Content-aware Neural Hashing for Cold-start Recommendation

Casper Hansen*
University of Copenhagen
c.hansen@di.ku.dk

Christian Hansen*
University of Copenhagen
chrh@di.ku.dk

Jakob Grue Simonsen
University of Copenhagen
simonsen@di.ku.dk

Stephen Alstrup
University of Copenhagen
s.alstrup@di.ku.dk

Christina Lioma
University of Copenhagen
c.lioma@di.ku.dk

ABSTRACT

Content-aware recommendation approaches are essential for providing meaningful recommendations for *new* (i.e., *cold-start*) items in a recommender system. We present a content-aware neural hashing-based collaborative filtering approach (NeuHash-CF), which generates binary hash codes for users and items, such that the highly efficient Hamming distance can be used for estimating user-item relevance. NeuHash-CF is modelled as an autoencoder architecture, consisting of two joint hashing components for generating user and item hash codes. Inspired from semantic hashing, the item hashing component generates a hash code directly from an item's content information (i.e., it generates cold-start and seen item hash codes in the same manner). This contrasts existing state-of-the-art models, which treat the two item cases separately. The user hash codes are generated directly based on user id, through learning a user embedding matrix. We show experimentally that NeuHash-CF significantly outperforms state-of-the-art baselines by up to 12% NDCG and 13% MRR in cold-start recommendation settings, and up to 4% in both NDCG and MRR in standard settings where all items are present while training. Our approach uses 2-4x shorter hash codes, while obtaining the same or better performance compared to the state of the art, thus consequently also enabling a notable storage reduction.

KEYWORDS

Hashing; Cold-start Recommendation; Collaborative Filtering; Content-Aware Recommendation; Autoencoders

ACM Reference Format:
Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2020. Content-aware Neural Hashing for Cold-start Recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401060

*Both authors share the first authorship

SIGIR '20, July 25–30, 2020, Virtual Event, China

ACM ISBN 978-1-4503-8016-4/20/07…$15.00
https://doi.org/10.1145/3397271.3401060

1 INTRODUCTION

Personalizing recommendations is a key factor in successful recommender systems, and thus is of large industrial and academic interest. Challenges arise both with regards to efficiency and effectiveness, especially for large-scale systems with tens to hundreds of millions of items and users.

Recommendation approaches based on collaborative filtering (CF), content-based filtering, and their combinations have been investigated extensively (see the surveys in [2, 31]), with CF based systems being one of the major methods in this area. CF based systems learn directly from either implicit (e.g., clicks) or explicit feedback (e.g., ratings), where matrix factorization approaches have traditionally worked well [5, 20]. CF learns m-dimensional user and item representations based on a factorization of the interaction-matrix between users and items, e.g., based on their click or rating history, such that the inner product can be used for computing user-item relevance. However, in the case of new unseen items (i.e., *cold-start* items), standard CF methods are unable to learn meaningful representations, and thus cannot recommend those items (and similarly for cold-start users). To handle these cases, content-aware approaches are used when additional content information is available, such as textual descriptions, and have been shown to improve upon standard CF based methods [21].

In large-scale recommendation settings, providing top-K recommendations among all existing items using an inner product is computationally costly, and thus provides a practical obstacle in employing these systems at scale. Hashing-based approaches solve this by generating *binary* user and item hash codes, such that user-item relevance can be computed using the Hamming distance (i.e., the number of bit positions where two bit strings are different). The Hamming distance has a highly efficient hardware-level implementation, and has been shown to allow for real-time retrieval among a billion items [29]. Early work on hashing-based collaborative filtering systems [17, 40, 41] learned real-valued user and item representations, which were then in a later step discretized into binary hash codes. Further work focuses on end-to-end approaches, which improve upon the two-stage approaches by reducing the discretizing error by optimizing the hash codes directly [24, 37]. Recent content-aware hashing-based approaches [22, 39] have been shown to perform well in both standard and cold-start settings, however they share the common problem of generating cold-start item hash codes differently from standard items, which we claim is unnecessary and limits their generalizability in cold-start settings.

We present a novel neural approach for content-aware hashing-based collaborative filtering (NeuHash-CF) robust to cold-start recommendation problems. NeuHash-CF consists of two joint hashing components for generating user and item hashing codes, which are connected in a variational autoencoder architecture. Inspired by semantic hashing [28], the item hashing component learns to directly map an item's content information to a hash code, while maximizing its ability to reconstruct the original content information input. The user hash codes are generated directly based on the user's id through learning a user embedding matrix, and are jointly optimized with the item hash codes to optimize the log likelihood of observing each user-item rating in the training data. Through this end-to-end trainable architecture, all item hash codes are generated in the same way, independently of whether they are seen or not during training. We experimentally compare our NeuHash-CF to state-of-the-art baselines, where we obtain significant performance improvements in cold-start recommendation settings by up to 12% NDCG and 13% MRR, and up to 4% in standard recommendation settings. Our NeuHash-CF approach uses 2-4x fewer bits, while obtaining the same or better performance than the state of the art, and notable storage reductions.

In summary, we **contribute** a novel content-aware hashing-based collaborative filtering approach (NeuHash-CF), which in contrast to existing state-of-the-art approaches generates item hash codes in a unified way (not distinguishing between standard and cold-start items).

2 RELATED WORK

The seminal work of Das et al. [9] used a Locality-Sensitive Hashing [10] scheme, called Min-Hashing, for efficiently searching Google News, where a Jaccard measure for item-sharing between users was used to generate item and user hash codes. Following this, Karatzoglou et al. [17] used matrix factorization to learn real-valued latent user and item representations, which were then mapped to binary codes using random projections. Inspired by this, Zhou and Zha [41] applied iterative quantization [11] as a way of rotating and binarizing the real-valued latent representations, which had originally been proposed for efficient hashing-based image retrieval. However, since the magnitude of the original real-valued representations are lost in the quantization, the Hamming distance between two hash codes might not correspond to the original relevance (inner product of real-valued vectors) of an item to a user. To solve this, Zhang et al. [40] imposed a constant norm constraint on the real-valued representations followed by a separate quantization.

Each of the above approaches led to improved recommendation performance, however, they can all be considered two-stage approaches, where the quantization is done as a post-processing step, rather than being part of the hash code learning procedure. Furthermore, post-processing quantization approaches have been shown to lead to large quantization errors [37], leading to the investigation of approaches learning the hash codes directly.

Next, we review (1) hashing-based approaches for recommendation with explicit feedback; (2) content-aware hashing-based recommendation approaches designed for the cold-start setting of item recommendation; and (3) the related domain of semantic hashing, which our approach is partly inspired from.

