Getting Deep Recommenders Fit: Bloom Embeddings for Sparse Binary Input/Output Networks

Joan Serrà and Alexandros Karatzoglou
Telefónica Research
Pl. Ernest Lluch i Martín, 5
Barcelona, Spain 08019
firstname.lastname@telefonica.com

ABSTRACT

Recommendation algorithms that incorporate techniques from deep learning are becoming increasingly popular. Due to the structure of the data coming from recommendation domains (i.e., one-hot-encoded vectors of item preferences), these algorithms tend to have large input and output dimensionalities that dominate their overall size. This makes them difficult to train, due to the limited memory of graphical processing units, and difficult to deploy on mobile devices with limited hardware. To address these difficulties, we propose Bloom embeddings, a compression technique that can be applied to the input and output of neural network models dealing with sparse high-dimensional binary-coded instances. Bloom embeddings are computationally efficient, and do not seriously compromise the accuracy of the model up to 1/5 compression ratios. In some cases, they even improve over the original accuracy, with relative increases up to 12%. We evaluate Bloom embeddings on 7 data sets and compare it against 4 alternative methods, obtaining favorable results. We also discuss a number of further advantages of Bloom embeddings, such as 'on-the-fly' constant-time operation, zero or marginal space requirements, training time speedups, or the fact that they do not require any change to the core model architecture or training configuration.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; *Learning latent representations*; • **Information systems** → *Document representation*;

KEYWORDS

Deep recommenders, sparse input/output, Bloom filters, neural network, embeddings.

1 INTRODUCTION

The size of neural network models that deal with sparse inputs and outputs is often dominated by the dimensionality of such inputs and outputs. Deep networks used for recommender systems and collaborative filtering are a paradigmatic case, as they have to deal with high-dimensional sparse vectors, typically in the order from tens of thousands to hundreds of millions, both at the input and the output of the network (e.g., [12, 26, 42, 49]). This results in large models that present a number of difficulties, both at training and prediction stages. Apart from training and prediction times, an obvious bottleneck of such models is space: their size (and even performance) is hampered by the physical memory of graphical processing units (GPUs), and they are difficult to deploy on mobile devices with limited hardware (cf. [24]).

One option to reduce the size of sparse inputs and outputs is to embed them into a lower-dimensional space. Embedding sparse high-dimensional inputs is commonplace (e.g., [5, 37, 44]). However, embedding sparse high-dimensional outputs, or even inputs and outputs at the same time, is much less common (cf. [1, 4, 48]). Importantly, typical embeddings still require the storage and processing of large matrices with the same dimensionality as the input/output (like the original neural network model would do). Thus, the gains in terms of space are limited. As mentioned, the size of such models is dominated by the input/output dimensionality, with input and output layers representing around 99.9% of the total amount of models' parameters. An example can be found in the deep recommender of Hidasi et al. [26], which uses a gated recurrent unit to perform session-based recommendations with input/output layers of dimensionality 330,000 and internal layers of dimensionality 100.

In general, an ideal embedding procedure for sparse high-dimensional inputs/outputs should produce compact embeddings, of much lower dimensionality than the original input/output. In addition, it should consume little space, both in terms of storage and memory space. Smaller sizes imply less parameters, thus training the model on embedded vectors would also be faster than with the original instances. The embedding of the output should also lead to a formulation for which the appropriate loss should be clear. Embeddings should not compromise the accuracy of the model nor the required number of training epochs to obtain that accuracy. In addition, no changes to the original core architecture of the model should be required to achieve good performance (obviously, input/output dimensions must change). The embedding should also be fast; if not to be done directly 'on-the-fly', at least fast enough so that speed improvements made during training are not lost in the embedding operation. Last, but not least, output embeddings should be easily reversible, so that the output of the model could be mapped to the original items at prediction time.

In this work, we propose an unsupervised embedding technique that fulfills all the previous requirements. It can be applied to both

RecSys '17, August 27-31, 2017, Como, Italy

ACM ISBN 978-1-4503-4652-8/17/08…$15.00
https://doi.org/10.1145/3109859.3109876

input and output layers of neural network models that deal with binary (one-hot encoded) inputs and/or outputs. In doing so, it produces lower-dimensionality binary embeddings that can be easily mapped to the original instances. Provided that the embedding dimension is not too low, the accuracy is not compromised. Furthermore, in some cases, we show that training with embedded vectors can even increase prediction accuracy (we find that is the case for three of the considered recommendation tasks). The embedding requires no changes to the core network structure nor to the model configuration, and works with a softmax output, the most common output activation for binary-coded instances. The embedding moreover preserves the ranking order of items which is crucial in recommender systems. As it is unsupervised, the embedding does not require any preliminary training. Moreover, it is a constant-time operation that can be either performed on-the-fly, requiring no disk or memory space, or can be cached in memory, occupying orders of magnitude less space than a typical embedding matrix. Lower dimensionality of input/output vectors result in faster training, and the mapping from the embedded space to the original one does not add an overwhelming amount of time to the prediction stage. The proposed embedding is based on the idea of Bloom filters [7], and therefore it inherits part of the theory developed around that idea [8, 9, 18, 38].

2 RELATED WORK

A common approach to embed high-dimensional inputs is the hashing trick [33, 41, 46]. However, the hashing trick approach does not deal with outputs, as it offers no explicit way to map back from the (dense) embedding space to the original space. A more elementary version of the hashing trick [20] can be used at the outputs by considering it as a special case of the Bloom-based methodology proposed here. A framework providing both encoding and decoding strategies is the error-correcting output codes (ECOC) framework [17]. Originally designed for single-class outputs, it can be also applied to class sets [3]. Another example of a framework offering recovery capabilities is kernel dependency estimation [48]. The compressed sensing approach of Hsu et al. [30] builds on top of ECOC to reduce multi-label regression to binary regression problems. Similarly, Cissé et al. [15] use Bloom filters to reduce multi-label classification to binary classification problems and improve the robustness of individual binary classifiers' errors.

Data-dependent embeddings that require some form of learning also exist. A typical approach is to rely on variants of latent semantic analysis or singular value decomposition (SVD), exploiting similarities or correlations that may be present in the data. Again, the issue of mapping from the embedding space to the original space is left unresolved. Nonetheless, recently, Chollet [14] has successfully applied a K-nearest neighbors (KNN) algorithm to perform such a mapping and to derive a ranking of the items in the original space. An SVD decomposition of the pairwise mutual information matrix (PMI) is used to perform the embedding, and cosine similarity is used as loss function and to retrieve neighbors. Using the KNN trick offers the possibility to exploit different types of factorization of similarity-based matrices. Canonical correlation analysis is an example that considers both inputs and outputs at the same time [28]. Other examples considering output embeddings are nuclear norm regularized learning [2], label embedding trees [4], or the WSABIE algorithm [47]. In the presence of side information, like text descriptions, item or class taxonomies, or manually-collected data, a range of approaches are applicable. Akata et al. [1] provide a comprehensive list. In our study, we assume no side information is available and focus on input/output-based embeddings.

