Unsupervised Multi-Index Semantic Hashing

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

Casper Hansen*
University of Copenhagen
c.hansen@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

Semantic hashing represents documents as compact binary vectors (hash codes) and allows both efficient and effective similarity search in large-scale information retrieval. The state of the art has primarily focused on learning hash codes that improve similarity search effectiveness, while assuming a brute-force linear scan strategy for searching over all the hash codes, even though much faster alternatives exist. One such alternative is multi-index hashing, an approach that constructs a smaller candidate set to search over, which depending on the distribution of the hash codes can lead to sub-linear search time. In this work, we propose Multi-Index Semantic Hashing (MISH), an unsupervised hashing model that learns hash codes that are both effective and highly efficient by being optimized for multi-index hashing. We derive novel training objectives, which enable to learn hash codes that reduce the candidate sets produced by multi-index hashing, while being end-to-end trainable. In fact, our proposed training objectives are model agnostic, i.e., not tied to how the hash codes are generated specifically in MISH, and are straight-forward to include in existing and future semantic hashing models. We experimentally compare MISH to state-of-the-art semantic hashing baselines in the task of document similarity search. We find that even though multi-index hashing also improves the efficiency of the baselines compared to a linear scan, they are still upwards of 33% slower than MISH, while MISH is still able to obtain state-of-the-art effectiveness.

KEYWORDS

Semantic hashing; multi-index hashing; similarity search

ACM Reference Format:
Christian Hansen, Casper Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2021. Unsupervised Multi-Index Semantic Hashing. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3442381.3450014

*Both authors share the first authorship.

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3450014

1 INTRODUCTION

Similarity search is a fundamental information retrieval task that aims at finding items similar to a given query. Efficient and effective similarity search is essential for a multitude of retrieval tasks such as collaborative filtering, content-based retrieval, and document search [28, 29]. Semantic Hashing [25] methods enable very efficient search by learning to represent documents (or other types of data objects) as compact bit vectors called *hash codes*, where the Hamming distance is used as the distance metric between hash codes. In this setting, similarity search is expressed as either radius search (finding all hash codes with a specified maximum Hamming distance), or as k-nearest neighbour (kNN) search by incrementally increasing the search radius until the Hamming distance to the k^{th} document is equal to the search radius. Early work on semantic hashing [31, 32] was inspired by techniques similar to spectral clustering [20] and latent semantic indexing [5], whereas modern approaches use deep learning techniques, typically unsupervised autoencoder architectures where hash codes are optimized by learning to reconstruct their original document representations [2, 3, 6, 8, 10, 27]. This line of work has led to extensive improvements of the effectiveness of document similarity search, but has had a lesser focus on efficiency, as it uses a brute-force linear scan of all the hash codes. While the highly efficient Hamming distance does enable large-scale search using linear scans [26], significantly faster alternatives exist. Multi-index hashing [7, 22, 23] is such an alternative, which, depending on a query hash code, can enable sub-linear search time by constructing a smaller set of candidate hash codes to search over. Hash codes can be used extremely efficiently as direct indices into a hash table for finding exact matches, however when doing radius search the number of such hash table lookups grows exponentially. Multi-index hashing is based on the observation that by splitting the hash codes into m substrings and building a hash table per substring, the exponential growth of the number of lookups can be significantly reduced. In practice, the efficiency of multi-index hashing is heavily dependent on the distribution of the hash codes, and most particularly their substrings, which affects the size of the constructed candidate set for a given query hash code. However, no existing semantic hashing methods consider this aspect, which we experimentally verify limits their efficiency.

To address the above efficiency problem, we **contribute** Multi-Index Semantic Hashing (MISH), an unsupervised semantic hashing model that generates hash codes that are both effective and highly efficient through being optimized for multi-index hashing. We identify two key hash code properties for improving multi-index

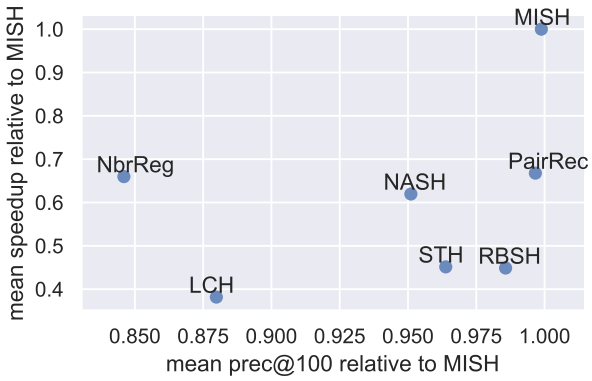

Figure 1: Method comparison relative to our proposed MISH, averaged over all used datasets. We plot each method as a point regarding its mean speedup of multi-index hashing compared to a linear scan, as well as mean prec@100, relative to MISH.

hashing efficiency, related to limiting the size of the candidate set produced by multi-index hashing. We operationalize these into two novel model agnostic training objectives that effectively reduce the number of hash codes per hash table lookup, while also limiting the necessary search radius for kNN search. These new objectives are fully differentiable and enable training MISH in an end-to-end fashion, and thus enable learning hash codes highly suited for multi-index hashing. We experimentally compare MISH to state-of-the-art semantic hashing baselines in the task of document similarity search. We evaluate their efficiency by comparing the speedup obtained by multi-index hashing over a linear scan, and find that on average the baselines are upwards of 33% slower than MISH. Even though MISH enables large efficiency gains, it is still able to obtain state-of-the-art effectiveness–summarized in Figure 1–where we plot the mean multi-index hashing speedup and mean prec@100 for each method relative to our MISH. Furthermore, we find that MISH can be tuned to enable even larger efficiency improvements at the cost of a slight reduction in effectiveness.

2 RELATED WORK

The problem of nearest neighbour search, also known as similarity search or proximity search, aims at finding the nearest item to a given query using a given distance measure. An efficient way of doing this is by using compact bit vectors (hash codes) that have low storage requirements and enable fast search through the use of the Hamming distance. Locality Sensitive Hashing (LSH) [4] is well-known type of hashing methods with strong theoretical guarantees [4, 15, 29]. However, these types of methods are *data-independent*, and thus unable to capture the semantics of a document (or other types of data objects such as images). In contrast, semantic hashing [25] based methods are *data-dependent*, and aim to learn hash codes such that nearest neighbour search on the hash codes leads to a search result similar to nearest neighbour search on the original document space [29].