2.1 Learning to Hash Directly

Discrete Collaborative Filtering (DCF) [37] was the first approach towards learning item and user hash codes directly, rather than through a two-step approach. DCF is based on a matrix factorization formulation with additional constraints enforcing the discreteness of the generated hash codes. DCF further investigated balanced and de-correlation constraints to improve generalization by better utilizing the Hamming space. Inspired by DCF, Zhang et al. [38] proposed Discrete Personalized Ranking (DPR) as a method designed for collaborative filtering with implicit feedback (in contrast to explicit feedback in the DCF case). DPR optimized a ranking objective through AUC and regularized the hash codes using both balance and de-correlation constraints similar to DCF. While these and previous two-stage approaches have led to highly efficient and improved recommendations, they are still inherently constrained by the limited representational ability of binary codes (in contrast to real-valued representations). To this end, Compositional Coding for Collaborative Filtering (CCCF) [24] was proposed as a hybrid approach between discrete and real-valued representations. CCCF considers each hash code as consisting of a number of blocks, each of which is associated with a learned real-valued scalar weight. The block weights are used for computing a *weighted* Hamming distance, following the intuition that not all parts of an item hash code are equally relevant for all users. While this hybrid approach led to improved performance, it has a significant storage overhead (due to each hash code's block weights) and computational runtime increase, due to the weighted Hamming distance, compared to the efficient hardware-supported Hamming distance.

2.2 Content-aware Hashing

A common problem for collaborative filtering approaches, both binary and real-valued, is the cold-start setting, where a number of items have not yet been seen by users. In this setting, approaches based solely on traditional collaborative filtering cannot generate representations for the new items. Inspired by DCF, Discrete Content-aware Matrix Factorization (DCMF) [22] was the first hashing-based approach that also handled the cold-start setting. DCMF optimizes a multi-objective loss function, which most importantly learns hash codes directly for minimizing the squared rating error. Secondly, it also learns a latent representation for each content feature (e.g., each word in the content vocabulary), which is multiplied by the content features to approximate the learned hash codes, such that this can be used for generating hash codes in a cold-start setting. DCMF uses an alternating optimization strategy and, similarly to DCF, includes constraints enforcing bit balancing and de-correlation. Another approach, Discrete Deep Learning (DDL) [39] learns hash codes similarly to DCMF, through an alternating optimization strategy solving a relaxed optimization problem. However, instead of learning latent representations for each content feature to solve the cold-start problem, they train a deep belief network [16] to approximate the already learned hash codes based on the content features. This is a problem as described below.

DCMF and DDL both primarily learn hash codes not designed for cold-start settings, but then as a sub-objective learn how to map content features to new compatible hash codes for the cold-start setting. In practice, this is problematic as it corresponds to learning

cold-start item hash codes based on previously learned hash codes from standard items, which we claim is unnecessary and limits their generalizability in cold-start settings. In contrast, our proposed NeuHash-CF approach does not distinguish between between the settings for generating item hash codes, but rather always bases the item hash codes on the content features through a variational autoencoder architecture. As such, our approach can learn a better mapping from content features to hash code, since it is learned directly, as opposed to learning it in two steps by approximating the existing hash codes that have already been generated.

2.3 Semantic Hashing

The related area of Semantic Hashing [28] aims to map objects (e.g., images or text) to hash codes, such that similar objects have a short Hamming distance between them. Early work focused on two-step approaches based on learning real-valued latent representations followed by a rounding stage [34–36]. Recent work has primarily used autoencoder-based approaches, either with a secondary rounding step [7], or through direct optimization of binary codes using Bernoulli sampling and straight-through estimators for back-propagation during training [13, 14, 30]. We draw inspiration from the latter approaches in the design of the item hashing component of our approach, as substantial performance gains have previously been observed in the semantic hashing literature over rounding-based approaches.

3 HASHING-BASED COLLABORATIVE FILTERING

Collaborative Filtering learns real-valued latent user and item representations, such that the inner product between a user u and item i corresponds to the item's relevance to that specific user, where the ground truth is denoted as a user-item rating $R_{u,i}$. Hashing-based collaborative filtering learns *hash codes*, corresponding to *binary* latent representations, for users and items. We denote m-bit user and item hash codes as $z_u \in \{-1, 1\}^m$ and $z_i \in \{-1, 1\}^m$, respectively. For estimating an item's relevance to a specific user in the hashing setting, the Hamming distance is computed as opposed to the inner product, as:

$$H(z_u, z_i) = \sum_{j=1}^{m} 1_{\left[z_u^{(j)} \neq z_i^{(j)}\right]} = \text{SUM}\left(z_u \text{ XOR } z_i\right) \tag{1}$$

Thus, the Hamming distance corresponds to summing the differing bits between the codes, which can be implemented very efficiently using hardware-level bit operations through the bitwise XOR and *popcount* operations. The relation between the inner product and Hamming distance of hash codes is simply:

$$z_u^T z_i = m - 2H(z_u, z_i) \tag{2}$$

meaning it is trivial to replace real-valued user and item representations with learned hash codes in an existing recommender system.

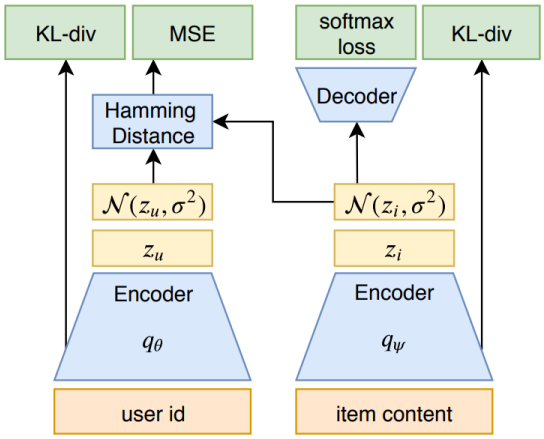

Figure 1: NeuHash-CF model overview.

3.1 Content-aware Neural Hashing-based Collaborative Filtering (NeuHash-CF)

We first give an overview of our model, Content-aware Neural Hashing-based Collaborative Filtering (NeuHash-CF), and then detail its components. NeuHash-CF consists of two joint components for generating user and item hash codes. The item hashing component learns to derive item hash codes directly from the content features associated with each item. The item hashing component has two optimization objectives: (1) to maximize the likelihood of the observed user-item ratings, and (2) the unsupervised objective of reconstructing the original content features. Through this design, all item hash codes are based on content features, thus directly generating hash codes usable for both standard and cold-start recommendation settings. This contrasts existing state-of-the-art models [22, 39] that separate how standard and cold-start item hash codes are generated. Through this choice, NeuHash-CF can generate higher quality cold-start item hash codes, but also improve the representational power of already observed items by better incorporating content features.

The user hashing component learns user hash codes, located within the same Hamming space as the item hash codes, by maximizing the likelihood of the observed user-item ratings, which is a shared objective with the item hashing component. Maximizing the likelihood of the observed user-item ratings influences the model optimization in relation to both user and item hash codes, while the unsupervised feature reconstruction loss of the item hashing component is focused only on the item hash codes. The aim of this objective combination is to ensure that the hash code distances enforce user-item relevance, but also that items with similar content have similar hash codes.