From a more general perspective, reducing the space of (or compressing) neural network models is an active research topic, driven by the need to deploy such models in systems with limited hardware resources. A common approach is to reduce the size of already trained models by some quantization and/or pruning of the connections in dense layers [16, 24, 31]. A less frequent approach is to reduce the model size before training [11]. These methods typically do not focus on input layers and, to the best of our knowledge, none of them deals with high-dimensional outputs. It is also worth noting that a number of techniques have been proposed to efficiently deal with high-dimensional outputs, specially in the natural language processing domain. The hierarchical softmax approach [39] or the more recent adaptive softmax [22] are two examples of those. Yet, as mentioned, the focus of these works is on speed, not on space. The work of Vincent et al. [45] focuses on both aspects of very large sparse outputs but, to the best of our knowledge, cannot be directly applied to common softmax outputs.

3 BLOOM EMBEDDINGS

3.1 Bloom filters

Bloom filters [7] are a compact probabilistic data structure that is used to represent sets of items, and to efficiently check whether an item is a member of a set [38]. Since the instances we deal with represent sets of one-hot encoded items, Bloom filters are an interesting option to embed those in a compact space with good recovery (or checking) guarantees.

In essence, Bloom filters project every item of a set to k different positions of a binary array $\mathbf{u}$ of size m. Projections are done using a set of k independent hash functions $\mathsf{H} = \{H_i\}_{i=1}^{k}$, each of which with a range from 1 to m, ideally distributing the projected items uniformly at random [38]. Proper independent hash functions can be derived using enhanced double hashing or triple hashing [18]. The number of hash functions k is usually a constant, $k \ll m$, proportional to the expected number of items to be projected.

To check if an item is in $\mathbf{u}$, one feeds it to the k hash functions H to get k array positions. If any of the bits at these positions is 0, then the item is definitely not in the set. Thus, item checks return no false negatives, meaning that the structure gives an answer with 100% recall [38]. However, if all k bits at the projected positions are 1, then either the item is in the set, or the bits have by chance been set to 1 during the insertion of other set items. This implies that false positives are possible, due to collisions between projections of different items [8]. The values of m and k can be adjusted to control the probability of such collisions. However, in practice, m is usually constrained by space requirements, and $k \leq 10$ is employed, independently of the number of items to be projected, and giving less than 1% false positive probability [9].

3.2 Embedding and recovery

In the following, we describe the use of Bloom filter techniques in embedding binary high-dimensional instances, and the recovery or mapping to such instances from these embeddings. We denote our approach as Bloom embedding (BE). The idea we pursue is to embed both inputs and outputs and to perform training in the embedding space. For that, only a probability-based output activation is required, together with a loss function that is appropriate for such activations.

Let $\mathbf{x}$ be an input or output instance with dimensionality d, such that $\mathbf{x} = [x_1, \dots x_d]$, $x_i \in \{0, 1\}$. Instances $\mathbf{x}$ are assumed to be sparse, that is, $\sum_{i=1}^{d} x_i \ll d$. Because of that, we can more conveniently (and compactly) represent $\mathbf{x}$ as set $\mathrm{p} = \{p_i\}_{i=1}^{c}$, $p_i \in \mathbb{N}_{\leq d}$, where c is the number of non-zero elements and p_i is the position of such elements in $\mathbf{x}$. For every set p, we generate an embedded instance $\mathbf{u}$ of dimensionality $m < d$, such that $\mathbf{u} = [u_1, \dots u_m]$, $u_i \in \{0, 1\}$. To do so, we first set all m components of $\mathbf{u}$ to 0. Then, iteratively, for every position p_i, $i = 1, \dots c$, and every projection H_j, $j = 1, \dots k$, we assign

$$u_{H_j(p_i)} = 1. \tag{1}$$

Notice that, since H_j has a range between 1 and m, $k \geq 1$, and $m < d$, a number of original positions p_i may map to the same index in $\mathbf{u}$. Bloom filters mitigate this by properly choosing k independent hash functions H (Sec. 3.1). Notice furthermore that the process has no space requirements, as H is computed on-the-fly. Finally, notice that the embedding of a set p is constant time: the process is $O(ck)$, with c bounded by the maximum number of active items in $\mathbf{x}$, $c \ll d$, and k being a constant that is set beforehand, $k \ll m < d$. In practice, this constant time is dominated by the time spent on H to generate a hash. If we want to be faster than that, and at the same time ensure an optimal (uniform) distribution of the outputs of H, we can pre-compute a hash matrix storing the projections or hash indices for all the potential items in p. We can do it by generating vectors $\mathbf{h} = [h_1, \dots h_k]$ for each p_i, where h_j is a uniformly randomly chosen integer between 1 and m (without replacement). This way, by pre-generating all projections for all d items, we end up with a $d \times k$ matrix $\mathbf{H}$ of integers between 1 and m, which we can easily store in random-access memory (RAM), not in the GPU memory.

We now explain how to recover a probability-based ranking of the d items at the output of the model. Assuming a softmax activation is used, we have a probability vector $\hat{\mathbf{v}} = [\hat{v}_1, \dots \hat{v}_m]$ that, at training time, is compared to the binary embedding $\mathbf{v}$ of the ground truth $\mathbf{y}$, with its active positions q (analogously to $\mathbf{x}$ and p). To unravel the embedding $\hat{\mathbf{v}}$ and map to the d original items of $\mathbf{y}$, we can understand $\hat{\mathbf{v}}$ as a k-way factorization of every item $\hat{y}_i$ of our prediction $\hat{\mathbf{y}}$. Then, following the idea of Bloom filters, if y_i maps to v_i and $\hat{v}_i = 0$, we can confirm that that item is definitely not in the output of the model (Sec. 3.1). Otherwise, if $\hat{v}_i$ is relatively large, we want the likelihood of that item to reflect that. Specifically, given an active position q_i from q representing $\mathbf{y}$, we can compute the likelihood

$$L(q_i) = \prod_{j=1}^{k} \hat{v}_{H_j(q_i)}, \tag{2}$$

and assign outputs $\hat{y}_i = L(q_i)$. Alternatively, if a more numerically-stable output is desired, we can compute the negative log-likelihood

$$L(q_i) = -\sum_{j=1}^{k} \log\left(\hat{v}_{H_j(q_i)}\right). \tag{3}$$

Both operations, when iterated for $i = 1, \dots d$, define a ranking over the items in $\mathbf{y}$ and from there, if needed, a probability distribution can be recovered by re-normalization. We here do not perform such re-normalization, as all the problems we consider can be mapped to a typical ranking recommendation setting.

