPS@1	DiPS
Online (1)	Book	2	0.0165±0.0001	0.0171±0.0001	0.0201±0.0001	**0.0377± 0.0008**	0.0331±0.0002
		4	0.0186±0.0002	0.0196±0.0002	0.0214±0.0002	**0.0388± 0.0003**	0.0344±0.0003
		8	0.0209±0.0001	0.022±0.0002	0.0227±0.0002	**0.0363± 0.0003**	0.0337±0.0003
	Foursquare	2	0.0342±0.0001	0.0332±0.0001	0.0333±0.0002	0.0365±0.0001	**0.0373± 0.0001**
		4	0.0367±0.0	0.0352±0.0001	0.0363±0.0001	0.0403±0.0001	**0.0407± 0.0001**
		8	0.0397±0.0001	0.0376±0.0001	0.0392±0.0001	0.0435±0.0	**0.0439± 0.0001**
Batch (4)	Book	2	0.0128±0.0001	0.0094±0.0001	0.017±0.0001	**0.0217± 0.0003**	0.0185±0.0001
		4	0.0148±0.0001	0.012±0.0001	0.0194±0.0002	**0.0263± 0.0003**	0.0244±0.0002
		8	0.0164±0.0002	0.0149±0.0001	0.0206±0.0002	**0.0263± 0.0002**	0.0247±0.0002
	Foursquare	2	0.0309±0.0001	0.027±0.0001	0.0304±0.0001	0.0315±0.0	**0.0317± 0.0001**
		4	0.0333±0.0001	0.0295±0.0	0.0327±0.0	**0.0362± 0.0001**	**0.0362± 0.0001**
		8	0.0358±0.0001	0.0324±0.0001	0.0351±0.0001	0.0383±0.0001	**0.0388± 0.0001**

Table 4: Mean and std MRR@20 for all methods under the online ($\tau = 1$) and batch setting ($\tau = 4$) on all implicit RS datasets.
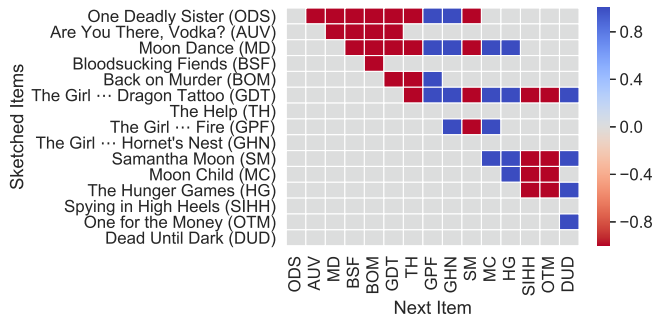

Figure 3: Visualization of the DiPS sketching policy ($K = 4$) on the Book dataset for a selected user over 15 time steps. Cell (i, j) represents whether item i is present in the sketch and used to successfully recommend item j.

we train a modified GRU4Rec model on the session items and the sketched items to recommend the next item. In Figure 2(b), we plot the performance of the modified GRU4Rec model. Note that $K = 0$ represents the standard GRU4Rec model using only the session data. We see that augmenting historical sketch items improve the performance of the session-based GRU4Rec model by more than 20% on both datasets. Moreover, the DiPS sketching policy achieves the same predictive quality as uniform reservoir sampling with 50% fewer sketch items.

Policy Visualization. In Figure 3, we plot the sketching process ($K = 4$) following the policy learned by DiPS for a selected user in the Book dataset for 15 time steps. We color-coded the columns (time steps) based on successful (blue)/unsuccessful (red) recommendations (from a total of $23,774$ distinct books), i.e., whether the actual item is included in the top-20 recommendations. This user is interested in the "Mystery/Suspense" and "Fiction" genres. The third book, "Moon Dance", is kept in the sketch between time steps 4 and 12 and used to successfully predict similar items, such as "Moon Child". The book "The Girl with the Dragon Tattoo" is kept in the sketching memory and used to successfully predict similar items "The Girl Who Played with Fire' ' and "The Girl Who Kicked the Hornet's Nest". The book "The Girl Who Kicked the Hornet's Nest" is not kept in the

sketch, possibly since it is the last book in the original "Millennium" series; its information is already well-captured in the sketch by the first book. Although we cannot successfully predict the book "Samantha Moon: The First Four Vampire", it is kept in the sketch to capture the user's interest on fiction, which is later used to successfully predict "Dead Until Dark" with the same theme. We note that although the model is not able to always successfully recommend items, the sketching policy captures item properties and adds/removes incoming items to improve future recommendations.