Next, we describe the architecture of our variational autoencoder (Section 3.2), followed by how users and items are encoded into hash codes (Section 3.3), decoded for obtaining a target value (Section 3.4), and lastly the formulation of the final loss function (Section 3.5). We provide a visual overview of our model in Figure 1.

3.2 Variational Autoencoder Architecture

We propose a variational autoencoder architecture for generating user and item hash codes, where we initially define the likelihood

functions of each user and item as:

$$p(u) = \prod_{i \in \mathbb{I}_u} p(R_{u,i}) \tag{3}$$

$$p(i) = p(c_i) + \prod_{u \in \mathbb{U}_i} p(R_{u,i}) \tag{4}$$

where $\mathbb{I}_u$ is the set of all items rated by user u, $\mathbb{U}_i$ is the set of all users who have rated item i, and $p(c_i)$ is the probability of observing the content of item i. We denote as $c_i \in \mathbb{R}$ the n-dimensional content feature vector (a bag-of-words representation) associated with each item, and denote the non-zero entries as $\mathbb{W}_{c_i}$. Thus, we can define the content likelihood similar to Eq. 3 and 4:

$$p(c_i) = \prod_{w \in \mathbb{W}_{c_i}} p(w). \tag{5}$$

In order to maximize the likelihood of the users and items, we need to maximize the likelihood of the observed ratings, $p(R_{u,i})$, as well as the word probabilities $p(w)$. Since they must be maximized based on the generated hash codes, we assume that $p(R_{u,i})$ is conditioned on both z_u and z_i, and that $p(w)$ is conditioned on z_i. For ease of derivation, we choose to maximize the log likelihood instead of the raw likelihoods, such that the log likelihood of the observed ratings and item content can be computed as:

$$\log p(R_{u,i}) = \log \sum_{z_i, z_u \in \{-1,1\}^m} p(R_{u,i}|z_i, z_u) p(z_i) p(z_u) \tag{6}$$

$$\log p(c_i) = \log \sum_{z_i \in \{-1,1\}^m} p(c_i|z_i) p(z_i) \tag{7}$$

where the hash codes are sampled by repeating m consecutive Bernoulli trials, which as a prior is assumed to have equal probability of sampling either 1 or -1. Thus, $p(z_i)$ and $p(z_u)$ can be computed simply as:

$$p(z) = \prod_{j=1}^{m} p^{\delta_j} (1-p)^{1-\delta_j}, \quad \delta_j = 1_{\left[z^{(j)} > 0\right]} \tag{8}$$

where $z^{(j)}$ is the j'th bit of a hash code (either user or item), and where we set $p = 0.5$ for equal sampling probability of 1 and -1. However, optimizing the log likelihoods directly is intractable, so instead we maximize their variational lower bounds [19]:

$$\log p(R_{u,i}) \geq E_{q_\psi, q_\theta}\left[\log p(R_{u,i}|z_i, z_u)\right] - \mathrm{KL}(q_\psi(z_i|i)||p(z_i)) - \mathrm{KL}(q_\theta(z_u|u)||p(z_u)) \tag{9}$$

$$\log p(c_i) \geq E_{q_\psi}\left[\log p(c_i|z_i)\right] - \mathrm{KL}(q_\psi(z_i|c_i)||p(z_i)) \tag{10}$$

where $q_\psi(z_i|i)$ and $q_\theta(z_u|u)$ are *learned* approximate posterior probability distributions (see Section 3.3), and KL is the Kullback-Leibler divergence. Intuitively, the conditional log likelihood within the expectation term can be considered a reconstruction term, which represents how well either the observed ratings or item content can be decoded from the hash codes (see Section 3.4). The KL divergence can be considered as a regularization term, by punishing large deviations from the Bernoulli distribution with equal sampling probability of 1 and -1, which is computed analytically as:

$$\mathrm{KL}(q_\psi(z_i|c_i)||p(z_i)) = q_\psi(c_i) \log \frac{q_\psi(c_i)}{p} + (1 - q_\psi(c_i)) \log \frac{1 - q_\psi(c_i)}{p} \tag{11}$$

with $p = 0.5$ for equal sampling probability. The KL divergence is computed similarly for the user hash codes using θ. Next we describe how to compute the learned approximate posterior probability distributions.

3.3 Encoder Functions

The learned approximate posterior distributions q_ψ and q_θ can be considered encoder functions for items and users, respectively, and are both modeled through a neural network formulation. Their objective is to transform users and items into m bit hash codes.

3.3.1 Item encoding. An item i is encoded based on its content c_i through multiple layers to obtain sampling probabilities for generating the hash code:

$$l_1 = \mathrm{ReLU}\left(W_1(c_i \odot w_{\text{imp}}) + b_1\right) \tag{12}$$

$$l_2 = \mathrm{ReLU}\left(W_2 l_1 + b_2\right) \tag{13}$$

where W and b are learned weights and biases, $\odot$ is elementwise multiplication, and w_{imp} is a learned importance weight for scaling the content words, which has been used similarly for semantic hashing [13]. Next, we obtain the sampling probabilities by transforming the last layer, l_2, into an m-dimensional vector:

$$q_\psi(c_i) = \sigma\left(W_3 l_2 + b_3\right) \tag{14}$$

where σ is the sigmoid function to scale the output between 0 and 1, and ψ is the set of parameters used for the item encoding. We can now sample the item hash code from a Bernoulli distribution, which can be computed for each bit as:

$$z_i^{(j)} = 2\lceil q_\psi(i)^{(j)} - \mu^{(j)} \rceil - 1 \tag{15}$$

where $\mu \in [0,1]^m$ is an m-dimensional vector with uniformly sampled values. The model is trained using randomly sampled μ vectors, since it encourages model exploration because the same item may be represented as multiple different hash codes during training. However, to produce a deterministic output for testing once the model is trained, we fix each value within μ to 0.5 instead of a randomly sampled value.

3.3.2 User encoding. The user hash codes are learned similarly to the item hash codes, however, since we do not have a user feature vector, the hash codes are learned using only the user id. Thus, the sampling probabilities are learned as:

$$q_\theta(u) = \sigma\left(E_{\text{user}} 1_u\right) \tag{16}$$

where $E_{\text{user}} \in \mathbb{R}^{|U| \times m}$ is the learned user embedding, and 1_u is a one-hot encoding of user u. Following the same approach as the item encoding, we can sample the user hash code based on $q_\theta(u)$ for each bit as:

$$z_u^{(j)} = 2\lceil q_\theta(u)^{(j)} - \mu^{(j)} \rceil - 1 \tag{17}$$

where θ is the set of parameters for user encoding. During training and testing, we use the same sampling strategy as for the item

encoding. For both users and items, we use a straight-through estimator [4] for computing the gradients for backpropgation through the sampled hash codes.