3.3 Suitability

Note that BE, by construction, already offers a number of the aforementioned desired qualities for sparse binary high-dimensional embeddings (Sec. 1). Specifically, BE is designed for both inputs and outputs, offering a rank-based mapping between the original instances and the embedded vectors. BE yields a more compact representation of the original instance and requires no disk or memory space (at most some marginal RAM space, not GPU memory). In addition, BE can be performed on-the-fly, without training, and in constant time. In the following, we demonstrate the remaining desirable qualities using a comprehensive experimental setup: we show that the accuracy of the model remains stable or even increases given a reasonable embedding dimension, that no changes in the model architecture nor configuration are required, that training times are faster thanks to the reduction of the number of parameters of the model, that evaluation times do not carry much overhead, and that performance is generally better than a number of alternative approaches.

4 EXPERIMENTAL SETUP

4.1 General considerations

We demonstrate that BE works under several settings and that it can be applied to multiple tasks. In particular, we focus on recommendation and collaborative filtering but also demonstrate the validity of the approach on a natural language processing task. We consider a number of data sets, network architectures, configurations, and evaluation measures. In total, we define 7 different setups, which we describe in Sec. 4.2. We also demonstrate that BE is competitive with respect to the available alternatives. To this end, we consider 4 different state-of-the-art approaches, which we overview in Sec. 4.3.

Data sets are formed by inputs with n instances, corresponding to either individual instances (or one-hot encoded user profiles) or to sequences of instances (or profile lists). Outputs, also of n instances, correspond to individual instances or to class labels. Instances have an original dimensionality d, corresponding to the cardinality of all possible profile items. Given the nature of the considered problems, instances are very sparse, with all but c items being different from 0, $c \ll d$, typically with c/d in the order of 10^{-5} (Table 1).

For each data set, and based on the literature, we select an appropriate baseline neural network architecture. We experiment with both feed-forward (autoencoder-like) and recurrent networks, carefully selecting their parameters and configuration to match (or even improve) the state-of-the-art results. For the sake of comparison, we

Table 1: Data set statistics after data cleaning and splitting. From left to right: data set name, number of instances n, test split size, instance dimensionality d, median number of non-zero components c, and median density c/d.

Data set	n	Split	d	c	c/d
ML	138,224	10,000	15,405	18	$1.2 \cdot 10^{-3}$
PTB	929,589	82,430	10,001	1	$1.0 \cdot 10^{-4}$
CADE	40,983	13,661	193,998	17	$8.8 \cdot 10^{-5}$
MSD	597,155	50,000	69,989	5	$7.1 \cdot 10^{-5}$
AMZ	916,484	50,000	22,561	1	$4.4 \cdot 10^{-5}$
BC	25,816	2,500	54,069	2	$3.7 \cdot 10^{-5}$
YC	1,865,997	50,000	35,732	1	$2.8 \cdot 10^{-5}$

Table 2: Experimental setup and baseline scores. From left to right: data set name, network architecture and optimizer, evaluation measure, random score $S_{\mathbf{R}}$, and baseline score S_0.

Data set	Architecture + Optimizer	Meas.	S_R	S_0
ML	Feed-forward + Adam	MAP	0.003	0.160
PTB	LSTM + SGD	RR	0.001	0.342
CADE	Feed-forward + RMSprop	Acc	8.5	58.0
MSD	Feed-forward + Adam	MAP	<0.001	0.066
AMZ	Feed-forward + Adam	MAP	<0.001	0.049
BC	Feed-forward + Adam	MAP	<0.001	0.010
YC	GRU + Adagrad	RR	<0.001	0.368

also choose appropriate and well-known evaluation measures [34]. Depending on the data set (Table 2), we work with mean average precision (MAP), reciprocal ranks (RR), or % of accuracy (Acc).

Each combination of data set, network architecture, configuration, and evaluation measure defines a task. For every task, we compute a baseline score S_0, corresponding to running the plain neural network model without any embedding. We then report the performance of the i-th execution with a particular embedding with respect to the baseline score using S_i/S_0. This way, we can compare the performance across different tasks using different evaluation measures, reporting relative improvement/loss with respect to the baseline. Similarly, to compare across different dimensionalities, we report the ratio of embedding dimensionality with respect to the original dimensionality, m/d, and to compare across different training and evaluation times, we report time ratios with respect to the baseline, T_i/T_0.

4.2 Tasks

We now give a brief summary of the 7 considered tasks (Tables 1 and 2). All data sets are publicly-available and, for all tasks, we make sure that the network architecture and the experimental setup is sufficient to achieve a state-of-the-art result. We use softmax outputs and categorical cross-entropy losses in all experiments.

(1) Movielens (ML): movie recommendation with the Movielens 20M data set[1] [25]. Ratings were discretized with a threshold of 3.5 and movies with less than 5 ratings and users with less than two movies were discarded. Inputs and outputs were built by splitting user profiles at a certain timestamp uniformly at random, ensuring a minimum of one movie in both input and output. To perform recommendations, we build on top of Wu et al. [49] and employ a 3-layer feed-forward neural network model with 150 rectified linear units [21] in the hidden layers. We optimize the weights of the network using cross-entropy and Adam [32], with a learning rate of 0.001 and parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We evaluate the accuracy of the model using MAP.

(2) Million song data set (MSD): song recommendation with the Million song data set[2] [6]. We assume that a user likes a song when he/she has listened to it a minimum of 3 times. We then remove the songs that appear less than 20 times and build user profiles with a minimum of 5 songs. Inputs and outputs are split at a uniformly random timestamp. To recommend future listens to the user we use a 3-layer feed-forward neural network and 300 rectified linear units in the hidden layers. As for the rest, we proceed as with the ML task. We evaluate the accuracy of the model with MAP.

(3) Amazon book reviews (AMZ): book recommendation with the Amazon book reviews data set[3] [35]. We proceed as with the ML data set, but this time setting the minimum number of ratings per book to 100. We employ a 4-layer feed-forward neural network with 300 rectified linear units in the hidden layers, and optimize its parameters with Adam. We evaluate the accuracy of the model with MAP.

