Clustering is Efficient for Approximate Maximum Inner Product Search

Alex Auvolat [*] [†]
alex.auvolat@ens.fr

Pascal Vincent [*] [‡]
vincentp@iro.umontreal.ca

Abstract

Locality Sensitive Hashing (LSH) techniques have recently become a popular solution for solving the approximate *Maximum Inner Product Search* (MIPS) problem, which arises in many situations and have in particular been used as a speed-up for the training of large neural probabilistic language models.

In this paper we propose a new approach for solving approximate MIPS based on a variant of the k-means algorithm. We suggest using *spherical k-means* which is an algorithm that can efficiently be used to solve the approximate *Maximum Cosine Similarity Search* (MCSS), and basing ourselves on previous work by Shrivastava and Li we show how it can be adapted for approximate MIPS.

Our new method compares favorably with LSH-based methods on a simple recall rate test, by providing a more accurate set of candidates for the maximum inner product. The proposed method is thus likely to benefit the wide range of problems with very large search spaces where a robust approximate MIPS heuristic could be of interest, such as for providing a high quality short list of candidate words to speed up the training of neural probabilistic language models.

1 Introduction

The *Maximum Inner Product Search* (MIPS) problem has recently received increased attention, as it appears naturally in classification tasks with a very large number of classes, that are now more commonly tackled. This is a common situation in NLP tasks where the large number of classes is the vocabulary size. This often poses computational challenges. For example in neural probabilistic language models [1] the probabilities of a next word given the context of the few previous words is computed in the last layer of the network as a multiplication of the last hidden layer representation with a very large matrix (an embedding dictionary) that has as many columns as there are words in the vocabulary. Each such column corresponds to the embedding of a vocabulary word in the hidden layer space. Thus an inner product is taken between each of these and the hidden representation, to yield an inner product "score" for each vocabulary word. Passed through a softmax nonlinearity, these yield the predicted probabilities for all possible words. The ranking of these probability values is unaffected by the softmax layer, so finding the k most probable words is exactly equivalent to finding the ones with the largest inner product scores, i.e. solving a k-MIPS problem.

Formally, given a set $\mathcal{X} = \{x_1, \ldots, x_n\}$ of points and a query vector q, the k-MIPS problem is described by:

$$\text{argmax}^{(k)}_{i \in 1,\ldots,n} q^\top x_i$$

[*]DIRO, Université de Montréal, 2920 chemin de la Tour, Montréal, Québec, H3T 1J8, Canada
[†]Département d'Informatique, École Normale Supérieure, 45 rue d'Ulm, 75005 Paris, France
[‡]CIFAR

where the $\text{argmax}^{(k)}$ notation corresponds to the set of the indices providing the k maximum values. Such a problem can be solved exactly in linear time by calculating all the $q^\top x_i$ and selecting the k maximum items, but such a method is too costly to be used on large applications where we typically have hundreds of thousands of entries in our vocabulary and embedding dictionary.

All the methods discussed in this article are based on the notion of a *candidate set*, i.e. a small subset of the dataset on which we do an exact k-MIPS, making the computation much faster. There is no guarantee that the candidate set contains the target elements, therefore these methods solve *approximate* k-MIPS. Better algorithms provide us with candidate sets that are both smaller and have larger intersections with the actual k maximum inner product vectors.

Popular approaches for approximate k-MIPS are based on *Locality Sensitive Hashing* (LSH) [2, 3]. We suggest a different approach that is not based on LSH but on an adaptation of the spherical k-means clustering algorithm [4]. One fundamental difference between our approach and hashing techniques is that our approach is *data-dependent*, meaning that the clustering adapts itself to the geometry of the data.

In section 2 we discuss related work such as hierarchical softmax, the use of *Locality Sensitive Hashing* (LSH) for approximate k-MIPS, as well as previous work by Shrivastava and Li [2] on which our approach is based. In section 3 we show how k-means clustering can be adapted to solve the approximate k-MCSS problem directly and the approximate k-MIPS problem by using the transformation of [2]. We then introduce in section 4 a hierarchical k-means variant that can be used to accelerate the searches over large databases, as well as make them more precise. In section 5 we provide experimental evidence showing that our method is better than WTA or SRP hashes for 1-MIPS, 10-MIPS and 100-MIPS, and that it degrades more gracefully when the query point is far away from any of the datapoints. In section 6 we briefly discuss possible applications of our method to speeding up the training of NLP models with large vocabularies.