3.4 Decoder Functions

3.4.1 User-item rating decoding. The first decoding step aims to reconstruct the original user-item rating $R_{u,i}$, which corresponds to computing the conditional log likelihood of Eq. 9, i.e., $\log p(R_{u,i})|z_i, z_u)$. We first transform the user-item rating into the same range as the inner product between the hash codes:

$$\hat{R}_{u,i} = 2m\frac{R_{u,i}}{\text{max rating}} - m \tag{18}$$

Similarly to [23, 27], we assume the ratings are Gaussian distributed around their true mean for each rating value, such that we can compute the conditional log likelihood as:

$$\log p(R_{u,i}|z_i, z_u) = \log \mathcal{N}\big(\hat{R}_{u,i} - z_i^T z_u, \sigma^2\big) \tag{19}$$

where the variance σ^2 is constant, thus providing an equal weighting of all ratings. However, the exact value of the variance is irrelevant, since maximizing Eq. 19 corresponds to simply minimizing the squared error (MSE) of the mean term, i.e., $\hat{R}_{u,i} - z_i^T z_u$. Thus, maximizing the log likelihood is equivalent to minimizing the MSE, as similarly done in related work [22, 37, 39]. Lastly, note that due to the equivalence between the inner product and the Hamming distance (see Eq. 2), this directly optimizes the hash codes for the Hamming distance.

3.4.2 Item content decoding. The secondary decoding step aims to reconstruct the original content features given the generated item hash code in Eq. 10, i.e., $\log p(c_i|z_i)$. We compute this as the summation of word log likelihoods (based on Eq. 5) using a softmax:

$$\log p(c_i|z_i) = \sum_{w \in \mathbb{W}_{c_i}} \log \frac{e^{z_i^T (E_{\text{word}}(1_w \odot w_{\text{imp}})) + b_w}}{e^{\sum_{w' \in \mathbb{W}} z_i^T (E_{\text{word}}(1_{w'} \odot w_{\text{imp}})) + b_{w'}}} \tag{20}$$

where 1_w is a one-hot encoding for word w, $\mathbb{W}$ is the set of all vocabulary words of the content feature vectors, $E_{\text{word}} \in \mathbb{R}^{|\mathbb{W}| \times m}$ is a learned word embedding, b_w is a word-level bias term, and the learned importance weight w_{imp} is the same as in Eq. 12. This softmax expression is maximized when the item hash codes are able to decode the original content words.

3.4.3 Noise infusion for robustness. Previous work on semantic hashing has shown that infusing random noise into the hash codes before decoding increases robustness, and leads to more generalizable hash codes [6, 13, 30]. Thus, we apply a Gaussian noise to both user and item hash codes before decoding:

$$\text{noise}(z, \sigma^2) = z + \epsilon\sigma^2, \quad \epsilon \sim \mathcal{N}(0, I) \tag{21}$$

where variance annealing is used for decreasing the initial value of σ^2 in each training iteration.

3.5 Combined Loss Function

NeuHash-CF can be trained in an end-to-end fashion by maximising the combination of the variational lower bounds from Eq. 9 and 10, corresponding to the following loss:

$$\mathcal{L} = \mathcal{L}_{\text{rating}} + \alpha\mathcal{L}_{\text{content}} \tag{22}$$

Table 1: Dataset statistics after preprocessing such that each user has at least rated 20 items, and each item has at least been rated by 20 users.

Dataset	#users	#items	#ratings	sparsity
Yelp	27,147	20,266	1,293,247	99.765%
Amazon	35,736	38,121	1,960,674	99.856%

where $\mathcal{L}_{\text{rating}}$ corresponds to the lower bound in Eq. 9, $\mathcal{L}_{\text{content}}$ corresponds to the lower bound in Eq. 10, and α is a tunable hyper parameter to control the importance of decoding the item content.

4 EXPERIMENTAL EVALUATION

4.1 Datasets

We evaluate our approach on well-known and publicly available datasets with explicit feedback, where we follow the same preprocessing as related work [22, 33, 39] as described in the following. We disallow users to have rated the same item multiple times and use only the last rating in these cases. Due to the very high sparsity of these types of datasets, we apply a filtering to densify the dataset. We remove users who have rated fewer then 20 items, as well items that have been rated by fewer than 20 users. Since the removal of either a user or item may violate the density requirements, we apply the filtering iteratively until all users and items satisfy the requirement. The datasets are described below and summarized in Table 1:

Yelp is from the Yelp Challenge[1], which consists of user ratings and textual reviews on locations such as hotels, restaurants, and shopping centers. User ratings range between 1 (worst) to 5 (best), and most ratings are associated with a textual review.

Amazon [15] is from a collection of book reviews from Amazon[2]. Similarly to Yelp, each user rates a number of books between 1 to 5, and most are accompanied by a textual review as well.

Similarly to related work [22, 33, 39], to obtain content information related to each item, we use the textual reviews (when available) by users for an item. We filter stop words and aggregate all textual reviews for each item into a single large text, and compute the TF-IDF bag-of-words representations, where the top 8000 unique words are kept as the content vocabulary. We apply this preprocessing step separately on each dataset, thus resulting in two different vocabularies.

4.2 Experimental Design

Following Wang and Blei [33], we use two types of recommendations settings: 1) in-matrix regression for estimating the relevance of known items with existing ratings, and 2) out-of-matrix regression for estimating the relevance of cold-start items. Both of these recommendation types lead to different evaluation setups as described next.

[1] https://www.yelp.com/dataset/challenge
[2] http://jmcauley.ucsd.edu/data/amazon/

4.2.1 In-matrix regression. In-matrix regression can be considered the standard setup of all items (and users) being known at all times, and thus corresponds to the setting solvable by standard collaborative filtering. We split each user's items into a training and testing set using a 50/50 split, and use 15% of the training set as a validation set for hyper parameter tuning.

4.2.2 Out-of-matrix regression. Out-of-matrix regression is also known as a cold-start setting, where new items are to be recommended. In comparison to in-matrix regression, this task cannot be solved by standard collaborative filtering. We sort all items by their number of ratings, and then proportionally split them 50/50 into a training and testing set, such that each set has approximately the same number of items with similar number of ratings. Similarly to the in-matrix regression setting, we use 15% of the training items as a validation set for hyper parameter tuning.

4.3 Evaluation Metrics

We evaluate the effectiveness of our approach and the baselines as a ranking task with the aim of placing the most relevant (i.e., highest rated) items at the top of a ranked list. As detailed in Section 4.2, each user has a number of rated items, such that the ranked list is produced by sorting each user's testing items by their Hamming distance between the user and item hash codes. To measure the quality of the ranked list, we use Normalized Discounted Cumulative Gain (NDCG), which incorporates both ranking precision and the position of ratings. Secondly, we are interested in the first position of the item with the highest rating, as this ideally should be in the top. To this end, we compute the Mean Reciprocal Rank (MRR) of the highest ranked item with the highest given rating from the user's list of testing items.