(4) Book crossing (BC): book recommendation with the book crossing data set[4] [50]. To perform recommendations we use the same architecture and configuration as with the MSD task, but this time we use 250 units in the hidden layers.

(5) YooChoose (YC): session-based recommendation with the YooChoose RecSys15 challenge data set[5]. We work with the training set of the challenge and keep only the click events. We take the first 2 million sessions of the data set which have a minimum of 2 clicks. To predict the next click, we proceed as in Hidasi et al. [26] and consider a GRU model [13]. We set the inner dimensionality to 100 and train the network with Adagrad [19], using a learning rate of 0.01. We evaluate the accuracy of the model with RR.

(6) Penn treebank (PTB): next word prediction with the Penn treebank data set [36]. The vocabulary is limited to 10,000 words, with all other words mapped to an 'unknown' token (Table 1). We consider the end of the sentence as an additional token and form input sequences of length 10. Inspired by Graves [23], we perform next word prediction with an LSTM network [27]. We set the inner dimensionality to 250 and train the network with SGD. We use a learning rate of 0.25, a momentum of 0.99, and clip gradients to have a maximum norm of 1 [23]. We evaluate the accuracy of the model with the RR of the correct prediction.

[1] http://grouplens.org/datasets/movielens/20m/

[2] http://labrosa.ee.columbia.edu/millionsong/tasteprofile

[3] http://jmcauley.ucsd.edu/data/amazon/

[4] http://www2.informatik.uni-freiburg.de/~cziegler/BX/

[5] http://recsys.yoochoose.net

(7) CADE web directory (CADE): text categorization with the CADE web directory of Brazilian web pages[6] [10]. The data set contains around 40,000 documents assigned to one of 12 categories such as services, education, health, or culture. To perform classification we use a 4-layer feed-forward neural network with a softmax output. The number of units is, from input to output, 400, 200, 100, and 12, and we use rectified linear units as activations for the hidden layers. We train the network with RMSprop [43], a learning rate of 0.0002, and exponential decay of 0.9. Notice that this is the only considered task where output embeddings are not required (classification into 12 text categories). We use Acc as evaluation measure.

4.3 Alternative approaches

To compare the performance of BE with the state-of-the-art, we consider 4 different embedding alternatives. We base our evaluation on performance, measured at a given input/output compression ratio. It is important to note that, in general, besides performance, alternative approaches do not present some of the other desired qualities that BE offers, such as on-the-fly operation, constant-time, no supervision, or no network/configuration changes (Secs. 1 and 3.3). Note also that methods for embedding both inputs and outputs, allowing to map embedded instances to the original ones, are scarce (Sec. 2). Because of that, in some of the considered alternatives we had to perform some adaptations.

(1) Hashing trick (HT). We first consider the popular hashing trick for classifier and recommender inputs [33, 46]. In general, these methodologies only focus on inputs and are not designed to deal with any type of output. Nonetheless, in the case of binary outputs, variants like the one used by Ganchev and Dredze [20] can be adapted to map to the original items using Eqs. 2 or 3. In fact, considering this adaptation for recovery, the approach can be seen as a special case of BE with $k = 1$ (Sec. 3.2).

(2) Error-correcting output codes (ECOC). Originally designed for single-class targets [17], ECOC can be applied to class sets (inputs and outputs), with its corresponding encoding and decoding strategies [3]. Yet, in the case of training neural networks, it is not clear which loss function should be used. The obvious choice would be to use the Hamming distance. However, in pre-analysis, a Hamming loss turned out to be significantly inferior than cross-entropy. Therefore, we use the latter in our experiments. We construct the ECOC matrix with the randomized hill-climbing method of [17].

(3) Pairwise mutual information (PMI). Recently, Chollet [14] has proposed a PMI approach for embedding sets of image labels into a dense space of real-valued vectors. The approach is based on the SVD of a PMI matrix computed from counting pairwise co-occurrences. It uses cosine similarity as the loss function and, at prediction time, it performs KNN (again using cosine similarity) with the projection of individual labels to obtain a ranking.

(4) Canonical correlation analysis (CCA). CCA is a common way to learn a joint dense, real-valued embedding for both inputs and outputs at the same time [28]. CCA can be computed using SVD on a correlation matrix [29] and, similarly to PMI, we can use the KNN trick to rank items or labels at prediction time. Correlation is now the metric of choice, both for the loss function and for determining the neighbors.

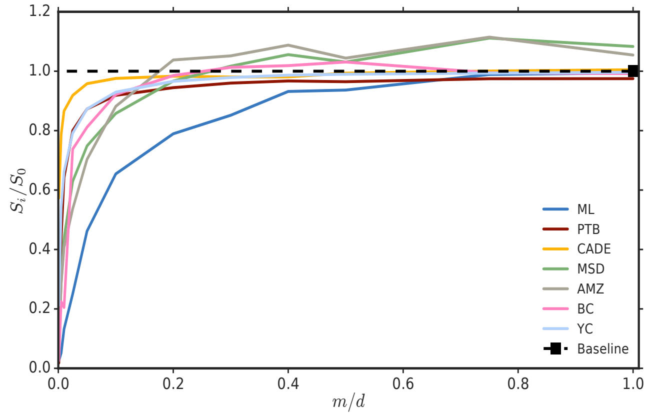

Figure 1: Score ratios S_i/S_0 as a function of dimensionality ratio m/d using $k = 4$. Qualitatively similar plots are observed for other values of k.

5 RESULTS

5.1 Compression and performance

We start by reporting on the performance of BE as a function of the embedding dimension. As mentioned, to facilitate comparisons, we report in relative terms, using score ratios S_i/S_0 and dimensionality ratios m/d. When plotting the former as a function of the latter, we see several things that are worth noting (Fig. 1). Firstly, we observe that, for most of the tasks, score ratios approach 1 as m approaches d. This indicates that the introduction of BE does not degrade the original score of the Baseline when the embedding dimension m is comparable to the original dimension d. Secondly, we observe that the lower the dimensionality ratio, the lower the score ratio. This is to be expected, as one cannot embed sets of items with their intrinsic dimensionality to an infinitesimally small m. Importantly, the reduction of S_i/S_0 should not be linear with m/d, but should maximize S_i for low m (thus getting curves close to the top left corner of Fig. 1). We see that BE fulfills this requirement. In general, we can reduce the input/output size up to 5 times ($m/d = 0.2$) and still maintain more than 92% of the value of the original score. The ML task is the only exception, which we think is due to the abnormally high density of the data (Table 1), inhibiting the embedding to low dimensions[7]. CADE is the task for which BE achieves the highest S_i for low m. Presumably, the CADE task is the easiest one we consider, as only input embeddings are required.