2 Related work

2.1 Hierarchical softmax and clustering approaches for large vocabularies

A first approach to solving the large softmax problem is the hierarchical softmax approach introduced in [5] and evaluated in [6]. This approach is based on a non data-dependant clustering of the words into a binary [5], or more generally n-ary tree that serves as a fixed structure for the learning process of the model. Each node of the tree has a weight vector and a bias which are adjusted as a part of the learning. The complexity of the learning is reduced from $O(n)$ to $O(\log n)$.

Common clustering approaches used for defining the fixed initial structure of hierarchical softmax include frequency-based clustering as well Brown clustering and are described in [7]. These clustering techniques produce fixed tree data structures that are not always compatible with the geometry of the word embeddings that we want to learn. A better approach has been suggested in [8] where the clustering is obtained by first training a model with very low-dimension embeddings and doing top-down clustering on those embeddings, yielding a data structure better adapted to the learning of full-fledged embeddings. The quality of the model is also vastly improved by allowing words to appear in several locations of the tree.

On the face of its hierarchical tree structure, hierarchical softmax appears very similar to the MIPS techniques we will explore in this paper. So we draw the attention of the reader to the fact that their rationale and workings are fundamentally different. Hierarchical softmax defines the probability of a leaf node as the *product* of all the probabilities given by all the intermediate softmaxes on the way to that leaf node (i.e. probability of belonging to a specific high-level group, and to a specific subgroup within that group, etc. . .). This results in a nicely normalized, exact and cheap to compute probability for the leaf node. But as a hierarchical softmax imposes this hierarchical group structure *in the definition of the probabilistic model*, it differs substantially from a regular large flat softmax probability model: it is much more inflexible. We hypothesize that this may be the reason why using hierarchical softmax has often been reported to lead to decreased model quality in practice, and why so many practitioners instead choose to pay the hefty computational price of a large flat softmax. By contrast the outlook of the approximate MIPS search is to impose no such arbitrary structure on the probabilistic *model*. Instead a single flexible flat softmax may be kept to train and use. In the MIPS outlook, the hierarchical structure will be employed *only* as a heuristic to lighten the computational burden, not to change or constrain the actual probabilistic model.

2.2 Sampling based approaches to the large vocabulary problem

The outlook of approximate MIPS is akin to techniques that each time heuristically sample a smaller candidate subset of the output nodes to evaluate, as e.g. in the *biased importance sampling* approach of [9, 10], or when employing *Noise Contrastive Estimation* [11] as proposed in [12]. But while these techniques sample blindly, MIPS approaches have the potential to bias the sampling, and the learner's attention, in a smarter way: towards nodes that are more fit contenders, likely to score high.

2.3 Locality sensitive hashing for approximate k-MIPS

Several *Locality Sensitive Hashing* (LSH) based techniques have recently been gaining in popularity [2, 3, 13, 14] as a solution for the large vocabulary learning problem. These techniques are based on a hashing of the embedding vectors into bins that group together vectors which share a common geometrical property. The candidate set is simply the bin in which the query point is located. The two approaches we have chosen to consider for our comparison are *Sign Random Projection* (SRP) hashes [2] and *Winner Take All* (WTA) hashes [3].

The definition of the bins, i.e. the geometry of the clustering, is not data-dependant in the hashing approach, since the projections (in the case of SRP) or the permutations (in the case of WTA) are decided in advanced as a fixed parameter of the data structure. We believe this to be an important limitation of LSH techniques which the k-means method that we introduce does not have.

2.4 MIPS to MCSS transformation

In previous work by Shrivastava and Li [2], a method is introduced for reducing the MIPS problem to the *Maximum Cosine Similarity Search* (MCSS) problem by ingeniously rescaling the vectors and adding new components, making the norms of all the vectors approximately the same. As our new approach is based on this transformation, we will present its basic definition and properties in this section, and refer to [2] for the details.

Let $\mathcal{X} = \{x_1, \dots, x_n\}$ be our dataset. Let $U < 1$ and $m \in \mathbb{N}^*$ be parameters of the algorithm (all our experiments were made with $m = 3$ and $U = 0.85$). The first step is to scale all the vectors in our dataset by the same factor such that $\max_i ||x_i||_2 = U$.