2.1 Semantic hashing

Early work on semantic hashing focused on Spectral Hashing (SpH) [31], an approach inspired by spectral clustering [20] that aims at learning hash codes that preserve global similarity structures of the original documents. Laplacian co-hashing (LCH) [32] learns hash codes representing document semantics through a decomposition similar to latent semantic indexing [5]. Similarly to SpH, Graph Hashing [17] uses a graph representation for learning to capture the underlying global structure. Self-Taught Hashing (STH) [33] contrasts prior work by learning to preserve the *local* structures between samples identified by an initial kNN search in the original document space. The prior work above has primarily been solved as relaxed optimization problems, while later work has utilized deep learning for better capturing document semantics. Variational Deep Semantic Hashing (VDSH) [3], the first work in this direction, uses a more complex encoding of documents through a variational autoencoder architecture (this has since become the primary architecture in subsequent work). Similarly to STH, the authors of VDSH later expanded their model (now named NbrReg) [2] by including a loss function forcing the hash codes to be able to reconstruct unique words occurring in both the document and its neighbours in the original document space (found using BM25 [24]). While both VDSH and NbrReg improved effectiveness, they share the problem of using a post-hoc rounding of learned real-valued vectors, rather than learning the hash codes end-to-end. To fix this, NASH [27] proposed to learn the hash codes end-to-end through learning to sample the bits according to a Bernoulli distribution, which reduced the quantization errors compared to a rounding approach. Based on the same principle, BMSH [6] uses a Bernoulli mixture prior, but only manages to outperform a simple version of NASH, rather than consistently outperform the full NASH model. Similarly to NbrReg and STH, recent state-of-the-art approaches have incorporated neighbourhood knowledge: RBSH [8] incorporates a ranking-based objective, while PairRec [10] uses a pairwise reconstruction loss. The pairwise reconstruction loss is also used in our proposed MISH, and forces two hash codes of semantically similar documents to be able to reconstruct the unique words occurring in both documents, thus directly enabling the encoding of neighbourhood information into the hash codes.

2.2 Semantic hashing efficiency

The semantic hashing approaches above have led to substantial improvements in effectiveness, but they all use a brute-force linear scan for doing similarity search. While this is fast due to the high efficiency of the Hamming distance, hash codes were originally developed to be used as direct indices into a hash table [21, 25, 31], as to avoid a linear scan on a dataset of potential massive size. Through a hash table, finding exact hash code matches only requires a single lookup, but when varying the search radius in a similarity search (e.g., for performing kNN search) it leads to an exponentially increasing number of lookups. To fix this, multi-index hashing [7, 22, 23] has been explored, which enables sub-linear search time by building hash tables on substrings of the original hash codes (see Section 3.2 for a detailed description), and has been used for fast kNN search in hashing-based approaches related to collaborative filtering [9, 11, 12, 16, 34], knowledge graph search [30], and video hashing [35]. However, none of these approaches optimize the hash codes towards improving their multi-index hashing efficiency, but rather simply apply it on already learned hash codes. In contrast, our

proposed MISH is designed to directly learn hash codes suited for multi-index hashing in an end-to-end fashion, which significantly improves efficiency.

3 PRELIMINARIES

3.1 Hamming distance

Given a document $d \in \mathcal{D}$, let $z_d \in \{-1, 1\}^n$ be its associated bit string of n bits, called a *hash code*. The Hamming distance between two hash codes is defined as the number of differing bits between the codes:

$$d_H(z_d, z_{d'}) = \sum_{i=1}^{n} 1_{z_{d,i} \neq z_{d',i}} = \text{SUM}(z_d \text{ XOR } z_{d'}) \tag{1}$$

where the summation can be computed efficiently due to the *popcnt* instruction that counts the number of bits set to one within a machine word. Due to the efficiency of the Hamming distance, representing documents as hash codes enables both efficient radius search (retrieving all hash codes with a maximum Hamming distance of r to a given hash code), as well as kNN search. Specifically, kNN is performed through radius search by incrementally increasing the search radius up until the distance to the k^{th} hash code is equal to the search radius.

3.2 Multi-index hashing

Norouzi et al. [22, 23] propose a multi-index hashing strategy for performing exact radius and kNN search in the Hamming space. The aim of multi-index hashing is to build a candidate set C of hash codes, significantly smaller than the full document collection, $|C| \ll |\mathcal{D}|$. Given a query hash code z, it then suffices to compute the Hamming distances between z and the hash codes within C, rather than every hash code in the full collection $\mathcal{D}$. If the size of C is sufficiently small, and can be constructed efficiently, this leads to a sub-linear runtime compared to computing all possible Hamming distances.

Multi-index hashing is an efficient and easy to implement algorithm for building the candidate set C, where each code $z \in \mathcal{D}$ is split into m disjoint substrings, $z = [z^1, z^2, ..., z^m]$[1]. It now follows by the pigeonhole principle that if two codes z and z' are within radius r, i.e., $d_H(z, z') \leq r$, there exists at least one substring where the distance between the two codes is at most $r^* = \lfloor \frac{r}{m} \rfloor$. More specifically, if we assume some arbitrary fixed ordering of the substrings, and write the search radius as $r = r^* m + a,\ a < m$, one substring will have distance at most r^* in the first $a + 1$ substrings, or distance $r^* - 1$ in the remaining $m - a - 1$ substrings. Thus, the candidate set can be constructed by finding all hash codes where the distance within a substring is at most r^* for the first $a + 1$ substrings, or at most $r^* - 1$ for the remaining $m - a - 1$ substrings. For ease of notation, we will denote the substring search radius for substring i as r_i^*.

3.2.1 Efficient candidate set construction. Multi-index hashing uses hash tables to construct the candidate set efficiently. It constructs m hash tables, one for each substring, where the integer value of the substring is used as a key into the hash table, which then maps to all documents containing the same substring. Given substring z^i with $\frac{n}{m}$ bits, finding exact matches would require only a single lookup, but the number of lookups for radius search scales exponentially with the substring radius r_i^* as $\sum_{r'=0}^{r_i^*} (\frac{n}{m})^{r'}$. However, the exponential growth is significantly suppressed through fixing $m > 1$, which reduces both the base (through the substring length) and exponent (through the substring search radius), thus making it feasible to run in practice.

Performing radius search on the hash codes can then be done in a straight-forward fashion, by searching within each substring using their associated hash table, and then taking the union over the documents found for each substring search. Note that for kNN search, incrementally increasing the search radius from r to $r + 1$ only changes the search radius within a single substring, and this procedure can therefore be done very efficiently by building the candidate set incrementally as r is increased.

3.2.2 Hash code properties for efficient multi-index hashing. The efficiency of multi-index hashing depends heavily on the properties of the hash codes and how they are distributed in the Hamming space. The computational cost is dominated by the cost of sorting the candidate set according to the Hamming distance to the query hash code, such that the largest speedups are obtained when the candidate set size is small. Focusing on kNN search, the candidate set size is controlled by two factors:

Documents per hash table lookup Given a query hash code, the hash codes should be distributed such that the documents added to the candidate set are likely to appear among the top k documents with the least Hamming distance to the query hash code. To achieve this, the hash codes should be generated such that two hash codes with a low substring Hamming distance also have a low Hamming distance between the entire hash codes.