An additional observation is worth noting (Fig. 1). Interestingly, we find that BE can improve the scores over the Baseline for a number of tasks. That is the case for 3 out of the 7 considered tasks (all of them recommendation tasks): MSD with $m/d \geq 0.3$, AMZ with $m/d \geq 0.2$, and BC with $0.3 \leq m/d \leq 0.6$. The fact that an

[6] http://ana.cachopo.org/datasets-for-single-label-text-categorization

[7] Note that the ML data is essentially collected through a survey-type method [25].

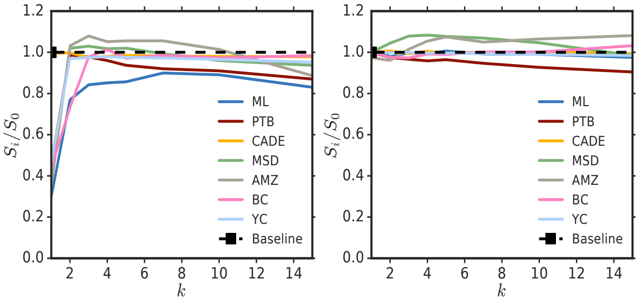

Figure 2: Score ratios S_i/S_0 as a function of the number of hash functions k: using $m/d = 0.3$ (left) and $m/d = 1$ (right).

embedding performs better than the original Baseline has been also observed in some other methods for specific data sets [14, 33, 48]. For instance, Chollet [14] has reported increases up to 7% using the PMI approach on the so-called JFT data set. Here, depending on the task and the embedding dimension, relative increases go from 1 to 12%. Given that the data sets where we observe these increases are some of the less dense ones (Table 1), we hypothesize that, in the case of BE, such increases come from having k times more active positions in the ground truth output (recall that one output item is projected k times using k independent hash functions, Sec. 3.2). With k more times elements set to 1 in the output, a better estimation of the gradient may be computed (larger errors that propagate back to the rest of the network).

We now focus on performance as a function of the number of projections k (Fig. 2), reporting score ratios S_i/S_0 as above. From repeating the plots for different values of m/d, we observe that S_i/S_0 is always low for $k = 1$ (Fig. 2, left), except when m approaches d, where we have an almost flat behavior (Fig. 2, right). In general, S_i/S_0 jumps up for $k \geq 2$ and remains stable until $k \approx 10$, where the decrease of S_i/S_0 becomes more apparent (Fig. 2, left). The best operating range typically corresponds to $2 \leq k \leq 4$. The ML task is again an exception, with a best operating range around $7 \leq k \leq 10$.

5.2 Training and retrieval time

Besides performance scores, it is interesting to assess whether the reduction of input and output dimensions has an effect to training and evaluation times. To this end, we plot the time ratios T_i/T_0 as a function of the dimensionality ratio m/d (Fig. 3). Regarding training times, we basically observe a linear decrease with m/d (Fig. 3, left). ML is an exception to the trend, and CADE and AMZ experiment almost no decrease for very low dimensionality ratios $m/d < 0.2$. In general, we confirm faster training times thanks to the reduction of the number of parameters of the model, dominated by input/output matrices (output dimension also affecting the time to compute the loss function). We obtain a 2 times speedup for a 2 times input/output compression and, roughly, a little bit over 3 times speedup for a 5 times input/output compression. Regarding evaluation times, we also observe a linear trend (Fig. 3, right). However, this time, T_i/T_0 is not as low, with values slightly above 1 but always below 1.5 (with the exception of CADE for $m/d > 0.6$). Overall, this indicates that, compared to the Baseline evaluation time, the mapping used by BE when reconstructing the output does not introduce an overwhelming amount of extra computation time. With the exception of ML, extra computation time is below 20% for $m/d < 0.5$.

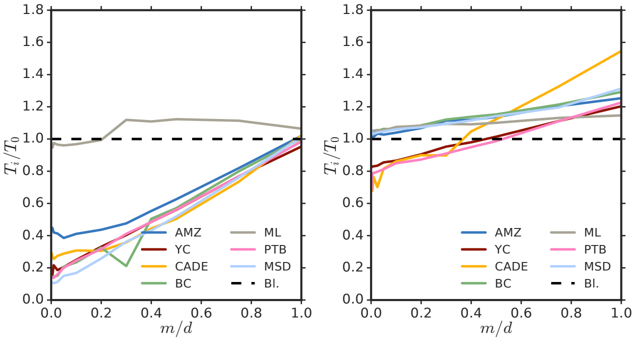

Figure 3: Time ratios T_i/T_0 as a function of dimensionality ratios m/d with $k = 4$: training time (left) and evaluation time (right). Qualitatively similar plots are observed for other values of k. Bl. denotes baseline.

5.3 Comparison with alternatives

Finally, we compare the performance of BE to the one of the considered alternative methods. We do so by establishing a dimensionality ratio m/d and computing the corresponding score ratio S_i/S_0 for a given task (Table 3). We see that BE is better than the alternative methods in 5 out of the 7 tasks (10 out of the 14 considered test points). PMI is better in one of the tasks (CADE) and CCA is better also in one of the tasks (AMZ). It is relevant to note that, when BE wins, it always does so by a relatively large margin (see, for instance, the ML or YC tasks). Otherwise, when an alternative approach wins, generally it does so by a smaller margin (see, for instance, the AMZ task).

These results become more relevant if we realize that PMI and CCA are both SVD-based approaches, introducing a separate degree of supervised learning to the task, exploiting pairwise item co-occurrences and correlations, respectively (Sec. 4.3). In contrast, BE does not require any learning. We formulate a co-occurrence-based version of BE below, which achieves moderate performance increments over BE and more closely approaches the performance of PMI and CCA on the two tasks where BE was not already performing best. A further interesting thing to note is that we confirm the small variation in the score ratios obtained for $2 \leq k \leq 10$ (Fig. 2). Here, score ratios for $3 \leq k \leq 5$ are often comparable in a statistical significance sense (Table 3).

6 GOING ONE STEP FURTHER WITH CO-OCCURRENCE-BASED COLLISIONS

6.1 Co-occurrence-based embedding

In Bloom filters and BE, collisions are unavoidable due to the lower embedding dimensionality and the use of multiple projections (Sec. 3). In addition, we have seen that some alternative approaches produce embeddings by exploiting co-occurrence information (Secs. 2 and 4.3). Here, we study a variant of BE that takes

Table 3: Comparison of BE with the considered alternatives. Score ratios S_i/S_0 for different combinations of data set and compression ratio m/d. Best results are highlighted in bold, up to statistical significance (Mann-Whitney U, $p > 0.05$).