We then apply two mappings P and Q, one on the datapoints and another on the query vector. These two mappings simply concatenate m new components to the vectors making the norms of the datapoints all be roughly the same. The mappings are defined as follows:

$$\begin{aligned} P(x) &= [x, 1/2 - ||x||_2^2, 1/2 - ||x||_2^4, \dots, 1/2 - ||x||_2^{2^m}] \\ Q(x) &= [x, 0, 0, \dots, 0] \end{aligned}$$

As shown in [2], mapping P brings all the vectors to roughly the same norm: we have $||P(x_i)||_2^2 = m/4 + ||x_i||_2^{2^{m+1}}$, with the last term vanishing at the tower rate when $m \to +\infty$, since $||x_i||_2 \leq U < 1$. We thus have the following approximation of MIPS by MCSS for any query vector q,

$$\text{argmax}_i^{(k)} q^\top x_i \quad \simeq \quad \text{argmax}_i^{(k)} \frac{Q(q)^\top P(x_i)}{||Q(q)||_2 \cdot ||P(x_i)||_2}$$

3 Adapting k-means clustering for maximum inner product search

3.1 Standard k-means for approximate L2 nearest neighbour search

The standard k-means algorithm [15, 16] (Algorithm 1) is a widely used method for clustering datapoints into locality-based clusters. k-means uses a fixed number K of clusters defined by their centroids. The algorithm is a fixpoint algorithm that iterates two steps until convergence: a) assign each datapoint to the cluster whose centroid is the nearest; b) recalculate the centroid of each cluster to be the mean of the points associated to that cluster. The algorithm converges when these two steps don't change the assignment of the points or the centroids of the clusters.

k-means clustering can be used to do an accelerated approximate nearest neighbour search: given a query point, we consider as *candidate set* the set of the points that are in the same cluster as the query point. We then do an exact nearest neighbour search on the points of the candidate set only. We can also consider as candidate set the set of points belonging to the p clusters whose centroids are the nearest to the query point, instead of taking only the points of the one best cluster, making our algorithm better suited for the cases where the query point is at the boundary between several clusters.

3.2 A variant of k-means clustering for approximate k-MCSS

If the datapoints $x_1, \dots, x_n$ have all been scaled to a norm of 1, then the *spherical k-means* algorithm discussed in [4] can be efficiently used to do approximate *Maximum Cosine Similarity Search* (MCSS). Algorithm 2 is a formal specification of the spherical k-means algorithm, where we denote by c_i the centroid of cluster i ($i \in \{1, \dots, K\}$) and a_j the index of the cluster assigned to each point x_j.

Algorithm 1 Standard k-means

```
$a_j \leftarrow \mathrm{rand}(K)$
**while** $c_i$ or $a_j$ changed at previous step **do**
    $c_i \leftarrow \frac{1}{\#\{j|a_j=i\}} \sum_{j|a_j=i} x_j$
    $a_j \leftarrow \mathrm{argmin}_{i\in\{1,\dots,k\}} ||x_j - c_i||$
**end while**
```

Algorithm 2 Spherical k-means

```
$a_j \leftarrow \mathrm{rand}(K)$
**while** $c_i$ or $a_j$ changed at previous step **do**
    $c_i \leftarrow \frac{\sum_{j|a_j=i} x_j}{||\sum_{j|a_j=i} x_j||}$
    $a_j \leftarrow \mathrm{argmax}_{i\in\{1,\dots,k\}} x_j^\top c_i$
**end while**
```

The difference between standard k-means clustering and spherical k-means is that in the spherical variant, the datapoints are clustered not according to their position in the Euclidian space, but according to their direction, which can be visualized as a point on the unit sphere. The initialization of the algorithm is random: each datapoint is associated to a random cluster, and the first step of the algorithm is to recalculate the centroids of the clusters. We have not experimented with more subtle initialization schemes as random initialization has shown sufficient convergence properties.

To find the one vector that has the maximum cosine similarity to query point q in a dataset clustered by this method, we first find the cluster whose centroid has the best cosine similarity with the query vector – i.e. the i such that $q^\top c_i$ is maximal – and consider all the points belonging to that cluster as the candidate set. We then simply take $\mathrm{argmax}_{j|a_j=i}\, q^\top x_j$ as an approximation for our maximum cosine similarity vector. This method can be extended for finding the k maximum cosine similarity vectors: we compute the cosine similarity between the query and all the vectors of the candidate set and take the k best matches. This can be also done with more than one cluster: to make our search more accurate, at the cost of a slightly longer computation, we can consider the points from the p best matching clusters as our candidate set.