Search radius for kNN Given a query hash code, the search radius for kNN search is determined by the Hamming distance to the k^{th} closest hash code, which is unknown at query time. Since the number of hash table lookups increases exponentially with the substring search radius, the hash codes should be distributed such that the Hamming distance to the k^{th} document is kept low to limit the exponential growth (corresponding to substring distance less than 2).

Based on these factors it follows that different sets of hash codes for the same dataset can potentially have highly varying search *efficiency* without necessarily affecting the search *effectiveness* of the hash codes. To ensure that learned hash codes enable both efficient *and* effective search, the learning procedure must reflect both of these as part of the training objective. In the next section, we present how such codes can be learned for semantic hashing.

4 MULTI-INDEX SEMANTIC HASHING

We present Multi-Index Semantic Hashing (MISH), a semantic hashing model for unsupervised semantic hashing, which learns to generate hash codes that enable *both* effective and efficient search through being optimized for multi-index hashing. For a document $d \in \mathcal{D}$, MISH learns to generate an n-bit hash code $z_d \in \{-1, 1\}^n$ that represents its semantics, such that two semantically similar

[1]For ease of notation we will assume the substrings have the same length, but it is not a requirement.

documents have a low Hamming distance between them. MISH consists of a component learning to encode document semantics, and two novel components that ensure the learned hash codes are well suited for multi-index hashing (see Section 3.2), based on the hash code properties discussed in Section 3.2.2. These two components are in fact model agnostic, i.e., not tied to how the hash codes are encoded, so they are straight-forward to include in future work for improving efficiency. Below is an overview of the components:

Semantic encoding MISH is based on a variational autoencoder architecture, where the encoder learns to generate the semantic hash code according to repeating n Bernoulli trials, while a decoder learns to be able to reconstruct the original document from the generated hash code. We choose to use the pairwise reconstruction loss proposed in the state-of-the-art PairRec [10] model to ensure that document semantics are well captured within the hash codes.

Reducing the number of documents per hash table lookup Given a query hash code z_q, during training we sample another hash code z_s with a low Hamming distance within one of its substrings, but a high Hamming distance using the full hash code, which can be considered a false positive match. We derive an objective that maximises the Hamming distance between the particular substrings of z_q and z_s, thus effectively pushing the substrings of z_q and z_s apart, reducing the number of such false positive matches in the hash table lookup.

Control the search radius for kNN Given a query hash code z_q, during training we sample a hash code z_r with $d_H(z_q, z_r) = r$, where r is the Hamming distance at the top k$^{\text{th}}$ position in a kNN search from z_q. In case r is too large, which leads to a large number of hash table lookups, r is reduced through an objective that minimizes the Hamming distance between z_q and z_r, thus effectively pushing the top k hash codes closer together.

In the following sections we present each component individually, and describe how they are jointly optimized for learning the hash codes in an end-to-end fashion.

4.1 Semantic encoding

We use a variational autoencoder to learn a document encoder that generates a hash code capturing the document's semantics, as well as encoding the local neighbourhood structure of encoded document. This is done by training the codes such that a hash code z should be able to reconstruct not only the original document d, but also documents in the neighborhood of d defined by an appropriate similarity function.

To learn the hash codes, we compute the log likelihood of document $d \in \mathcal{D}$ conditioned on its code z as a sum of word likelihoods, which needs to be maximized:

$$\log p(d|z) = \sum_{j \in \mathcal{W}_d} \log p(w_j|z) \tag{2}$$

where $p(z)$ is sampled by repeating n Bernoulli trials and W_d is the set of unique words in document d. However, due to the size of the Hamming space, the above is intractable to compute in practice, so the variational lower bound [14] is maximized instead:

$$\log p(d) \geq E_{Q(\cdot|d)}[\log p(d|z)] - \mathrm{KL}(Q(z|d)||p(z)) \tag{3}$$

where $Q(z|d)$ is a learned approximation of $p(z)$ that functions as the decoder, and KL is the Kullback-Leibler divergence. In the text below, we first describe the encoder ($Q(z|d)$), then the decoder ($p(d|z)$), and lastly the loss function.

4.1.1 Encoder. The encoder is a feed forward network, with two hidden layers using ReLU activation units, followed by a final output layer using a sigmoid activation function, to get the bitwise sampling probabilities:

$$Q(z|d) = \mathrm{FF}_{\sigma}(\mathrm{FF}_{\mathrm{ReLU}}(\mathrm{FF}_{\mathrm{ReLU}}(d \odot e_{\mathrm{imp}}))) \tag{4}$$

where FF denotes a feed forward layer, and e_{imp} is a learned word level importance embedding. The purpose of the importance embedding is to scale each word of the document representation, such that unimportant words have less influence on generating the hash codes. During training, the bits are sampled according to the bitwise sampling probabilities, while being chosen deterministically for evaluation (choosing the most probable bit value without sampling). To make the sampling differentiable, we employ the straight-through estimator [1].

4.1.2 Decoder. The decoder, $\log p(d|z)$, is defined as maximizing the log likelihood of each word in document d:

$$\log p(d|z) = \sum_{j \in \mathcal{W}_d} \log \frac{e^{\mathrm{logit}(w_j|z)}}{e^{\sum_{i \in \mathcal{W}_{\mathrm{all}}} \mathrm{logit}(w_i|z)}} \tag{5}$$

where $\mathrm{logit}(w|z)$ is the logit for word w_j and $\mathcal{W}_{\mathrm{all}}$ are all the words in the corpus. The logit for each word is computed as:

$$\mathrm{logit}(w|z) = f(z)^T (E_{\mathrm{word}}(I(w) \odot e_{\mathrm{imp}})) + b_w \tag{6}$$

where $f(z)$ is a noise-infused hash code with added Gaussian noise (zero mean and a parameterized variance σ^2), which is annealed during training and results in lower variance for the gradient estimates [14]. E_{word} is a word embedding learned during training, $I(w)$ denotes a one-hot encoding of word w, and b_w is a bias term.

4.1.3 Semantic encoder loss. To make the loss function aware of the local neighbourhood structure around a given document, we use pairwise reconstruction as proposed by Hansen et al. [10]. To this end, we use a similarity function independent of the learned hash codes to compute a set of the p most semantically similar documents in the neighbourhood around document d, denoted as $\mathcal{N}_d^p$. For each $d_+ \in \mathcal{N}_d^p$, we construct (d_q, d_+) with corresponding hash codes (z_d, z_+) and define the loss function based on the variational lower bound from Eq. 3 as:

$$\begin{aligned}\mathcal{L}_{\mathrm{semantic}} = &-E_{Q(\cdot|d_q)}[\log p(d_q|z_q)] + \beta \mathrm{KL}(Q(z_q|d_q)||p(z_q)) \\ &-E_{Q(\cdot|d_+)}[\log p(d|z_+)] + \beta \mathrm{KL}(Q(z_+|d_+)||p(z_+))\end{aligned} \tag{7}$$

As the hash codes, z_q and z_+ both have to reconstruct document d_q (known as *pairwise reconstruction*) the hash codes are forced to not only encode their associated document, but also the local neighbourhood $\mathcal{N}_d^p$ as a whole.