4.4 Baselines

We compare NeuHash-CF against existing state-of-the-art content-aware hashing-based recommendation approaches, as well as hashing-based approaches that are not content-aware to highlight the benefit of including content:

DCMF Discrete Content-aware Matrix Factorization [22][3] is a content-aware matrix factorization technique, which is discretized and optimized through solving multiple mixed-integer subproblems. Similarly to our approach, its primary objective is to minimize the squared error between the rating and estimated rating based on the Hamming distance. It also learns a latent representation for each word in the text associated to each item, which is used for generating hash codes for cold-start items.

DDL Discrete Deep Learning [39][4] also uses an alternating optimizing strategy for solving multiple mixed-integer subproblems, where the primary objective is a mean squared error loss. In contrast to DCMF, DDL uses a deep belief network for generating cold-start item hash codes, which is trained by learning to map the content of known items into their hash codes generated in the first part of the approach.

DCF Discrete Collaborative Filtering [37][5] can be considered the predecessor to DCMF, but is not content-aware, which was the primary novelty of DCMF.

NeuHash-CF/no.C We include a version of our NeuHash-CF that is not content-aware, which is done by simply learning item hash codes similarly to user hash codes, thus not including any content features.

For both DCMF and DDL, hash codes for cold-start items are seen as a secondary objective, as they are generated differently from non-cold-start item hash codes. In contrast, our NeuHash-CF treats all items identically as all item hash codes are generated based on content features alone.

To provide a comparison to non-hashing based approaches, which are notably more computationally expensive for making recommendations (see Section 4.7), we also include the following baselines:

FM Factorization Machines [26] works on a concatenated n-dimensional vector of the one-hot encoded user id, one-hot encoder item id, and the content features. It learns latent vectors, as well as scalar weights and biases for each of the n dimensions. FM estimates the user-item relevance by computing a weighted sum of all non-zero entries and all interactions between non-zero entries of the concatenated vector. This results in a large amount of inner product computations and a large storage cost associated with the latent representations and scalars. We use the FastFM implementation [3][6].

MF Matrix Factorization [20] is a classic non-content-aware collaborative filtering approach, which learns real-valued item and user latent vectors, such that the inner product corresponds to the user-item relevance. MF is similar to a special case of FM without any feature interactions.

4.5 Tuning

For training our NeuHash-CF approach, we use the Adam [18] optimizer with learning rates selected from $\{0.0005, 0.0001\}$ and batch sizes from $\{500, 1000, 2000\}$, where 0.0005 and 2000 were consistently chosen. We also tune the number of encoder layers from $\{1, 2, 3\}$ and the number of neurons in each from $\{500, 1000, 2000\}$; most runs had the optimal validation performance with 2 layers and 1000 neurons. To improve robustness of the codes we added Gaussian noise before decoding the hash codes, where the variance was initially set to 1 and decreased by 0.01% after every batch. Lastly, we tune α in Eq. 22 from $\{0.001, 0.01, 0.1\}$, where 0.001 was consistently chosen. The code[7] is written in TensorFlow [1]. For all baselines, we tune the hyper parameters on the validation set as described in the original papers.

4.6 Results

The experimental comparison is summarized in Table 2 and 3 for NDCG@$\{2, 6, 10\}$ and MRR, respectively. The tables are split into in-matrix and out-of-matrix evaluation settings for both datasets, and the methods can be categorized into groups: (1) content-aware

[3] https://github.com/DefuLian/recsys/tree/master/alg/discrete/dcmf
[4] https://github.com/yixianqianzy/ddl
[5] https://github.com/hanwangzhang/Discrete-Collaborative-Filtering
[6] https://github.com/ibayer/fastFM
[7] We make the code publicly available at https://github.com/casperhansen/NeuHash-CF

Table 2: NDCG@k scores on in-matrix and out-of-matrix settings for the Amazon and Yelp datasets. Bold numbers represent the best hashing-based approach and statistically significant results compared to the best hashing-based baseline per column are marked with a star. Dashed lines correspond to not content-aware approaches in out-of-matrix setting.

	Yelp (in-matrix)									Yelp (out-of-matrix)								
	16 dim.			32 dim.			64 dim.			16 dim.			32 dim.			64 dim.		
NDCG	@2	@6	@10	@2	@6	@10	@2	@6	@10	@2	@6	@10	@2	@6	@10	@2	@6	@10
NeuHash-CF	**.662***	**.701***	**.752***	**.681***	**.718***	**.766***	**.697***	**.731***	**.776***	**.646***	**.694***	**.747***	**.687***	**.725***	**.772***	**.702***	**.737***	**.780***
DCMF	.642	.678	.733	.655	.691	.743	.670	.701	.752	.611	.647	.703	.617	.655	.709	.626	.664	.717
DDL	.636	.674	.729	.651	.686	.739	.664	.698	.749	.575	.615	.673	.579	.622	.681	.612	.646	.700
NeuHash-CF/no.C	.634	.672	.727	.655	.689	.741	.666	.699	.749	-	-	-	-	-	-	-	-	-
DCF	.639	.676	.730	.649	.685	.738	.671	.700	.750	-	-	-	-	-	-	-	-	-
MF (real-valued)	.755*	.763*	.800*	.755*	.763*	.800*	.755*	.763*	.800*	-	-	-	-	-	-	-	-	-
FM (real-valued)	.754*	.763*	.801*	.750*	.760*	.798*	.744*	.755*	.794*	.731*	.750*	.789*	.724*	.744*	.785*	.719*	.740*	.781*
	Amazon (in-matrix)									Amazon (out-of-matrix)								
	16 dim.			32 dim.			64 dim.			16 dim.			32 dim.			64 dim.		
NDCG	@2	@6	@10	@2	@6	@10	@2	@6	@10	@2	@6	@10	@2	@6	@10	@2	@6	@10
NeuHash-CF	**.759***	**.777***	**.810***	**.780***	**.798***	**.827***	**.786***	**.803***	**.831***	**.758***	**.778***	**.809***	**.769***	**.788***	**.818***	**.787***	**.804***	**.831***
DCMF	.749	.767	.800	.761	.777	.810	.773	.788	.818	.727	.748	.782	.729	.749	.784	.733	.752	.786
DDL	.734	.755	.791	.748	.768	.802	.762	.779	.811	.704	.728	.766	.705	.729	.767	.705	.727	.766
NeuHash-CF/no.C	.748	.768	.802	.760	.776	.808	.771	.785	.816	-	-	-	-	-	-	-	-	-
DCF	.745	.767	.802	.759	.776	.809	.774	.787	.818	-	-	-	-	-	-	-	-	-
MF (real-valued)	.824*	.826*	.848*	.824*	.826*	.848*	.824*	.826*	.848*	-	-	-	-	-	-	-	-	-
FM (real-valued)	.821*	.822*	.845*	.817*	.819*	.843*	.813*	.816*	.841*	.792*	.800*	.827*	.785*	.793*	.821*	.780*	.790*	.819*

Table 3: MRR scores in both in-matrix and out-of-matrix settings. Bold numbers represent the best hashing-based approach and statistically significant results compared to the best hashing-based baseline per column are marked with a star. Dashed lines correspond to not content-aware approaches in out-of-matrix setting.