3.3 Adapting spherical k-means for approximate k-MIPS

The spherical k-means algorithm solves approximate MCSS which is a different problem than approximate MIPS. However the transformation explained in section 2.4 allows to reduce the MIPS problem to a MCSS problem, with a reasonable approximation. Therefore in order to solve approximate k-MIPS, we can run our spherical k-means clustering algorithm on the vectors $\{P(x_1), \dots, P(x_n)\}$, and have our candidate set for exact k-MIPS be the set of points that fall in the cluster i that maximizes $Q(q)^\top c_i$.

4 Hierarchical k-means for faster and better recall

If we have n points, we will typically cluster our dataset into $\sqrt{n}$ clusters so that each cluster contains approximately $\sqrt{n}$ points, reducing the complexity of the search from $O(n)$ to $O\left(\sqrt{n}\right)$. If we use the single closest cluster as a candidate set, then the candidate set size is of the order of $\sqrt{n}$. But when doing approximate k-MIPS with k very big (for example 100), we will typically want to consider the two or three closest clusters as a candidate set, in order to limit the problems that arise when the query point is close to the boundary between several clusters. This simple approach may not be optimal on datasets with many points, as the candidate set can quickly grow uselessly big, containing many unwanted items. To restrict the candidate set to a smaller count of better targeted items, we would need to have smaller clusters, but then the search for the best matching clusters becomes the most expensive part. To solve this problem we propose an approach where we cluster our dataset into many small clusters, and then cluster the small clusters into bigger clusters, and so on any number of times. Our approach is thus a bottom-up clustering approach.

In the experiment we have done, we cluster our datasets in $n^{2/3}$ first-level, small clusters, and then cluster the centroids of the first-level clusters into $n^{1/3}$ second-level clusters, making our data structure a two-layer hierarchical clustering. This approach can be generalized to as many levels of clustering as necessary.

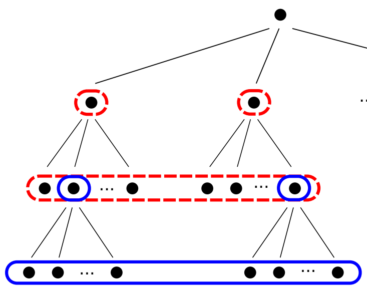

Figure 1: Walk down a hierarchical clustering tree: at each level we have a candidate set for the next level. In the first level, the dashed red boxed represent the p best matches, which gives us a candidate set for the second level, etc.

To search for the small clusters that best match the query point and will constitute an optimal candidate set, we go down the hierarchy keeping at each level only the p best matching clusters. This process is illustrated in Figure 1. Since at all levels the clusters are of much smaller size, we can take much larger values for p, for example $p = 8$ or $p = 16$.

Formally, if we have L levels of clustering, let I_l be a set of indices for the clusters at level $l \in \{0, \ldots, L\}$. Let $c_i^{(l)}, i \in I_l$ be the centroids of the clusters at level l, with $\{c_i^{(L)}\}$ conveniently defined as being the datapoints themselves, and let $a_i^{(l)} \in I_{l-1}, i \in I_l$ be the assignment of the centroids $c_i^{(l)}$ to the clusters of layer $l-1$. The candidate set is found using the method described in Algorithm 3. Our candidate set is the set C_L obtained at the end of the algorithm.

Algorithm 3 Search in hierarchical spherical k-means

```
$C_0 = I_0$
for $l = 0, \ldots, L-1$ do
    $A_l = \text{argmax}^{(p)}_{i \in C_l} q^\top c_i^{(l)}$
    $C_{l+1} = \left\{ i | a_i^{(l+1)} \in A_l \right\}$
end for
return $C_L$
```

In our approach, we do a bottom-up clustering, i.e. we first cluster the dataset into small clusters, then we cluster the small cluster into bigger clusters, and so on until we get to the top level which is only one cluster. Other approaches have been suggested such as in [8], where the method employed is a top-down clustering strategy where at each level the points assigned to the current cluster are divided in smaller clusters. The approach of [8] also addresses the problem that using a single lowest-level cluster as a candidate set is an inaccurate solution by having the datapoints be in multiple clusters.