4.2 Reduce the number of documents per hash table lookup

The candidate set estimated by multi-index hashing can be reduced by limiting the number of documents added by each hash table lookup. Specifically, we are interested in limiting the number of false positive matches, i.e., candidate documents added due to a low substring Hamming distance, but where the Hamming distance on the full hash code is above the search radius. Given z_q, a substring i, and the top k search radii r and r_i^*, we sample a hash code z_s as follows:

$$z_s = \underset{z_j}{\text{argmax}} \quad d_H(z_q, z_j) \cdot 1_{[d_H(z_q^i, z_j^i) \leq r_i^*]} \cdot 1_{[d_H(z_q, z_j) > r]} \tag{8}$$

which corresponds to sampling the hash code with the largest Hamming distance that has a substring Hamming distance below r_i^* and is outside the r-ball centered on z_q (expressed via $1_{[d_H(z_q,z_j)>r]}$). By sampling hash codes with the largest value of $d_H(z_q, z_s)$, z_s is unlikely to be within top k, but would still appear in the candidate set due to the low substring Hamming distance. Based on the sampling of such hash codes, we can derive an objective that maximizes the Hamming distance within the substring as long as z_s appears in the candidate set, which is expressed in the following loss function:

$$\mathcal{L}_{\text{false-positive}} = -d_H(z_q^i, z_s^i) \tag{9}$$

In case one or both indicator functions in Eq. 8 are always 0 for a given query, this loss is simply set to 0.

4.2.1 Finding the pair (z_q, z_s). While training the network, the hash codes potentially change in each iteration, hence we need to continuously sample the pair (z_q, z_s) during training. As recomputing every hash code for every batch is computationally expensive, we employ a memory module that is continuously updated with the generated hash codes in addition to the associated document ids. We denote this memory module as M. The memory size (i.e., the number of hash codes to keep in M) is denoted by s_{mem}, and the memory module is updated using a first-in first-out (FIFO) strategy. Due to the compact representation of the hash codes and document ids, we fix s_{mem} to be the size of the training set[2].

Using the sampling requirements from Eq. 8, z_s can now be obtained from the memory module. However, since the memory module may contain outdated hash codes due to model updates in previous training iterations, the validity of the pair (z_q, z_s) must be ensured. This is done by recomputing z_s (based on the document features obtained through the stored document id) on the current model parameters, and verifying whether it is still valid according to the sampling requirements (if not, the loss is set to 0). Note that the search radii r_i^* and r for top k retrieval also needs to be estimated based on the memory module. However, since z_s is sampled as the hash code with the largest Hamming distance to z_q, smaller deviations from the true radii are not problematic, as the worst-case outcome simply is that the already far apart (z_q, z_s) pair is pushed slightly further apart then necessary.

[2]For truly massive-scale datasets, or due to specific hardware constraints, the memory size could be fixed to a number less than the training set size.

Table 1: Dataset statistics.

	documents	multi-class	classes	unique words
TMC	28,596	Yes	22	18,196
reuters	9,848	Yes	90	16,631
agnews	127,598	No	4	32,154

4.3 Control search radius

In Section 4.2 we detailed how to reduce the number of false positive documents per hash table lookup, while we now focus on how to reduce the number of such lookups. Given a query hash code z_q, we aim to control the search radius r to limit the exponential increase in the number of hash table lookups, which happens when the substring search radius $r_i^* > 1,\ i \in \{1, ..., m\}$ (see Section 3.2.1), corresponding to $r > 2m - 1$. To this end, we compute r for the query based on the memory module, and sample a hash code z_r with $d_H(z_q, z_r) = r$, resulting in the hash code pair (z_q, z_r). To reduce the number of lookups, we define a loss function that minimizes the Hamming distance of the pair:

$$\mathcal{L}_{\text{radius}} = d_H(z_q, z_r) \cdot 1_{[r>2m-1]} \tag{10}$$

where the indicator function ensures that the Hamming distance is only minimized in cases where the search radius is too large. Similarly to sampling z_s for reducing the number of documents per hash table lookup (Eq. 8), z_r may be outdated in the memory module, but is recomputed and it is verified whether its radius is still equal to r (otherwise the loss is set to 0).

4.4 Combined loss function

MISH is trained in an end-to-end fashion by jointly optimizing the semantic loss (Eq. 7), reducing the number of false positive documents per hash table lookup (Eq. 9), and controlling the number of such lookups (Eq. 10) as follows:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{semantic}} + \alpha_1 \mathcal{L}_{\text{false-positive}} + \alpha_2 \mathcal{L}_{\text{radius}} \tag{11}$$

where the hyperparameter weights, α_1 and α_2, control the trade-off between the semantic encoding and tuning the hash codes towards more efficient multi-index hashing search. However, optimizing both effectiveness and efficiency are not necessarily mutually exclusive because any permutation of the hash code bits provides the same effectiveness, but some permutations result in better multi-index hashing efficiency. Lastly, observe that $\mathcal{L}_{\text{false-positive}}$ and $\mathcal{L}_{\text{radius}}$ are model agnostic, as neither are tied to how the hash codes are generated, and can thus easily be incorporated in any semantic hashing model for improving efficiency.

5 EXPERIMENTAL EVALUATION

5.1 Datasets

We evaluate MISH on well-known and publicly available datasets used in related work [3, 8, 10, 27] and summarized in Table 1: (1) *TMC* consists of multi-class air traffic reports; (2) *Agnews* consists of single-class news documents; and (3) *reuters* consists of multi-class news documents, where a filtering is applied that removes documents if none of its labels occur among the top 20 most frequent labels (as done in [3, 8, 10, 27]).

We use the preprocessed data provided by Hansen et al. [10], where documents are represented using TF-IDF vectors, where words occurring only once, or in more than 90% of the documents, are removed. We use the provided data splits, which split the datasets into training (80%), validation (10%), and testing (10%), where the validation loss is used for early stopping.

5.2 Evaluation setup

We evaluate the hash codes in the task of document similarity search (using kNN search), where we evaluate both effectiveness and efficiency. Each document is represented as a hash code, such that document similarity search can be performed using the Hamming distance between two hash codes. For evaluation purposes, we denote a document to be relevant (i.e., similar) to a query document, if the documents share at least one label, meaning that in multi-class datasets two documents do not need to share all labels.

For effectiveness we follow related work [3, 8, 10, 27] by considering the retrieval performance as measured by precision@100. However, existing work computes the scores based on random tie splitting, which is problematic for hash codes as ties occur often due to the limited number of $n+1$ different Hamming distances for n-bit hash codes. Instead, we compute the tie-aware precision@100 metric [19] corresponding to the average-case retrieval performance. Additionally, due to the large number of possible ties, we also compute the *worst-case* retrieval performance by fixing ties such that irrelevant documents appear before relevant ones before computing precision@100.