Test point		Alternative methods				BE		
Data set	m/d	HT	ECOC	PMI	CCA	$k = 3$	$k = 4$	$k = 5$
ML	0.2	0.234	0.342	0.043	0.209	**0.750**	**0.770**	0.722
ML	0.3	0.285	0.208	0.045	0.200	0.796	**0.813**	**0.815**
PTB	0.2	0.357	0.453	0.837	0.638	**0.919**	0.908	0.881
PTB	0.4	0.528	0.454	0.836	0.695	**0.942**	0.920	0.902
CADE	0.01	0.857	0.359	**0.984**	0.928	0.862	0.853	0.855
CADE	0.03	0.914	0.363	**1.002**	0.950	0.914	0.925	0.926
MSD	0.05	0.078	0.268	0.216	0.679	0.695	**0.738**	**0.738**
MSD	0.1	0.151	0.310	0.321	0.740	**0.835**	**0.841**	0.832
AMZ	0.1	0.166	0.182	0.851	**1.030**	0.864	0.881	0.861
AMZ	0.2	0.289	0.185	0.995	**1.048**	1.016	1.029	1.008
BC	0.05	0.189	0.817	0.022	0.313	0.777	0.750	**0.837**
BC	0.1	0.199	0.886	0.025	0.465	**0.965**	**0.919**	0.831
YC	0.03	0.150	0.076	0.776	0.466	**0.841**	**0.858**	**0.858**
YC	0.05	0.240	0.083	0.777	0.517	**0.919**	**0.910**	**0.928**

Algorithm 1 Pseudocode for CBE.

Input: Input and/or output instances $\mathbf{X}$ ($n \times d$ sparse binary matrix), embedding dimensionality m, number of projections k, and pre-computed hashing matrix $\mathbf{H}$ ($d \times k$ integers matrix).

Output: Co-occurrence-based hashing matrix $\mathbf{H}'$.

```
1:  $\mathbf{C} \leftarrow \mathbf{X}^{\mathrm{T}}\mathbf{X}$
2:  $\mathbf{C} \leftarrow \mathbf{C} \odot \textsc{sgn}(\mathbf{C} - \textsc{avgfreq}(\mathbf{X}))$
3:  $\mathbf{c}^{\mathrm{VAL}}, \mathbf{c}^{\mathrm{ROW}}, \mathbf{c}^{\mathrm{COL}} \leftarrow \textsc{coord}(\textsc{lowtri}(\mathbf{C}))$
4:  for $i$ in $\textsc{argsort}(\mathbf{c}^{\mathrm{VAL}})$
5:      $a, b \leftarrow c_i^{\mathrm{ROW}}, c_i^{\mathrm{COL}}$
6:      $r \leftarrow \textsc{urnd}(1, m, \mathsf{h}_a \cup \mathsf{h}_b)$
7:      $j_a \leftarrow \textsc{urnd}(1, k, \emptyset)$
8:      $j_b \leftarrow \textsc{urnd}(1, k, \emptyset)$
9:      $h_{a,j_a}, h_{b,j_b} \leftarrow r$
10: return $\mathbf{H}$
```

advantage of co-occurrence information to adjust the collisions that will inevitably take place when performing the embedding. We denote this approach by co-occurrence-based Bloom embedding (CBE).

What we propose is a quite straightforward approach to CBE, which does not add much extra pre-computation time. Training and testing times remain the same, as CBE uses a pre-computed hashing matrix **H** (Sec. 3.2). The general idea of the proposed approach is to 're-direct' the collisions of the co-occurring items to the same bits or positions of **u**. Our implementation of this idea is detailed in Algorithm 1, and briefly explained below.

First, we count pairwise co-occurrences and store them in a sparse matrix **C** (line 1). Next, we threshold **C** by the average item frequency in **X** using the Hadamard product ⊙ and a component-wise sign function (line 2). We then get the lower triangular part of **C** and return it in coordinates format, that is, using a tuple of values, row indices, and column indices (line 3). We will use the order in $\mathbf{c}^{\mathrm{VAL}}$ to update the hash matrix **H**. To do so, we first loop over the indices of the sorted values of $\mathbf{c}^{\mathrm{VAL}}$ in increasing order (line 4). After selecting the corresponding items a and b (line 5), we then draw integers from URND (lines 6–8). The function $\textsc{urnd}(x, y, \mathsf{z})$ is a uniform random integer generator between x and y (both included) such that the output integer is not included in the set z, that is, $\textsc{urnd}(x, y, \mathsf{z}) \notin \mathsf{z}$. Rows a and b of **H** are transformed to sets h_a and h_b and its union is computed (line 6). Finally, we use the integers generated by URND to pick projections j_a and j_b from **H**, and assign them the same bit r (line 9). By updating the projections in **H** in increasing order of co-occurrence (line 4), we give priority to the pairs with largest co-occurrence, setting them to collide to the same bit r (line 9).

6.2 CBE results

Overall, the performance of CBE only provides moderate increments over the original BE approach (Fig. 4). With the exception of the BC task, the performance of CBE is always higher than the one of BE. However, with the exception of the AMZ task, we do not observe dramatic increases of CBE over BE. On average, such increases are between 0.4% and 8.4% (Table 4, right). One possible explanation for these moderate performance increases is the low co-occurrence in the considered data (Table 4, left). As it can be seen, typically less than 3% of all possible pairs show a co-occurrence. Moreover, the average co-occurrence count of such co-occurring pairs is very low, with ratios ρ to the total number of instances n in the order of 10^{-5} or 10^{-6}.