We use an alternative solution that consists in exploring several branches of the clustering hierarchy in parallel.

5 Experimental results

5.1 Experimental setting

To evaluate our approach, we have chosen to work on the word2vec word embeddings provided by Google researchers [12], as well as the 100 000 word embeddings provided by Collobert and Weston [17]. For the word2vec setting, we have restricted ourselves to the first 100 000 words of the training set, so that the two datasets were the same size $n = 100\,000$. The embeddings provided by Collobert and Weston are of dimension 50, whereas the word2vec embeddings are of dimension 300. We discuss results on word2vec, but the results are similar on Collobert and Weston's embeddings and are available in Appendix B.

The algorithms we have compared are: clustering in $\sqrt{n}$ clusters and taking the 1, 2, 3 best matching clusters; clustering in two layers of $n^{1/3}$ clusters and taking at each level the 2, 4, 8, 16 best matching clusters; WTA hashing; and SRP hashing.

We have tried to adapt the parameters n, p and k of the hashing algorithms so that the candidate set would be of a size comparable to that obtained with the clustering algorithms, which is tricky since with the hashing algorithm we observed that the candidate set size diminishes a lot when we add more noise to the query, but is much more stable with the clustering approaches. We have finally selected $(k, p, n) = (16, 4, 100)$ for WTA hashing and $(p, n) = (16, 100)$ for SRP hashing.

The query vectors we have chosen for testing our algorithms are vectors from the dataset to which we add Gaussian noise of varying magnitude. We have tested all the algorithms for 1-MIPS, 10-MIPS and 100-MIPS. We record the recall frequency, that is the proportion of target vectors (i.e. exact k-MIPS) that are in the candidate set returned by the algorithm, as well as the size of that candidate set.

5.2 Results on recall rate

Table 1 shows recall rates when the query vector is exactly one of the words present in the dataset, and Table 2 shows recall rates when the query vector is a random normal Gaussian vector. We have tried to group the various parameters for the algorithms into groups based on the size of the candidate set (last column of the table), so as to show that for similar candidate set sizes, the clustering approaches provide much better recall rates. The tables also show several cases where the hierarchical clustering version outperforms the one-layer flat clustering approach.

An alternative presentation could have been to group the results by similar recall rate so as to show that the clustering approach provides candidate sets much smaller for the same recall rates, meaning that the clustering is more precise and better adapted to the data.

Figures 2, 3 and 4 (in Appendix A) show the recall rate for 1-MIPS, 10-MIPS and 100-MIPS respectively as a function of the magnitude of the Gaussian noise added to the query vectors. Figure 2 shows that the hashing approaches have nearly perfect recall rate on the one best match task while the noise is small, but once the noise increases the recall rates fall very rapidly, which is not so much the case with the clustering approaches. On the 100-MIPS problem the hashing approaches perform very badly even with no noise.

5.3 Performance considerations

The clustering approach is efficient for the search part as the hierarchical approach yields a complexity of $O(Ln^{1/L})$ when we have L levels of clustering. Clustering is however much slower for the preprocessing part, i.e. when we build the clusters. This is especially true when we do bottom-up clustering for the hierarchical method, where the algorithm basically goes to $O(n^2)$ complexity (to be precise, $O(n^{5/3})$ in the case of the two-layer clustering that we have experimented with). This is due to our hierarchical approach being a bottom-up approach and could be partially solved by using a top-down approach instead, which we have not explored.

Table 1: Query is one of the vectors in the dataset

Method	1-MIPS	10-MIPS	100-MIPS	$\lvert C\rvert$
KM^1_{300}	94.2%	61.6%	47.5%	390
HKM^4	93.4%	**74.3%**	**56 %**	**327**
SRP	**100%**	28.8%	10.2%	333
KM^2_{300}	99.1%	74.9%	63%	775
HKM^8	98%	**85%**	**70%**	**620**
WTA	**100%**	43.8%	19.7%	663
KM^3_{300}	**99.8%**	80.9%	71%	**1159**
HKM^{16}	99.6%	**91.5%**	**81%**	1167

Table 2: Query is a random Gaussian vector

Method	1-MIPS	10-MIPS	100-MIPS	$\lvert C\rvert$
HKM^2	**10.6%**	**8.6%**	**5.8%**	**112**
SRP	1.4%	1.1%	0.9%	176
KM^1_{300}	14.9%	12.8%	9.5%	326
HKM^4	**17.8%**	**14.8%**	**10.3%**	**214**
WTA	2.5%	2.5%	1.9%	315
KM^3_{300}	28.7%	25.6%	20%	988
HKM^{16}	**40.3%**	**34.8%**	**26.0%**	**794**