For efficiency we measure the runtime for performing top-100 retrieval on the training set, where each test document acts as a query document once. We perform a linear scan, i.e., brute-force computation of all Hamming distances, as well as multi-index hashing based search [23]. We use the linear scan and multi-index implementation made available by Norouzi et al. [23][3], where we follow the practical recommendation of splitting hash codes into substrings of 16 bits for multi-index hashing[4]. We repeat all timing experiments 100 times, and report the median speedup of using multi-index hashing over the linear scan. Note that the runtime of multi-index hashing naturally varies between the methods used for generating hash code, whereas the linear scan time is the same independent of the used method. All timing experiments were performed on an Intel Core i9-9940X@3.30 GHz.

5.3 Baselines

We compare our proposed MISH to non-neural approaches with post-processing rounding of document vectors to obtain hash codes (STH [33] and LCH [32]), neural approaches with post-processing rounding (NbrReg [2]), and neural end-to-end approaches that incorporate the rounding as part of the model (NASH [27], RBSH [8], and PairRec [10]). All baselines are tuned following the original papers.

In contrast to our MISH, existing semantic hashing baselines have focused on maximizing effectiveness, while simply assumed retrieval is done using a brute-force linear scan, rather than faster alternatives such as multi-index hashing. However, how bits are assigned into substrings impacts multi-index hashing efficiency, as the candidate set size may be larger than necessary. To this end, we include the greedy substring optimization (GSO) heuristic proposed by Norouzi et al. [23], which greedily assigns bits to substrings as to minimize the correlation between the bits within each substring.

5.4 Tuning

To tune MISH[5] we fix the number of hidden units in the encoder to 1000, and vary the number of documents in the pairwise reconstruction neighbourhood ($\mathcal{N}_d^p$) from $\{10, 25, 50, 100\}$, where both 10 and 25 worked well for reuters and TMC, whereas 100 was consistently chosen for agnews. Similarly to Hansen et al. [10], $\mathcal{N}_d^p$ is constructed based on retrieving the top p most semantically similar documents based on 64 bit STH hash codes. For the KL-divergence, we tune β from $\{0, 0.01\}$, where 0 was chosen most often, thus effectively removing the KL term in those cases. For the variance annealing in the noise-infused hash codes, we fix the initial value to 1 and reduce by 10^{-6} every iteration (as per [8, 10]). For the combined loss, we tune α_1 from $\{1, 3, 5, 7\}$ and α_2 from $\{0.01, 0.05, 0.1\}$, where $\alpha_1 = 3$ and $\alpha_2 = 0.01$ was chosen for reuters and TMC, and $\alpha_1 = 7$ and $\alpha_2 = 0.05$ for agnews. As we focus on learning hash codes that maintain state-of-the-art effectiveness, while improving efficiency, we choose to use only the semantic loss ($\mathcal{L}_{\text{semantic}}$) on the validation set for model selection and early stopping, rather than the weighted total loss. Lastly, MISH is optimized using the Adam optimizer [13] with a learning rate from $\{0.001, 0.005\}$, where 0.005 was chosen for reuters and TMC, and 0.001 for agnews.

5.5 Results

The experimental results are summarized for effectiveness in Table 2 and efficiency in Table 3. In Table 2, the highest worst-case and average-case scores per column are highlighted in bold, and the second highest are underlined. In Table 3, the largest and second largest speedups (independent of applying the greedy substring optimization (GSO)) are highlighted in bold and underlined, respectively. Additionally, we report the linear scan time per document as a point of reference for the speedups. In both tables, ▲ represents statistically significant improvements over the second best method at the 0.05 level using a two -tailed paired t-test.

5.5.1 Retrieval effectiveness. Table 2 shows the effectiveness measured by worst-case and average-case prec@100 across the datasets using 32 and 64 bit hash codes (corresponding to the typical machine word sizes). Across all methods, we observe a larger gain in worst-case prec@100 when increasing the number of bits in the hash codes, compared to the average-case prec@100, where only smaller increases are obtained. Thus, increasing the number of bits is beneficial when worst-case performance is important no matter the chosen method. The increase in worst-case prec@100 happens because the documents are being spread out more in the Hamming space as the number of bits are increased, which reduce the number of Hamming distance ties.

Our MISH method obtains the best results for worst-case prec@100 in all cases, and additionally it obtains the best average-case prec@100 for reuters and agnews. For TMC, PairRec obtains marginally higher

[3] https://github.com/norouzi/mih/
[4] https://github.com/norouzi/mih/blob/master/RUN.sh

[5] We make our code publicly available at https://github.com/Varyn/MISH

Table 2: Worst case and average case precision@100. The highest precision is highlighted in bold, and the second highest is underlined. ▲ represents statistically significant improvements over the second best method at the 0.05 level using a two tailed paired t-test.

	Reuters				TMC				Agnews			
	32 bits		64 bits		32 bits		64 bits		32 bits		64 bits	
Prec@100	Worst	Average	Worst	Average	Worst	Average	Worst	Average	Worst	Average	Worst	Average
LCH	0.5995	0.6616	0.6283	0.6613	0.6658	0.7510	0.7421	0.7817	0.6822	0.7599	0.7423	0.7775
STH	0.7730	0.8046	0.7803	0.7968	0.6858	0.7693	0.7481	0.7816	0.6823	0.8237	0.7931	0.8374
NbrReg	0.5785	0.6329	0.6327	0.6616	0.3272	0.6648	0.5862	0.6827	0.7274	0.7914	0.7535	0.7928
NASH	0.7330	0.7737	0.7767	0.7967	0.6845	0.7709	0.7535	0.7953	0.7059	0.8018	0.7748	0.8107
RBSH	0.7809	0.8110	0.8011	0.8182	0.7459	0.8107	0.7852	0.8158	0.7797	0.8347	0.8053	0.8317
PairRec	0.7812	0.8218	0.8087	0.8316	0.7320	**0.8187**	0.7922	**0.8288**	0.7700	0.8348	0.8114	0.8407
MISH	**0.7965**	**0.8286**	**0.8248**	**0.8377**	**0.7608**▲	0.8156	**0.7931**	0.8261	**0.7818**	**0.8375**	**0.8116**	**0.8419**

Table 3: Speedup of multi-index hashing over a brute-force linear scan, as well as linear scan time per document. Greedy substring optmizing (GSO) [23] corresponds to the correlation-based post-hoc heuristic, and Default corresponds to using the hash codes as is. The largest speedup is highlighted in bold (independent of post-hoc fix), and the second largest is underlined. ▲ represents statistically significant improvements (100 repeated timing experiments) over the second best method at the 0.05 level using a two tailed paired t-test.