Despite being moderate on average, we observed that the increments provided by CBE were more prominent for low dimensionality ratios m/d. By relating CBE with the best approaches resulting from the previous comparison of Table 3, we see that CBE is generally better than BE, sometimes with a statistically significant difference (Table 5). Furthermore, we see that CBE, being based on co-occurrences, more closely approaches PMI and CCA in the tasks where those were performing best, and even outperforms

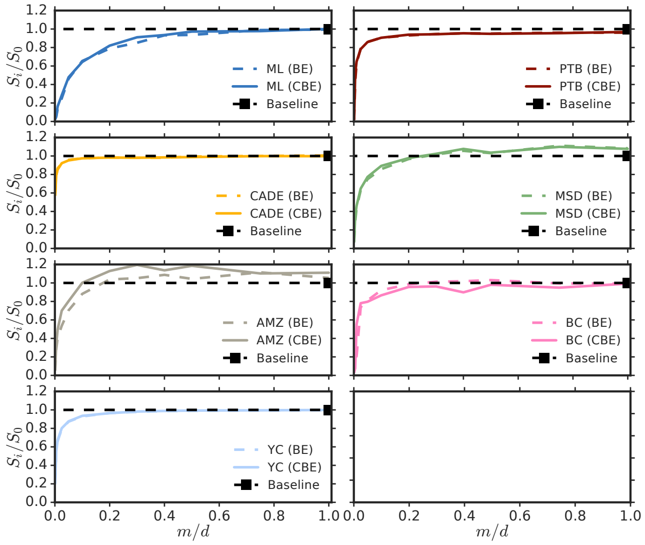

Figure 4: Comparison of score ratios S_i/S_0 as a function of dimensionality ratio m/d for BE (dashed lines) and CBE (solid lines) using $k = 4$. Qualitatively similar plots are observed for other values of k.

Table 4: Co-occurrence statistics and average score increase of CBE over BE. From left to right: data set name, input percent of co-occurrent pairs (%), input average co-occurrence ratio of co-occurrent pairs (ρ), output percent of co-occurrent pairs (%), output average co-occurrence ratio of co-occurrent pairs (ρ), and average score increases of CBE over BE (%; calculated using $100(S_j - S_i)/S_0$ and averaging over all m/d points). Co-occurrence values for PTB and YC inputs correspond to considering training sequences, not isolated sequence items.

Data set	Co-occurrence statistics				Score increase	
	Input		Output		(%)	
	%	ρ	%	ρ	$k = 3$	$k = 4$
ML	25.2	$1.3 \cdot 10^{-4}$	32.9	$1.0 \cdot 10^{-4}$	+0.9	+1.7
PTB	3.3	$2.4 \cdot 10^{-5}$	0	0	+0.1	+0.9
CADE	1.3	$8.8 \cdot 10^{-5}$	N/A	N/A	−0.4	−0.1
MSD	1.3	$3.0 \cdot 10^{-6}$	1.3	$3.1 \cdot 10^{-6}$	+0.5	+1.5
AMZ	3.0	$1.8 \cdot 10^{-6}$	3.0	$1.8 \cdot 10^{-6}$	+6.6	+8.4
BC	0.8	$4.9 \cdot 10^{-5}$	0.4	$4.9 \cdot 10^{-5}$	−3.4	−1.0
YC	0.2	$1.5 \cdot 10^{-6}$	0	0	+0.4	+0.3

them in one test point (AMZ, $m/d = 0.2$). Being closer to those co-occurrence-based approaches is an indication that CBE leverages co-occurrence information to some extent.

7 CONCLUSION

We have proposed the use of Bloom embeddings to represent sparse high-dimensional binary-coded inputs and outputs. We have shown that a compact representation can be obtained without compromising the performance of the original neural network model or, in some cases, even increasing it by a substantial factor. Due to the compact representation, the loss function and the input and output layers deal with less parameters, which results in faster training times. The approach compares favorably with respect to the considered alternatives, and offers a number of further advantages such as on-the-fly operation or zero space requirements, all this without introducing changes to the core network architecture, task configuration, or loss function.

Table 5: Comparison of CBE versus the results in Table 3. Score ratios S_i/S_0 for different combinations of data set and compression ratio m/d. Best results are highlighted in bold, up to statistical significance (Mann-Whitney-U, $p > 0.05$).

Test point		Best so far		CBE	
Data set	m/d	Method	S_i/S_0	$k = 3$	$k = 4$
ML	0.2	BE	0.770	0.760	**0.781**
ML	0.3	BE	0.815	0.812	**0.867**
PTB	0.2	BE	**0.919**	**0.915**	**0.907**
PTB	0.4	BE	**0.942**	**0.937**	0.922
CADE	0.01	PMI	**0.984**	0.854	0.853
CADE	0.03	PMI	**1.002**	0.921	0.922
MSD	0.05	BE	0.738	**0.759**	0.756
MSD	0.1	BE	0.841	0.856	**0.873**
AMZ	0.1	CCA	**1.030**	0.994	0.991
AMZ	0.2	CCA	1.048	**1.109**	**1.117**
BC	0.05	BE	**0.837**	0.774	0.808
BC	0.1	BE	**0.965**	0.880	0.878
YC	0.03	BE	0.858	0.871	**0.880**
YC	0.05	BE	0.928	0.933	**0.936**

In the future, besides continuing to exploit co-occurrences, one could enhance the proposed approach by considering further extensions of Bloom filters such as counting Bloom filters [9]. In theory, those extensions could provide a more compact representation by breaking the binary nature of the embedding. However, they could require the modification of the loss function or the mapping process (Eqs. 2 and 3). A faster mapping process using the sorted probabilities of **v** could also be studied. A detailed, comparative analysis of false positives and false negatives is also pending. Finally, it would be interesting to assess the utility of BE in combination with classical collaborative filtering approaches or factorization machines [40].

ACKNOWLEDGMENTS

We thank the curators of the data sets used in this study for making them publicly-available and Santi Pascual for his comments on a previous version of the paper.

REFERENCES

[1] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid. 2015. Label-embedding for image classification. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 38, 7 (2015), 1425–1438.

[2] Y. Amit, M. Fink, N. Srebro, and S. Ullman. 2007. Uncovering shared structures in multiclass classification. In *Proc. of the Int. Conf. on Machine Learning (ICML)*. 17–24.

[3] G. Armano, C. Chira, and N. Hatami. 2012. Error-correcting output codes for multi-label text categorization. In *Proc. of the Italian Information Retrieval Conf. (IIR)*. 26–37.

[4] S. Bengio, J. Weston, and D. Grangier. 2010. Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems (NIPS)*. Vol. 23. 163–171.

[5] Y. Bengio, R. Ducharme, and P. Vincent. 2000. A neural probabilistic language model. In *Advances in Neural Information Processing Systems (NIPS)*, T. K. Leen, T. G. Dietterich, and V. Tresp (Eds.). Vol. 13. 932–938.

[6] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere. 2011. The million song dataset. In *Proc. of the Int. Soc. for Music Information Retrieval Conf. (ISMIR)*. 591–596.

[7] B. H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.