Recall rate for various algorithms on 1-MIPS, 10-MIPS and 100-MIPS. KM stands for spherical k-means, the subscript is the number K of clusters, the superscript is the number of best matching clusters considered as the candidate set. HKM stands for hierarchical spherical k-means, the superscript is the number p of best matching clusters considered at each level of the hierarchy. WTA stands for Winner Take All hashing. SRP stands for Sign Random Projection hashing. Results are shown for the word2vec embeddings. $|C|$ is the size of the candidate set. Algorithms are grouped by similar candidate set sizes. We observe that for similar candidate set sizes, the clustering approaches vastly outperform the hashing approach, meaning that the provided candidate sets much better match the query.

Future work will investigate how we can adapt the clustering structure for clustering during a learning phase (e.g. as word embeddings evolve), which means that the performance problem will be considered from the different perspective of online clustering.

6 Possible applications of clustering approaches to approximate MIPS

In simple neural probabilistic language models [1], the last layer of the neural network is a large matrix of word embeddings that are matched against the vector output by the previous, hidden, layers. A large softmax operation is done at the end so that the model outputs a probability distribution over all words. The softmax operation is very expensive on large models (with several dozen, or several hundred thousand words) and is often the main computational bottleneck in these methods.

To handle the large vocabulary size and speed up the training, several approaches have been suggested, such as in [9] where the softmax is applied only on a subset of the whole vocabulary, and the backpropagation is equally done also only on that subset (meaning that at any iteration of the algorithm, only the embeddings for a limited number of words will be updated). The approach of [3] uses WTA hashing to select the set of words whose embeddings are to be updated at each iteration. The word embeddings are re-hashed in a rolling fashion, so that each embedding is re-hashed every several thousand iterations.

We suggest that a similar approach can be tried where the set of words used in the softmax and the backpropagation is not the whole vocabulary, but restricted to the candidate set given by one of the clustering algorithms we have introduced, maybe combined with a random sampling approach. Updating the clustering data structure as the word embeddings evolve remains an open problem, but a potential solution may be based on online k-means variants [4, 18].

7 Discussion and conclusion

k-MIPS is a problem that arises in many scenarios where we have to exploit models over a very large number of classes, such as is the case with natural language models. In this paper we have proposed a new and efficient way of solving approximate k-MIPS based on clustering, which we suggest could be a viable alternative to currently used LSH techniques. The advantage of clustering result from the fact that clustering is a data-dependant data structure, meaning that it adapts to the geometry of the data. We have shown that using many small clusters is better than using few large clusters, and have proposed a hierarchical clustering algorithm to deal with the growing computational costs associated with small clusters. Results on a simple benchmark task have shown that our approach is viable and compares favorably with LSH. This is especially true when the query points are not exactly one of the datapoints of the training set, which is the case that will matter most for practical applications inside learning algorithms, i.e. clustering MIPS *generalizes* better to related but unseen data than the hashing approaches we evaluated.

In further research we hope to explore new methods for training large language models using a clustering-based speed-up, which is currently one of the main applications of LSH.

Acknowledgments

The authors would like to thank the developpers of Theano [19,20] for developping such a powerfull tool. We acknowlege the support of the following organizations for research funding and computing support: Samsung, NSERC, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR.

References

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. 3:1137–1155, 2003.

[2] Anshumali Shrivastava and Ping Li. Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (MIPS). arxiv:1410.5410, 2014.

[3] Sudheendra Vijayanarasimhan, Jonathon Shlens, Rajat Monga, and Jay Yagnik. Deep networks with large output spaces. arxiv:1412.7479, 2014.

[4] Shi Zhong. Efficient online spherical k-means clustering. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 5, pages 3180–3185. IEEE, 2005.

[5] Frédéric Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. pages 246–252, 2005.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations: Workshops Track*, 2013.

[7] Yongzhe Shi, Wei-Qiang Zhang, Jia Liu, and Michael T Johnson. RNN language model with word clustering and class-based output layer. *EURASIP Journal on Audio, Speech, and Music Processing*, 2013(1):1–7, 2013.