Conclusions and Future Work

In this paper, we developed a framework for differentiable sketching policy learning for recommender systems applications. The policy decides which past items to keep in a small sketch to explicitly maximize future predictive quality using items in the sketch. We use a bi-level optimization setup to directly learn such a sketching policy in a data-driven manner. Extensive experimental results on real-world datasets under various recommender systems settings show that our framework can sometimes significantly outperform existing static, informativeness-based sketching policies. Although side information (or metadata) often plays an important role in recommender systems, we did not use any side information in this paper. We briefly discuss how to incorporate metadata in the DiPS framework in the supplementary material. Avenues for future work include i) using more sophisticated recommender systems models that take item metadata into account, and ii) using interpretable recommender systems architecture to explicitly interpret how past items in the sketch help us predict future items (Shi et al. 2020).

Acknowledgements

References

Aljundi, R.; Kelchtermans, K.; and Tuytelaars, T. 2019. Task-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11254–11263.

Amos, B.; Koltun, V.; and Kolter, J. Z. 2019. The limited multi-label projection layer. *arXiv preprint arXiv:1906.08707*.

Belletti, F.; Chen, M.; and Chi, E. H. 2019. Quantifying long range dependence in language and user behavior to improve RNNs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1317–1327.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Borsos, Z.; Mutnỳ, M.; and Krause, A. 2020. Coresets via Bilevel Optimization for Continual Learning and Streaming. In *Advances in Neural Information Processing Systems*.

Chang, S.; Zhang, Y.; Tang, J.; Yin, D.; Chang, Y.; Hasegawa-Johnson, M. A.; and Huang, T. S. 2017. Streaming recommender systems. In *Proceedings of the 26th international conference on world wide web*, 381–389.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Cook, R. D.; and Weisberg, S. 1982. *Residuals and influence in regression*. New York: Chapman and Hall.

Cormode, G. 2011. Sketch techniques for approximate query processing. *Foundations and Trends in Databases. NOW publishers*.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning*, volume 70, 1126–1135.

Franceschi, L.; Frasconi, P.; Salzo, S.; Grazzi, R.; and Pontil, M. 2018. Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In *International Conference on Machine Learning*, 1568–1577.

Ghosh, A.; and Lan, A. 2021. BOBCAT: Bilevel Optimization-Based Computerized Adaptive Testing. In Zhou, Z.-H., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 2410–2417. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press.

Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Grant, E.; Finn, C.; Levine, S.; Darrell, T.; and Griffiths, T. 2018. Recasting gradient-based meta-learning as hierarchical bayes. In *6th International Conference on Learning Representations, ICLR 2018*.

Guo, L.; Yin, H.; Wang, Q.; Chen, T.; Zhou, A.; and Quoc Viet Hung, N. 2019. Streaming session-based recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1569–1577.

Harper, F. M.; and Konstan, J. A. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4): 1–19.

He, R.; and McAuley, J. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, 507–517.

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T.-S. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, 173–182.

Hidasi, B.; and Karatzoglou, A. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM international conference on information and knowledge management*, 843–852.

Hidasi, B.; Karatzoglou, A.; Baltrunas, L.; and Tikk, D. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparametrization with Gumbel-Softmax. In *International Conference on Learning Representations*. OpenReviews.net.

Kingma, D. P.; and Ba, J. 2015. Adam: A method for stochastic optimization. In *Proc. International Conference on Learning Representations*.

Koh, P. W.; and Liang, P. 2017. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, 1885–1894.

Lee, K.; Maji, S.; Ravichandran, A.; and Soatto, S. 2019. Meta-learning with differentiable convex optimization. In *IEEE Conference on Computer Vision and Pattern Recognition*, 10657–10665.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic Differentiation in PyTorch. In *NeurIPS Workshop on Autodiff*.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*.

Settles, B. 2011. From theories to queries: Active learning in practice. In *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, 1–18.

Shi, S.; Chen, H.; Ma, W.; Mao, J.; Zhang, M.; and Zhang, Y. 2020. Neural logic reasoning. In *Proc. ACM International Conference on Information & Knowledge Management*, 1365–1374.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958.

Vitter, J. S. 1985. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1): 37–57.

Wang, W.; Yin, H.; Huang, Z.; Wang, Q.; Du, X.; and Nguyen, Q. V. H. 2018. Streaming ranking based recommender systems. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 525–534.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4): 229–256.

Wu, C.-Y.; Ahmed, A.; Beutel, A.; Smola, A. J.; and Jing, H. 2017. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*, 495–503.

Xue, H.-J.; Dai, X.; Zhang, J.; Huang, S.; and Chen, J. 2017. Deep Matrix Factorization Models for Recommender Systems. In *IJCAI*, volume 17, 3203–3209. Melbourne, Australia.

Yang, D.; Qu, B.; Yang, J.; and Cudre-Mauroux, P. 2019. Revisiting user mobility and social relationships in lbsns: A hypergraph embedding approach. In *The world wide web conference*, 2147–2157.