	Reuters				TMC				Agnews			
	32 bits		64 bits		32 bits		64 bits		32 bits		64 bits	
Speedup	Default	GSO	Default	GSO	Default	GSO	Default	GSO	Default	GSO	Default	GSO
LCH	2.5182	0.3508	0.8937	0.5603	7.2427	1.5879	2.1837	3.0767	18.2709	25.6365	3.7311	5.1354
STH	2.9374	0.2783	1.0116	0.5695	7.0382	2.2211	2.1947	2.4791	14.6936	29.2779	4.4515	9.5676
NbrReg	6.3841	0.7190	4.9589	0.8299	7.1497	1.7346	2.8800	5.2506	21.7102	23.7698	6.8518	7.5279
NASH	5.6356	0.6037	4.5869	0.8417	9.4069	1.5725	4.3819	4.3886	20.7673	23.1443	4.8355	5.3849
RBSH	4.4342	0.2965	1.7083	0.8051	7.0831	1.9708	2.7957	2.9146	25.9186	27.1164	4.5036	4.7345
PairRec	5.4296	0.3721	3.0770	0.9433	10.9118	1.7063	5.1129	4.9614	29.7880	33.4096	7.1765	7.8676
MISH	**7.0698**▲	1.1216	**5.4466**▲	1.0282	**14.6296**▲	2.1923	**8.8696**▲	6.1645	**44.0151**▲	35.9177	**13.6756**▲	12.5788
Linear scan time	0.000070 s		0.000069 s		0.000111 s		0.000109 s		0.000432 s		0.000407 s	

average-case prec@100 compared to MISH, but PairRec is in most cases the second best method. In general, the difference in effectiveness between the best and second best performing method is relatively small, and we only obtain statistically significant improvements for worst-case prec@100 at 32 bits for TMC. As both MISH and the state-of-the-art PairRec are based on the same semantic loss (Eq. 7), it was to be expected that MISH would not significantly improve effectiveness over PairRec. However, it is important to notice that including the two additional losses in MISH, for improving efficiency, did not negatively impact effectiveness either. In fact, the additional losses have a regularizing effect that reduces the number of Hamming distance ties (hence improving the worst-case performance), as the losses force the hash codes to be better spread in the Hamming space.

Table 4 shows the average percentage decreases in prec@100 (for both worst-case and average-case) compared to the best worst-case and average-case scores, respectively. For a given method, a decrease of 0% corresponds to that method always being the best performing method across all datasets. We observe that on average, MISH outperforms the other methods in both cases, with a noticeable better average worst-case effectiveness. Additionally, it

Table 4: Average decrease in worst-case and average-case prec@100 compared to the best scores per dataset. An average decrease of 0% corresponds to obtaining the best performance across all datasets.

	32 bits		64 bits	
	Δ Worst	Δ Average	Δ Worst	Δ Average
LCH	-16.65%	-12.57%	-12.93%	-11.46%
STH	-8.51%	-3.53%	-4.45%	-3.71%
NbrReg	-30.44%	-15.98%	-18.84%	-14.83%
NASH	-9.24%	-5.58%	-5.12%	-4.21%
RBSH	-1.39%	-1.15%	-1.55%	-1.71%
PairRec	-2.40%	-0.38%	-0.70%	-0.29%
MISH	**0.00%**	**-0.13%**	**0.00%**	**-0.11%**

can be seen that the baselines generally have larger worst-case decreases compared to the average-case decreases, which shows that the baselines broadly cluster the hash codes more, thus resulting in larger number of Hamming distance ties.

5.5.2 Retrieval efficiency. Table 3 shows the relative speedup compared to a linear scan of the hash codes produced by each method,

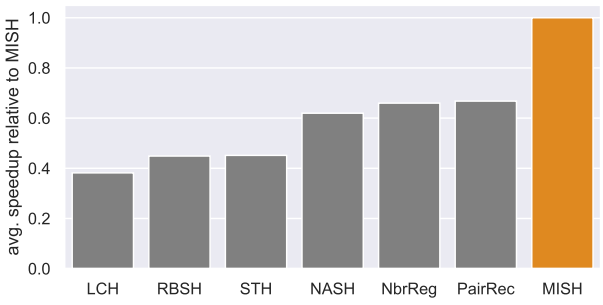

Figure 2: Speedup of multi-index hashing over linear scan relative to MISH averaged across all datasets and bit configurations.

with and without greedy substring optimization (GSO) [23], together with the linear scan time per document. The linear scan time per document is slightly lower for 64 bit hash codes compared to 32 bit, due to our machine using a 64 bit operating system. In addition, due to the sequential access pattern in a linear scan, the larger memory requirement of 64 bit hash codes does not increase the scan time. Overall, we observe that all methods achieve a higher speedup for 32 bits compared to the speedup at 64 bits, caused by an increase in the search radius for top 100 retrieval, which leads to a larger number of hash table lookups, thus increasing the candidate set of multi-index hashing. Furthermore, as expected the speedup increases as the dataset size increases, because the relative size of the candidate set decreases compared to the entire set of hash codes.

Table 3 shows that MISH is significantly faster than the baselines on all datasets and number of bits. The speedup obtained by the baselines are largely varied across the datasets and number of bits. No baseline can perform consistently well in all settings. GSO leads to larger speedups for the baselines on agnews and for 64 bit hash codes on TMC (except for PairRec), but worse on reuters and 32 bit hash codes on TMC. For MISH, applying GSO always decreases the speedup, which is caused by GSO changing the substring structure learned during training through our proposed loss functions (Eq. 9 and Eq. 10), to a less optimal one. This highlights that GSO is not always beneficial and, more importantly, the limitation of making a post-hoc change to the structure of the hash codes. In contrast, MISH directly optimizes the desirable hash code properties for multi-index hashing (see Section 3.2.2) in an end-to-end fashion, which significantly improves multi-index hashing efficiency.

Figure 2 shows the average relative speedup, across all datasets and number of bits, compared to MISH. Generally the baselines can be clustered in two groups of similar average efficiency (LCH, RBSH, STH) and (NASH, NbrReg, PairRec). Interestingly, RBSH is the only neural method among the least efficient methods, even though PairRec and RBSH share the same underlying neural architecture with the difference being that RBSH uses a pairwise ranking loss and PairRec uses a pairwise reconstruction loss. This shows that it is problematic to assume the produced hash codes by a given method will be efficient, without directly optimizing them as done in MISH, as even small changes in the model may greatly influence the efficiency.

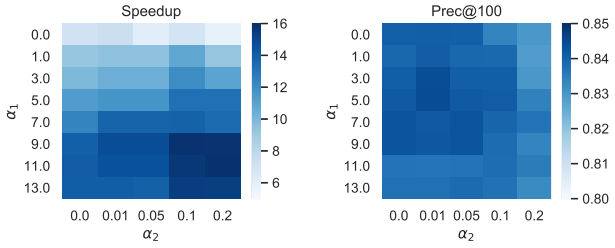

Figure 3: Hyperparameter impact on speedup and prec@100: α_1 reduces the number of documents per hash table lookup, and α_2 controls the number of such lookups.