	Yelp (in-matrix)			Yelp (out-of-matrix)			Amazon (in-matrix)			Amazon (out-of-matrix)		
MRR	16 dim.	32 dim.	64 dim.	16 dim.	32 dim.	64 dim.	16 dim.	32 dim.	64 dim.	16 dim.	32 dim.	64 dim.
NeuHash-CF	**.646***	**.668***	**.687***	**.628***	**.674***	**.692***	**.749***	**.770***	**.779***	**.750***	**.764***	**.782***
DCMF	.629	.644	.660	.598	.604	.612	.738	.753	.767	.719	.721	.726
DDL	.620	.638	.651	.557	.562	.604	.721	.741	.753	.696	.694	.694
NeuHash-CF/no.C	.621	.642	.656	-	-	-	.737	.752	.764	-	-	-
DCF	.626	.636	.664	-	-	-	.736	.751	.769	-	-	-
MF (real-valued)	.767*	.767*	.767*	-	-	-	.826*	.826*	.826*	-	-	-
FM (real-valued)	.761*	.756*	.750*	.730*	.722*	.717*	.824*	.821*	.815*	.792*	.784*	.780*

(NeuHash-CF, DCMF, DDL), (2) not content-aware (NeuHash-CF/no.C, DCF), (3) real-valued not content-aware (MF), and (4) real-valued content-aware (FM). For all methods, we compute hash codes (or latent representations for MF and FM) of length $m \in \{16, 32, 64\}$. We use a two-tailed paired t-test for statistical significance testing against the best performing hashing-based baseline. Statistically significant improvements, at the 0.05 level, over the best performing hashing-based baseline per column are marked with a star (*), and the best performing hashing-based approach is shown in bold.

4.6.1 In-matrix regression. In the in-matrix setting, where all items have been rated in the training data, our NeuHash-CF significantly outperforms all hashing-based baselines. On Yelp, we observe improvements in NDCG by up to 0.03, corresponding to a 4.3% improvement. On Amazon, we observe improvements in NDCG by up to 0.02, corresponding to a 2.7% improvement. Similar improvements are noted on both datasets for MRR (1.6-4.1% improvements). On all datasets and across the evaluated dimensions, NeuHash-CF performs similarly or better than state-of-the-art hashing-based approaches while using 2-4 times fewer bits, thus providing both a significant performance increase as well as a 2-4 times storage reduction. Interestingly, the performance gap between existing content-aware and not content-aware approaches is relatively small. When considering the relative performance increase of our NeuHash-CF with and without content features, we see the benefit of basing the item hash codes directly on the content. DCMF and DDL both utilize the content features for handling cold-start items, but not to the same degree for the in-matrix items, which we argue explains the primary performance increase observed for NeuHash-CF, since NeuHash-CF/no.C performs similarly to the baselines.

We also include MF and FM as real-valued baselines to better gauge the discretization gap. As expected, the real-valued approaches outperform the hashing-based approaches, however as the number of bit increases the performance difference decreases. This is to be expected, since real-valued approaches reach faster a potential representational limit, where more dimensions would not positively impact the ranking performance. In fact, for FM we observe a marginal performance drop when increasing its number of

latent dimensions, thus indicating that it is overfitting. In contrast, MF keeps the same performance (differing on far out decimals) independently of its number of latent dimensions.

4.6.2 Out-of-matrix regression. We now consider the out-of-matrix setting, corresponding to recommending cold-start items. NeuHash-CF significantly outperforms the existing state-of-the-art hashing-based baselines even more than for the in-matrix setting. On Yelp, we observe the smallest NDCG increase for 16 bit at 0.035, which is however doubled in most cases for 32 and 64 bits, corresponding to improvements of up to 12.1% gain over state-of-the-art baselines. We observe a similar trend on Amazon, where the lowest improvement of 0.027 NDCG is observed at 16 bits, but increasing the number of bits leads to consistently larger improvements of up to 7.4%. These results are also consistent with MRR, where increasing the number of bits provides increasingly larger performance increases between +5 and +13.1% on Yelp and between +4.3 and +7.7% on Amazon. In all cases, the performance of NeuHash-CF on 16 bits is even better than the best baseline at 64 bits, thus verifying the high quality of the hash codes generated by NeuHash-CF.

For the real-valued FM baseline, we observe that it outperforms ours and existing baselines at 16 and 32 dimensions, however at 64 dimensions NeuHash-CF outperforms FM on Amazon for NDCG@{6, 10} (across all dimensions). When we consider Yelp, NeuHash-CF obtains a NDCG@10 within 0.01 of FM, but worse on the other NDCG cut offs and on MRR.

4.6.3 Out-of-matrix regression with limited training data. To evaluate how the content-aware approaches generalize to the cold-start setting depending on the number of training items, we furthermore create smaller versions of the 50/50 out-of-matrix split used previously. In addition to using 50% of the data for the training set, we consider splits using 10%, 20%, 30%, and 40% as well. In all out-of-matrix settings the validation and testing sets are identical to be able to compare the impact of the training size. The results can be seen in Table 4 for 32 bit hash codes and 32 latent dimensions in FM. Similarly to before, NeuHash-CF outperforms the hashing-based baselines in all cases with similar gains as observed previously. Most approaches, except DDL on Amazon, obtain the lowest performance using 10% of the data, and more training items generally improve the performance, although at 30-50% the pace of improvement slows down significantly. This indicates that the methods have observed close to sufficiently many training items and increasing the amount may not lead to better generalizability of the cold-start hash codes. Interestingly, NeuHash-CF obtains the largest improvement going from 10% to 50% on both NDCG and MRR, indicating that it generalizes better than the baselines. In contrast, DDL does not improve on Amazon by including more training items, which indicates that its ability to generalize to cold-start items is rather limited.

4.7 Computational Efficiency

To study the high efficiency of using hash codes in comparison to real-valued vectors, we consider a setup of 100,000 users and 1,000-1,000,000 items. We randomly generate hash codes and real-valued vectors and measure the time taken to compute all Hamming distances (or inner products) from each user to all items, resulting in a total 10^8-10^{11} computations. We use a machine with a 64 bit instruction set[8], and hence generate hash codes and vectors of length 64. We report the average runtime over 10 repetitions in Figure 2, and observe a speed up of a factor 40-50 for the Hamming distance, highlighting the efficiency benefit of hashing-based approaches. For FM, its dominating cost is its large number of inner product computations, which scales quadratically in the number of non-zero content features for a given item, thus making it highly intractable in large-scale settings.