[8] Andriy Mnih and Geoffrey E. Hinton. A scalable hierarchical distributed language model. pages 1081–1088, 2009.

[9] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. arXiv:1412.2007, 2014.

[10] Y. Dauphin, X. Glorot, and Y. Bengio. Large-scale learning of embeddings with reconstruction sampling. In *Proceedings of the 28th International Conference on Machine learning*, ICML '11, 2011.

[11] M. Gutmann and A. Hyvarinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. 2010.

[12] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS'2013*, pages 3111–3119. 2013.

[13] Anshumali Shrivastava and Ping Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2321–2329. Curran Associates, Inc., 2014.

[14] Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric LSHs for inner product search. arxiv:1410.5518, 2014.

[15] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[16] Stuart Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[17] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[18] Leon Bottou and Yoshua Bengio. Convergence properties of the k-means algorithms. In *Advances in Neural Information Processing Systems 7,[NIPS Conference, Denver, Colorado, USA, 1994]*, pages 585–592, 1994.

[19] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proc. SciPy*, 2010.

[20] Frederic Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. Submited to the Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, http://www.iro.umontreal.ca/ lisa/publications2/index.php/publications/show/551, 2012.

A Complete results on word2vec embeddings

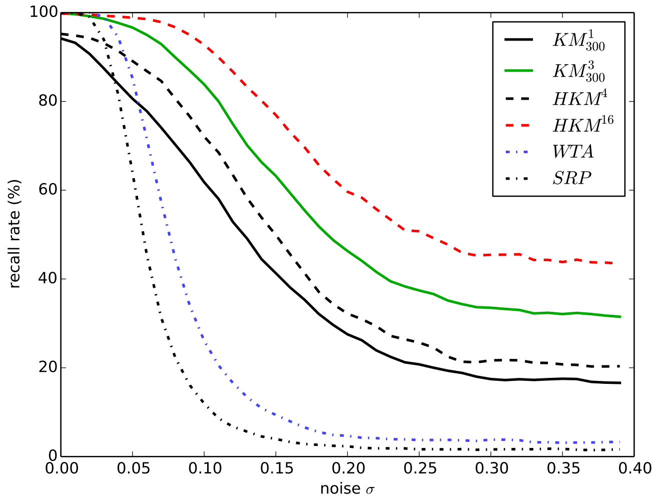

Figure 2: Recall rate degradation with noise magnitude, on top-1 recall task. Although hashing is very efficient in low noise regimes, its efficiency drops drastically and much more rapidly than clustering when the noise increases.

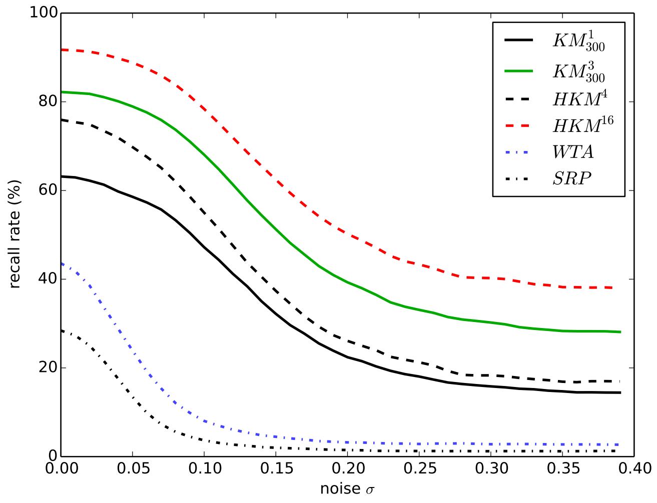

Figure 3: Recall rate degradation with noise magnitude, on top-10 recall task.

