A. DWTA Hash: Densified Winner Take All Hash

DWTA (Chen & Shrivastava, 2018) hash maps the data into a transformed space such that the hamming distance between vectors correlates with their rank similarity measure in the original space. Traditional WTA hashing (Yagnik et al., 2011) works by taking a vector x and applying a permutation Θ. The hash value is then taken as the index of the largest of the first R values in the permutation $\Theta(x)$. From here we can use the standard (K, L) LSH parameterization by simply computing the hash on $K \times L$ permutations of the vector, or more efficiently, group a single permutation into $K \times L$ bins and computing the hash as the index of the largest component in each bin. The intuition behind this hashing scheme is that it tends to group vectors based on their largest components, which is useful in this type of learning application in which we are interested in maximizing the inner product between two vectors. Densified WTA hashing (DWTA) is an extension to standard WTA hashing to improve its discriminative power on sparse datasets. Classical WTA may fail on sparse datasets because it becomes likely that some bin of the permuted vector contains no non-zero terms. DWTA uses densification to solve this problem by looking at the bins that contain no non-zeros and taking their hash to be the hash of the nearest bin that does contain a non-zero component. It has been shown (Chen & Shrivastava, 2018) that the collision probability of DWTA is precisely the collision probability of WTA hash for nonempty bins, irrespective of the sparsity.

B. Dataset

Amazon-670K dataset is a product recommendation dataset with 670K labels. Each input is a vector representation of a product, and the corresponding labels are other products (among 670K choices) that a user might be interested in purchase. This is an anonymized and aggregated behavior data from Amazon and poses a significant challenge owing to a large number of classes.

Wiki-325K dataset is extracted from Wikipedia and consists of over 1.7 Million training samples, 1.6 Million sparse feature dimension and 325K labels correspond to the category of the instance.

ODP is extracted from Open Directory Project, a comprehensive human-edited directory of the Web. Each sample in the dataset is a document, and the feature representation is bag-of-words. The class label is the category associated with the document.

Amazon-Uniform is a subsampled version of Amazon-670K and its label distribution is near uniform. More details are in Section 3.5.

C. Sparsity

The number of negative samples C is more like a budget hyperparameter for sparsity e.g. for Amazon-670K, we set sparsity 0.05, meaning $C = 0.05N$, and we keep sampling the buckets from hash tables till the sparsity budget is exhausted and then we stop. Generally, C is independent of N. For example, Wiki-325K has 325k classes and it requires more sparsity (0.1) compared to Amazon-670K which has twice the N (670k) where 0.05 sparsity is sufficient to get the best accuracy. C is dependent on the dataset and the hardness of classification. Also for ODP-105K dataset $C = 0.04N$, and for Amazon-Uniform dataset $C = 0.02N$. Table 4 shows that both versions of our algorithm performs surprisingly well even with extremely low sparsity **(0.005)**, while static-based methods such as Sampled softmax enormously fails, and it requires **at least 0.2** sparsity to hit **even 35%** accuracy.

Table 4. Impact of number of negative samples C on P@1(%) for Amazon-670K dataset.

C	LSH Embedding	LSH Label	Sampled Softmax
$0.005N$	33.4	32	16
$0.05N$	36.1	35.5	32.4

C.1. Impact of Number of Hash Tables

Tables 5 and 6 show the effect of the number of hash tables L on P@1 for Amazon-670K and Wiki-325K datasets. Increasing number of hash tables helps the algorithm to retrieve the informative samples with higher probability, thus improves the accuracy. However, it increases the average training time per epoch proportionally.

Table 5. Impact of Number of Hash Tables on Amazon-670K

Num of Hash Tables	P@1(%)	Avg training time per epoch
$L = 100$	34.5	Baseline
$L = 200$	35.4	0.54x
$L = 300$	35.7	0.36x
$L = 400$	36	0.26x

Table 6. Impact of Number of Hash Tables on Wiki-325K

Num of Hash Tables	P@1(%)	Avg training time per epoch
$L = 100$	53.4	Baseline
$L = 200$	55.4	0.60x
$L = 300$	56.3	0.47x

There is a trade-off between increasing L and running time. Increasing L improves the accuracy, however at some point it reaches a saturation mode where the accuracy improvement is slight comparing to the increase in time.

C.2. Impact of Number of Hash Functions

Table 7 shows the impact of the number of hash functions on accuracy for all datasets with the LSH Embedding scheme. The number of hash functions K determines sampling quality. LNS with an optimal value of K retrieves informative samples, while smaller K retrieves less informative and low-quality samples, and larger values of K leads the algorithm to miss the important and informative samples. For instance, the optimal number of hash functions for Wiki-325K is $K = 5$ which achieves higher accuracy than $K = 4$ or $K = 6$.

Table 7. Impact of number of hash functions K on P@1(%)

Dataset	$K = 4$	$K = 5$	$K = 6$
Amazon-670	-	33.4	36.1
Wiki-325K	38.4	56.3	54.6
ODP-105K	16.5	16.8	15.4
Amaz-Uniform	22.1	23.1	22.8