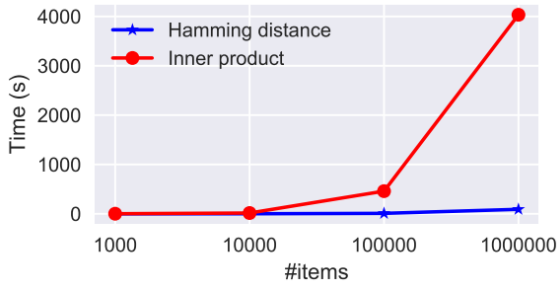

Figure 2: Computation time for all Hamming distances and inner products for 100,000 users and up to 1,000,000 items.

4.8 Impact of Average Item Popularity per User

We now look at how different user characteristics impact the performance of the methods. We first compute the average item popularity of each user's list of rated items, and then order the users in ascending order of that average. An item's popularity is computed as the number of users who have rated that specific item, and thus the average item popularity of a user is representative of their attraction to popular content. Figure 3 plots the NDCG@10 for 32 dimensional representations using a mean-smoothing window size of 1000 (i.e., each shown value is averaged based on the values within a window of 1000 users). Generally, all methods perform better for users who have a high average item popularity, where for Yelp we see a NDCG@10 difference of up to 0.25 from the lowest to highest average popularity (0.2 for Amazon). This observation can be explained by highly popular items occurring more times in the training data, such that they have a better learned representation. Additionally, the hashing-based approaches have a larger performance difference, compared to the real-valued MF and FM, which is especially due to their lower relative performance for users with a very low average item popularity (left side of plots). In the out-of-matrix setting the same trend is observed, however with our NeuHash-CF performing highly similarly to FM when excluding the users with the lowest average item popularity. We hypothesize that users with a low average item popularity have a more specialized preference, thus benefitting more from the higher representational power of real-valued representations.

4.9 Impact of Number of Items per User

We now consider how the number of items each user has rated impacts performance. We order users by their number of rated items and plot NDCG@10 for 32 bit hash codes. Figure 4 plots this in the

[8] We used an Intel Xeon CPU E5-2670

Table 4: NDCG@10 and MRR scores for 32 dimensional representations in varying cold-start scenarios with 10-50% of the items used for training. Bold numbers represent the best hashing-based approach and statistically significant results compared to the best hashing-based baseline in each column are marked with a star.

	Yelp (out-of-matrix)										Amazon (out-of-matrix)									
	10%		20%		30%		40%		50%		10%		20%		30%		40%		50%	
NDCG	@10	MRR	@10	MRR	@10	MRR	@10	MRR	@10	MRR	@10	MRR	@10	MRR	@10	MRR	@10	MRR	@10	MRR
NeuHash-CF	**.730***	**.603***	**.750***	**.634***	**.769***	**.666***	**.771***	**.668***	**.772***	**.674***	**.794***	**.727***	**.812***	**.753***	**.817***	**.761***	**.818***	**.763***	**.818***	**.764***
DCMF	.688	.572	.693	.578	.704	.593	.710	.602	.709	.604	.774	.710	.778	.712	.781	.717	.784	.720	.784	.721
DDL	.678	.556	.681	.562	.687	.572	.684	.571	.681	.562	.770	.713	.766	.689	.767	.700	.765	.693	.767	.694
FM (real-valued)	.766*	.688*	.776*	.707*	.778*	.712*	.786*	.724*	.785*	.722*	.806*	.759*	.813*	.771*	.817*	.775*	.823*	.786*	.821*	.784*

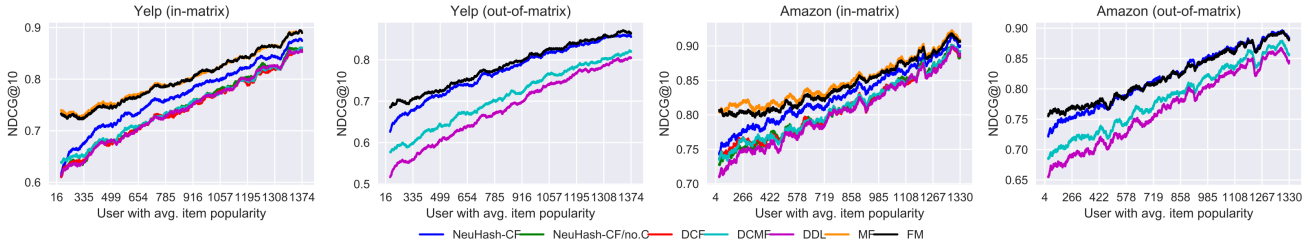

Figure 3: Impact of the average item popularity per user on NDCG@10 for 32 bit hash codes.

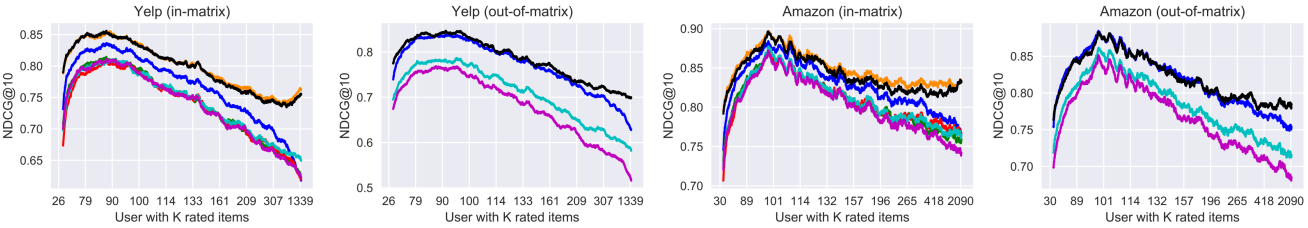

Figure 4: Impact of the number of items per user on NDCG@10 for 32 bit hash codes.

same way as in Figure 3. Generally across all methods, we observe that performance initially increases, but then drops once a user has rated close to 100 items, depending on the dataset. While the hashing-based approaches keep steadily dropping in performance, MF and FM do so at a slower pace and even increase for users with the highest number of rated items in the in-matrix setting. The plots clearly show that the largest performance difference, between the real-valued and hashing-based approaches, is for the group of users with a high number of rated items, corresponding to users with potentially the highest diversity of interests. In this setting, the limited representational power of hash codes, as opposed to real-valued representations, may not be sufficient to encode users with largely varied interests. We observe very similar trends for the out-of-matrix setting for cold-start items, although the performance gap between our NeuHash-CF and the real-valued approaches is almost entirely located among the users with a high number of rated items.