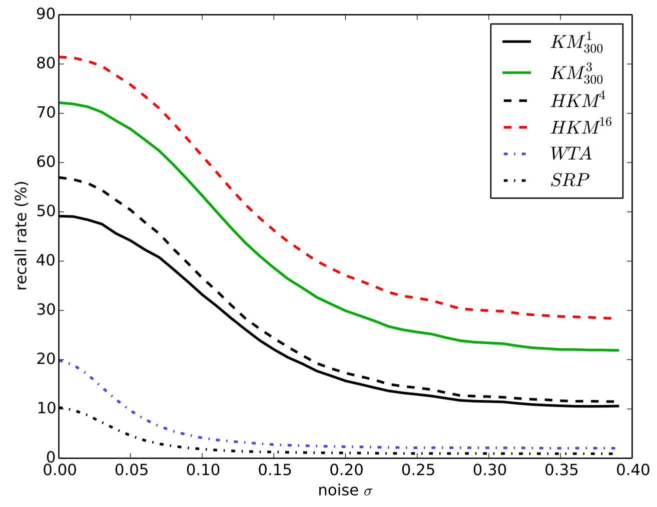

Figure 4: Recall rate degradation with noise magnitude, on top-100 recall task. We observe that hashing is very inefficient for this task even in low-noise regimes, whereas clustering provides robust results.

B Complete results on Collobert and Weston embeddings

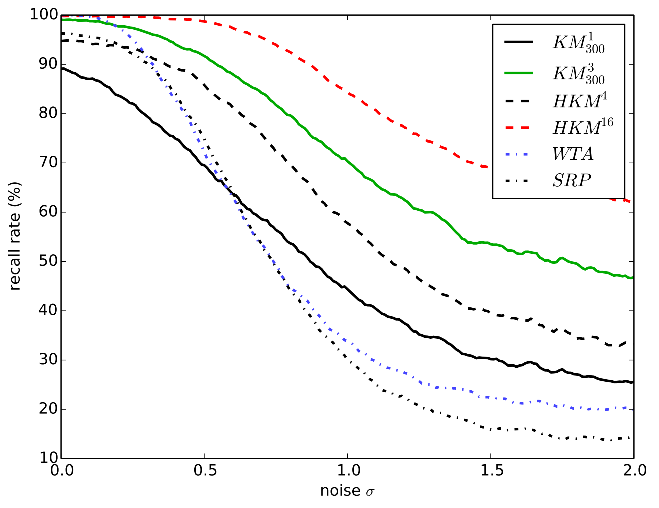

Figure 5: Recall rate degradation with noise magnitude, on top-1 recall task.

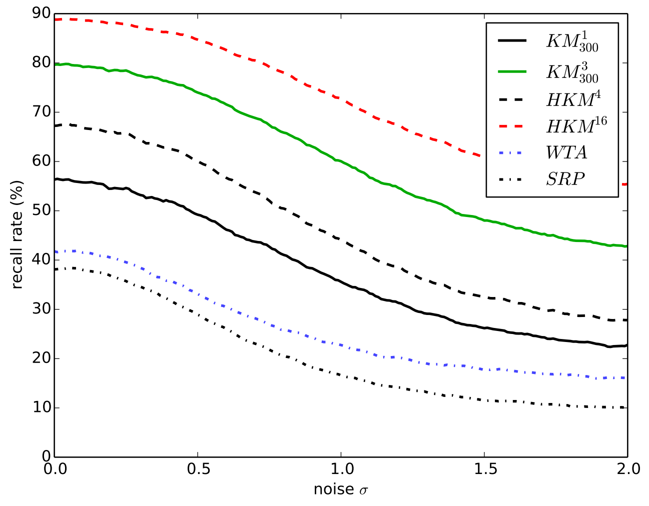

Figure 6: Recall rate degradation with noise magnitude, on top-10 recall task.

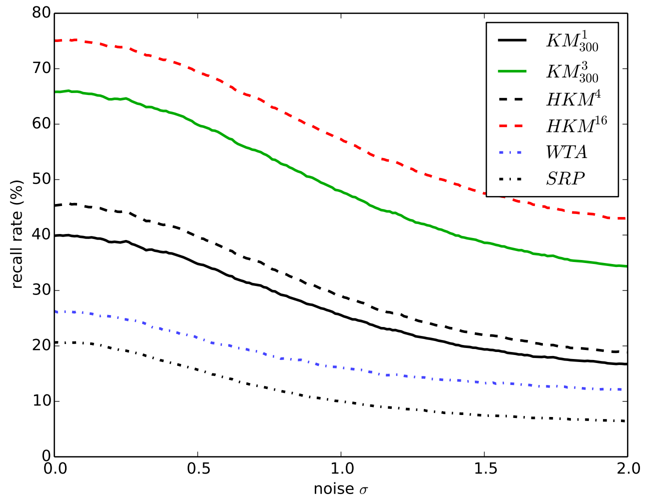

Figure 7: Recall rate degradation with noise magnitude, on top-100 recall task.