[8] J. Blustein and A. El-Maazawi. 2002. *Bloom filters – a tutorial, analysis, and survey.* Technical Report. Faculty of Computer Science, Dalhousie University, Halifax, Canada. https://www.cs.dal.ca/research/techreports/cs-2002-10

[9] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. 2006. An improved construction for counting Bloom filters. In *European Symposium on Algorithms (ESA)*, Y. Azar and T. Erlebach (Eds.). Lecture Notes in Computer Science, Vol. 4168. Springer-Verlag, Berlin, Germany, 684–695.

[10] A. Cardoso-Cachopo. 2007. *Improving methods for single-label text categorization.* Ph.D. Dissertation. Instituto Superior Tecnico, Universidade Tecnica de Lisboa.

[11] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. 2015. Compressing neural networks with the hashing trick. In *Proc. of the Int. Conf. on Machine Learning (ICML)*. 2285–2294.

[12] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. 2016. Wide & deep learning for recommender systems. In *Proc. of the Workshop on Deep Learning for Recommender Systems (DLRS)*. 7–10.

[13] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. 2014. On the properties of neural machine translation: encoder-decoder approaches. In *Proc. of the Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST)*. 103–111.

[14] F. Chollet. 2016. Information-theoretic label embeddings for large-scale image classification. ArXiv: 1607.05691. (2016).

[15] M. Cissé, N. Usunier, T. Artières, and P. Gallinari. 2013. Robust Bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems (NIPS)*. 1851–1859.

[16] M. Courbariaux, Y. Bengio, and J.-P. David. 2015. BinaryConnect: training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems (NIPS)*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). 3123–3131.

[17] T. G. Dietterich and G. Bakiri. 1995. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research* 2 (1995), 263–286.

[18] P. C. Dillinger and P. Manolios. 2004. Bloom filters in probabilistic verification. In *Proc. of the Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*. 367–381.

[19] J. Duchi, E. Hazan, and Y. Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12 (2011), 2121–2159.

[20] K. Ganchev and M. Dredze. 2008. Small statistical models by random feature mixing. In *ACL Workshop on Mobile Language Processing (MLP)*. 19–20.

[21] X. Glorot, A. Bordes, and Y. Bengio. 2011. Deep sparse rectifier neural networks. In *Proc. of the Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*. 315–323.

[22] E. Grave, A. Joulin, M. Cissé, D. Grangier, and H. Jégou. 2016. Efficient softmax approximation for GPUs. ArXiv: 1609.04309. (2016).

[23] A. Graves. 2013. Generating sequences with recurrent neural networks. ArXiv: 1308.0850. (2013).

[24] S. Han, H. Mao, and W. J. Dally. Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*. arXiv:1510.00149

[25] F. M. Harper and J. K. Konstan. 2015. The MovieLens datasets: history and context. *ACM Trans. on Interactive Intelligent Systems* 5, 4 (2015), 19.

[26] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*. arXiv:1511.06939

[27] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory networks. *Neural Computation* 9, 8 (1997), 1735–1780.

[28] H. Hotelling. 1936. Relations between two sets of variates. *Biometrika* 28, 3-4 (1936), 321–377.

[29] D. Hsu, S. M. Kakade, and T. Zhang. 2012. A spectral algorithm for learning hidden Markov models. *J. Comput. System Sci.* 78, 5 (2012), 1460–1470.

[30] D. J. Hsu, S. M. Kakade, J. Langford, and T. Zhang. 2009. Multi-label prediction via compressed sensing. In *Advances in Neural Information Processing Systems (NIPS)*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta (Eds.). Vol. 22. 772–780.

[31] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*. arXiv:1511.06530

[32] D. P. Kingma and J. L. Ba. 2015. Adam: a method for stochastic optimization. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*. arXiv:1412.6980 https://arxiv.org/abs/1412.6980

[33] J. Langford, L. Li, and A. Strehl. 2007. *Vowpal wabbit online learning project.* Technical Report. http://hunch.net/?p=309

[34] C. D. Manning, P. Raghavan, and H. Schütze. 2008. *Introduction to information retrieval.* Cambridge University Press, Cambridge, UK.

[35] J. McAuley, R. Pandey, and J. Leskovec. 2015. Inferring networks of substitutable and complementary products. In *Proc. of the ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*. 785–794.

[36] T. Mikolov. 2012. *Statistical language models based on neural networks.* Ph.D. Dissertation. Brno University of Technology.

[37] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean. 2013. Efficient estimation of word representations in vector space. arXiv: 1301.3781. (2013).

[38] M. Mitzenmacher and E. Upfal. 2005. *Probability and computing: randomized algorithms and probabilistic analysis.* Cambridge University Press, Cambridge, UK.

[39] F. Morin and Y. Bengio. 2005. Hierarchical probabilistic neural network language model. In *Proc. of the Int. Workshop on Artificial Intelligence and Statistics (AISTATS)*. 246–252.

[40] S. Rendle. 2010. Factorization machines. In *Proc. of the IEEE Int. Conf. on Data Mining (ICDM)*. 995–1000.

[41] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. V. N. Vishwanathan. 2009. Hash kernels for structured data. *Journal of Machine Learning Research* 10 (2009), 2615–2637.

[42] F. Strub, R. Gaudel, and J. Mary. 2016. Hybrid recommender system based on autoencoders. In *Proc. of the Workshop on Deep Learning for Recommender Systems (DLRS)*. 11–16.

[43] T. Tieleman and G. Hinton. 2012. Lecture 6.5-RMSprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning 4, 2. (2012).

[44] J. Turian, L. Ratinov, and Y. Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*. 384–394.

[45] P. Vincent, A. Brébisson, and X. Bouthilier. 2015. Efficient exact gradient update for training deep networks with very large sparse targets. In *Advances in Neural Information Processing Systems (NIPS)*. 1108–1116.

[46] K. Weinberger, A. Dasgupta, J. Attenberg, J. Langford, and A. Smola. 2009. Feature hashing for large scale multitask learning. In *Proc. of the Int. Conf. on Machine Learning (ICML)*. 1113–1120.

[47] J. Weston, S. Bengio, and N. Usunier. 2010. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine Learning* 81, 1 (2010), 21–35.

[48] J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. Vapnik. 2002. Kernel dependency estimation. In *Advances in Neural Information Processing Systems (NIPS)*, S. Becker, S. Thrun, and K. Obermayer (Eds.). Vol. 15. 873–880.

[49] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester. 2016. Collaborative denoising auto-Encoders for top-N recommender systems. In *Proc. of the ACM Int. Conf. on Web Search and Data Mining (WSDM)*. 153–162.

[50] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. 2005. Improving recommendation lists through topic diversification. In *Proc. of the Int. World Wide Web Conf. (WWW)*. 22–32.