5 CONCLUSION

We presented content-aware neural hashing for collaborative filtering (NeuHash-CF), a novel hashing-based recommendation approach, which is robust to cold-start recommendation problems (i.e., the setting where the items to be recommended have not been rated previously). NeuHash-CF is a neural approach that consists of two joint components for generating user and item hash codes. The user hash codes are learned from an embedding based procedure using only the user's id, whereas the item hash codes are learned directly from associated content features (e.g., a textual item description). This contrasts existing state-of-the-art content-aware hashing-based methods [22, 39], which generate item hash codes differently depending on whether they are cold-start items or not. NeuHash-CF is formulated as a variational autoencoder architecture, where both user and item hash codes are sampled from learned Bernoulli distributions to enforce end-to-end trainability. We presented a comprehensive experimental evaluation of NeuHash-CF in both standard and cold-start settings, where NeuHash-CF outperformed state-of-the-art approaches by up to 12% NDCG and 13% MRR in cold-start recommendation (up to 4% in both NDCG and MRR in standard recommendation settings). In fact, the ranking performance of NeuHash-CF on 16 bit hash codes is better than that of 32-64 bit state-of-the-art hash codes, thus resulting in both a significant effectiveness increase, but also in a 2-4x storage reduction. Analysis of our results showed that the largest performance difference between hashing-based and real-valued approaches occurs for users interested in the least popular items, and for the group of users with the highest number of rated items. Future work includes extending the architecture to accept richer item and user representations, such as [8, 12, 25, 32].

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 265–283.

[2] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering* 6 (2005), 734–749.

[3] Immanuel Bayer. 2016. fastFM: A Library for Factorization Machines. *Journal of Machine Learning Research* 17, 184 (2016), 1–5.

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[5] James Bennett, Stan Lanning, et al. 2007. The netflix prize. In *KDD cup and workshop*, Vol. 2007. 35.

[6] Suthee Chaidaroon, Travis Ebesu, and Yi Fang. 2018. Deep Semantic Text Hashing with Weak Supervision. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1109–1112.

[7] Suthee Chaidaroon and Yi Fang. 2017. Variational deep semantic hashing for text documents. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 75–84.

[8] Felipe Soares Da Costa and Peter Dolog. 2019. Collective embedding for neural context-aware recommender systems. In *ACM Conference on Recommender Systems*. 201–209.

[9] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *ACM Conference on World Wide Web*. 271–280.

[10] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.

[11] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2012. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 12 (2012), 2916–2929.

[12] Casper Hansen, Christian Hansen, Stephen Alstrup, Jakob Grue Simonsen, and Christina Lioma. 2019. Contextually Propagated Term Weights for Document Representation. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 897–900.

[13] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2019. Unsupervised Neural Generative Semantic Hashing. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 735–744.

[14] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2020. Unsupervised Semantic Hashing with Pairwise Reconstruction. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. in press.

[15] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *ACM Conference on World Wide Web*. 507–517.

[16] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation* 18, 7 (2006), 1527–1554.

[17] Alexandros Karatzoglou, Alex Smola, and Markus Weimer. 2010. Collaborative filtering on a budget. In *International Conference on Artificial Intelligence and Statistics*. 389–396.

[18] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

[19] Diederik P Kingma and Max Welling. 2014. Auto-encoding variational bayes. In *International Conference on Learning Representations*.

[20] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (Aug. 2009), 30–37.

[21] Defu Lian, Yong Ge, Fuzheng Zhang, Nicholas Jing Yuan, Xing Xie, Tao Zhou, and Yong Rui. 2015. Content-aware collaborative filtering for location recommendation based on human mobility data. In *IEEE international conference on data mining*. 261–270.

[22] Defu Lian, Rui Liu, Yong Ge, Kai Zheng, Xing Xie, and Longbing Cao. 2017. Discrete Content-aware Matrix Factorization. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 325–334.

[23] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational autoencoders for collaborative filtering. In *ACM Conference on World Wide Web*. 689–698.

[24] Chenghao Liu, Tao Lu, Xin Wang, Zhiyong Cheng, Jianling Sun, and Steven C.H. Hoi. 2019. Compositional Coding for Collaborative Filtering. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 145–154.

[25] Ahmed Rashed, Josif Grabocka, and Lars Schmidt-Thieme. 2019. Attribute-aware non-linear co-embeddings of graph features. In *ACM Conference on Recommender Systems*. 314–321.

[26] Steffen Rendle. 2010. Factorization machines. In *International Conference on Data Mining*. IEEE, 995–1000.

[27] Noveen Sachdeva, Giuseppe Manco, Ettore Ritacco, and Vikram Pudi. 2019. Sequential Variational Autoencoders for Collaborative Filtering. In *Proceedings of the ACM International Conference on Web Search and Data Mining*. 600–608.

[28] Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Semantic hashing. *International Journal of Approximate Reasoning* 50, 7 (2009), 969–978.

[29] Ying Shan, Jie Zhu, JC Mao, et al. 2018. Recurrent binary embedding for gpu-enabled exhaustive retrieval from billion-scale semantic vectors. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2170–2179.

[30] Dinghan Shen, Qinliang Su, Paidamoyo Chapfuwa, Wenlin Wang, Guoyin Wang, Ricardo Henao, and Lawrence Carin. 2018. NASH: Toward End-to-End Neural Architecture for Generative Semantic Hashing. In *Annual Meeting of the Association for Computational Linguistics*. 2041–2050.

[31] Yue Shi, Martha Larson, and Alan Hanjalic. 2014. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys (CSUR)* 47, 1 (2014), 3.

[32] Benyou Wang, Donghao Zhao, Christina Lioma, Qiuchi Li, Peng Zhang, and Jakob Grue Simonsen. 2020. Encoding word order in complex embeddings. In *International Conference on Learning Representations*.

[33] Chong Wang and David M Blei. 2011. Collaborative topic modeling for recommending scientific articles. In *ACM SIGKDD international conference on Knowledge discovery and data mining*. 448–456.

[34] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *Advances in neural information processing systems*. 1753–1760.

[35] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. 2010. Laplacian co-hashing of terms and documents. In *European Conference on Information Retrieval*. Springer, 577–580.

[36] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. 2010. Self-taught hashing for fast similarity search. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 18–25.

[37] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 325–334.

[38] Yan Zhang, Defu Lian, and Guowu Yang. 2017. Discrete personalized ranking for fast collaborative filtering from implicit feedback. In *AAAI Conference on Artificial Intelligence*. 1669–1675.

[39] Yan Zhang, Hongzhi Yin, Zi Huang, Xingzhong Du, Guowu Yang, and Defu Lian. 2018. Discrete Deep Learning for Fast Content-Aware Recommendation. In *ACM International Conference on Web Search and Data Mining*. 717–726.

[40] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference preserving hashing for efficient recommendation. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 183–192.

[41] Ke Zhou and Hongyuan Zha. 2012. Learning binary codes for collaborative filtering. In *ACM SIGKDD international conference on Knowledge discovery and data mining*. 498–506.