5.6 Efficiency and effectiveness impact of α_1 and α_2

To maintain state-of-the-art effectiveness, model selection was done based only on the semantic loss (Eq. 7) on the validation set, but model training is naturally still done on the weighted total loss (Eq. 11). We now investigate the impact of the total loss weights (α_1 and α_2) on the efficiency and effectiveness of MISH, where α_1 reduces the number of documents per hash table lookup, and α_2 controls the number of such lookups. To this end, we report the speedup and average-case prec@100 for as two grid plots with α_1 and α_2 as the axes, exemplified for 64 bit hash codes on agnews.

The grid plots can be seen in Figure 3, where the top left corner ($\alpha_1 = 0,\ \alpha_2 = 0$) corresponds to the PairRec baseline [10]. For the speedup plot, we observe a clear trend that higher values of both α_1 and α_2 improve efficiency, but α_1 has the largest impact. This is expected since α_1 directly reduces the number of documents per hash table lookup, reducing the candidate set across all queries, while α_2 affects a smaller subset of queries who exhibit a large search radius. For the prec@100 plot, we observe that higher values of α_1 and α_2 reduce effectiveness, which highlights the possible trade-off when tuning the hyperparameters. However, the area with the largest prec@100 scores is relatively large, thus enabling a large efficiency improvement without compromising effectiveness.

5.7 Distribution of candidate set sizes

The computational cost of multi-index hashing is dominated by the cost of sorting the candidate set according to the Hamming distances to the query hash code. We now investigate the distribution of the candidate set size per hash code query for each method, as visualized in Figure 4 using symmetrical violin plots[6]. For the cases where GSO leads to improved speedups, we compute the candidate set based on the hash codes after applying GSO. Across all datasets and bit sizes, we observe that MISH is able to better concentrate the density towards a lower number of candidates compared to the baselines, thus explaining the large speedup improvements. By observing the larger candidate sizes for 64 bit hash codes compared to 32 bit hash code, we can directly see the reason for the speedup gap between the two hash code sizes reported in Table 3. Furthermore, the baselines exhibiting poor speedups are primarily due to long density tails or a more uniform distribution of candidate set sizes.

[6]A violin plot is a combination of a boxplot and a symmetrical density plot that shows the full distribution of the data.

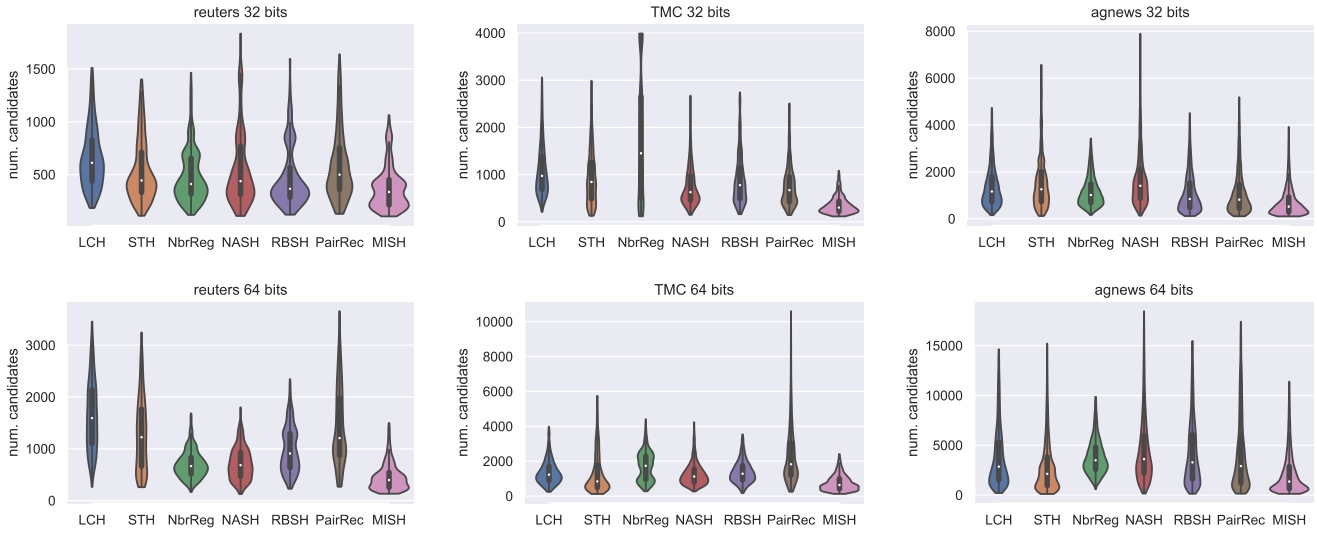

Figure 4: Violin plots (combined density and boxplot) of the number of document candidates found using multi-index hashing for top-100 retrieval.

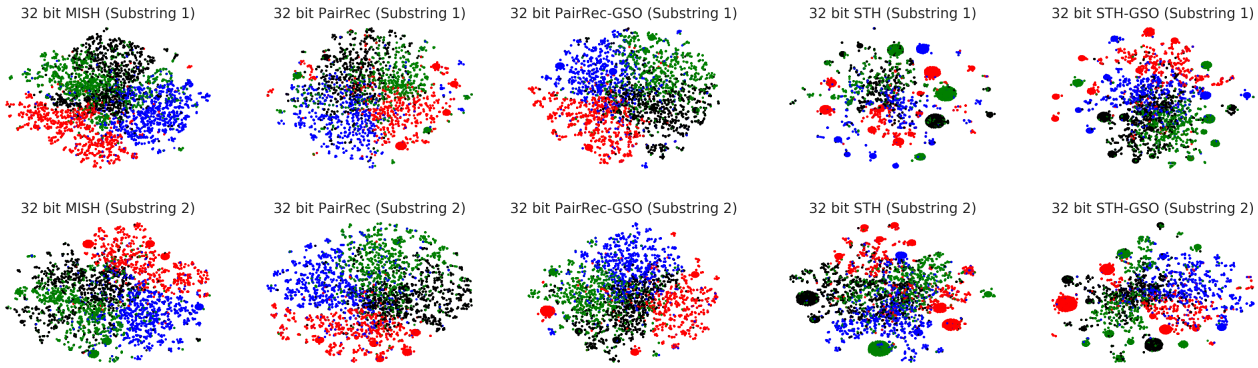

Figure 5: t-SNE [18] visualization of the two substring of MISH, PairRec, and STH 32 bit hash codes on agnews. Each color represents one of 4 different classes. Greedy substring optimization (GSO) [23] is applied on PairRec and STH as it improved their effiency for multi-index hashing.

While this naturally leads to worse overall efficiency, it also has the potential problem of large query time variance, which may limit their application in extremely time-constrained use cases.

5.8 Substring-level visualization

To further investigate the differences in how the methods distribute the bits within the hash codes, we choose to visualize the individual substrings. We consider 32 bit hash codes from the test set of agnews, as it only contains two substrings, while agnews is a single-class dataset, which makes it easier to visualize if hash codes of the same class are clustered. We consider MISH, PairRec, and STH as representative methods, as it also enables visualizing the impact of GSO on the two baselines. PairRec is chosen due to being the second best method on both effectiveness and efficiency for 32 bit hash codes on agnews, while STH represents a method where GSO greatly improves (doubles) its speedup (see Table 3). Figure 5 shows a two-dimensional t-SNE [18] visualization, where it is important to keep in mind that each plot contains the same number of points, such that a plot appearing more sparse (more distance between points) corresponds to more hash codes being highly similar. When comparing MISH and PairRec, we observe that they do appear similar, but MISH is slightly less sparse, meaning the hash codes are better spread throughout the Hamming space. For STH, we observe that prior to applying GSO, the hash codes are tightly clustered and highly sparse (especially in the first substring), but GSO is able to redistribute the bits such that the substrings better utilize the space. This redistribution reduces the amount of false positive candidates found in multi-index hashing, thus leading to the large observed speedup.

5.9 Conclusion

We presented Multi-Index Semantic Hashing (MISH), an unsupervised semantic hashing model that learns hash codes well suited for multi-index hashing [23], which enables highly efficient document similarity search. Compared to a brute-force linear scan over all the hash codes, multi-index hashing constructs a smaller candidate set to search over, which can provide sub-linear search time. We identify key hash code properties that affect the size of the candidate set, and use them to derive two novel objectives that enable MISH to learn hash codes that results in smaller candidate sets when using multi-index hashing. Our objectives are model agnostic, i.e., not tied to how the hash codes are generated specifically in MISH, which means they are straight-forward to incorporate in existing and future semantic hashing models. We experimentally compared MISH to state-of-the-art semantic hashing baselines in the task of document similarity search, where we evaluated both efficiency and effectiveness. While multi-index hashing also improves the efficiency of the baseline hash codes compared to a linear scan, they are still upwards of 33% slower than our proposed MISH. Interestingly, these large efficiency gains of MISH can be obtained without reducing effectiveness, as MISH is still able to obtain state-of-the-art effectiveness, but we do find that even further efficiency improvements can be obtained, but at the cost of an effectiveness reduction. In future work, we plan to explore supervised versions of MISH, specifically the impact of expanding our proposed efficiency objectives with label information, which could decrease the number of irrelevant documents in the candidate sets.

REFERENCES

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

[2] Suthee Chaidaroon, Travis Ebesu, and Yi Fang. 2018. Deep Semantic Text Hashing with Weak Supervision. SIGIR, 1109–1112.

[3] Suthee Chaidaroon and Yi Fang. 2017. Variational deep semantic hashing for text documents. In *SIGIR*. 75–84.

[4] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 253–262.

[5] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science* 41, 6 (1990), 391–407.

[6] Wei Dong, Qinliang Su, Dinghan Shen, and Changyou Chen. 2019. Document Hashing with Mixture-Prior Generative Models. In *EMNLP*. 5226–5235.

[7] Dan Greene, Michal Parnas, and Frances Yao. 1994. Multi-index hashing for information retrieval. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE, 722–731.

[8] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2019. Unsupervised Neural Generative Semantic Hashing. In *SIGIR*. 735–744.

[9] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2020. Content-aware Neural Hashing for Cold-start Recommendation. In *SIGIR*. 971–980.

[10] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2020. Unsupervised Semantic Hashing with Pairwise Reconstruction. In *SIGIR*. 2009–2012.

[11] Christian Hansen, Casper Hansen, Jakob Grue Simonsen, and Christina Lioma. 2021. Projected Hamming Dissimilarity for Bit-Level Importance Coding in Collaborative Filtering. In *Proceedings of The Web Conference 2021*. In print.

[12] Wang-Cheng Kang and Julian McAuley. 2019. Candidate Generation with Binary Codes for Large-Scale Top-N Recommendation. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1523–1532.

[13] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *ICLR*.

[14] Diederik P Kingma and Max Welling. 2014. Auto-encoding variational bayes. In *ICLR*.

[15] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2020. *Mining of massive data sets*. Cambridge university press.

[16] Defu Lian, Xing Xie, and Enhong Chen. 2019. Discrete matrix factorization and extension for fast item recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[17] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2011. Hashing with graphs. In *ICML*.

[18] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[19] Frank McSherry and Marc Najork. 2008. Computing Information Retrieval Performance Measures Efficiently in the Presence of Tied Scores. In *Proceedings of the IR Research, 30th European Conference on Advances in Information Retrieval*. 414–421.

[20] Andrew Y Ng, Michael I Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *NeurIPS*. 849–856.

[21] Mohammad Norouzi and David J Fleet. 2011. Minimal loss hashing for compact binary codes. In *ICML*.

[22] Mohammad Norouzi, Ali Punjani, and David J Fleet. 2012. Fast search in hamming space with multi-index hashing. In *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 3108–3115.

[23] Mohammad Norouzi, Ali Punjani, and David J Fleet. 2013. Fast exact search in hamming space with multi-index hashing. *IEEE transactions on pattern analysis and machine intelligence* 36, 6 (2013), 1107–1119.

[24] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at TREC-3. *Nist Special Publication Sp* 109 (1995), 109.

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

[26] Ying Shan, Jian Jiao, Jie Zhu, and JC Mao. 2018. Recurrent binary embedding for gpu-enabled exhaustive retrieval from billion-scale semantic vectors. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2170–2179.

[27] 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 *ACL*. 2041–2050.

[28] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. 2015. Learning to hash for indexing big data—A survey. *Proc. IEEE* 104, 1 (2015), 34–57.

[29] Jingdong Wang, Ting Zhang, Nicu Sebe, and Heng Tao Shen. 2018. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence* 40, 4 (2018), 769–790.

[30] Meng Wang, Haomin Shen, Sen Wang, Lina Yao, Yinlin Jiang, Guilin Qi, and Yang Chen. 2019. Learning to Hash for Efficient Search Over Incomplete Knowledge Graphs. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1360–1365.

[31] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *NeurIPS*. 1753–1760.

[32] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. 2010. Laplacian co-hashing of terms and documents. In *ECIR*. Springer, 577–580.

[33] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. 2010. Self-taught hashing for fast similarity search. In *SIGIR*. ACM, 18–25.

[34] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 325–334.

[35] Hanwang Zhang, Meng Wang, Richang Hong, and Tat-Seng Chua. 2016. Play and rewind: Optimizing binary representations of videos by self-supervised temporal hashing. In *Proceedings of the 24th ACM international conference on Multimedia*. 781–790.