SOLAR: SPARSE ORTHOGONAL LEARNED AND RANDOM EMBEDDINGS

Tharun Medini[1], **Beidi Chen**[2], **Anshumali Shrivastava**[1]
[1]Rice University, [2]Stanford University
tharun.medini@rice.edu, beidi.chen@stanford.edu, anshumali@rice.edu

ABSTRACT

Dense embedding models are commonly deployed in commercial search engines, wherein all the document vectors are pre-computed, and near-neighbor search (NNS) is performed with the query vector to find relevant documents. However, the bottleneck of indexing a large number of dense vectors and performing an NNS hurts the query time and accuracy of these models. In this paper, we argue that high-dimensional and ultra-sparse embedding is a significantly superior alternative to dense low-dimensional embedding for both query efficiency and accuracy. Extreme sparsity eliminates the need for NNS by replacing them with simple lookups, while its high dimensionality ensures that the embeddings are informative even when sparse. However, learning extremely high dimensional embeddings leads to blow up in the model size. To make the training feasible, we propose a partitioning algorithm that learns such high dimensional embeddings across multiple GPUs without any communication. This is facilitated by our novel asymmetric mixture of **S**parse, **O**rthogonal, **L**earned **a**nd **R**andom (SOLAR) Embeddings. The label vectors are random, sparse, and near-orthogonal by design, while the query vectors are learned and sparse. We theoretically prove that our way of one-sided learning is equivalent to learning both query and label embeddings. With these unique properties, we can successfully train 500K dimensional SOLAR embeddings for the tasks of searching through 1.6M books and multi-label classification on the three largest public datasets. We achieve superior precision and recall compared to the respective state-of-the-art baselines for each task with up to $\mathbf{10}\times$ faster speed.

1 INTRODUCTION

Embedding models have been the mainstay algorithms for several machine learning applications like Information Retrieval (IR) (8; 2) and Natural Language Processing (NLP) (21; 16; 31; 9) in the last decade. Embedding models are learned spin-offs from the low-rank approximation and Matrix Factorization techniques that dominated the space of recommendation systems prior to the emergence of Deep Learning (DL). The primary purpose of these models is to project a rather simple and intuitive representation of an input to an abstract low-dimensional dense vector space. This projection enables two things: 1) tailoring the vectors to specific downstream applications and 2) pre-processing and storing documents or products as vectors, thereby making the retrieval process computationally efficient (often matrix multiplication followed by sorting, which are conducive to modern hardware like GPUs).

Besides the computational advantage, embedding models capture the semantic relationship between queries and products. A good example is product prediction for a service like Amazon. A user-typed query has to be matched against millions of products and the best search results have to be displayed within a fraction of a second. With naive product data, it would be impossible to figure out that products with ‘aqua’ in their titles are actually relevant to the query ‘water’. Rather, if we can project all the products to a dense low-dimensional vector space, a query can also be projected to the same space and an inner product computation can be performed with all the product vectors (usually a dot product). We can then display the products with the highest inner product. These projections can be learned to encapsulate semantic information and can be continually updated to reflect temporal changes in customer preference. To the best of our knowledge, embedding models are the most prevalent ones in the industry, particularly for product and advertisement recommendations (Amazon’s - DSSM (23), Facebook’s DLRM (22)).

However, the scale of these problems has blown out of proportion in the past few years prompting research in extreme classification tasks, where the number of classes runs into several million. Consequentially, approaches like Tree-based Models (26; 15; 1) and Sparse-linear Models (36; 39; 38) have emerged as powerful alternatives. Particularly, Tree-based models are much faster to train and evaluate compared to the other methods. However, most real Information Retrieval systems have dynamically changing output classes and all the extreme classification models fail to generalize to new classes with limited training data (*e.g.*, new products being added to the catalogue every day). This has caused the resurgence of embedding models for large scale Extreme Classification (5; 29; 3; 7).

Our Contributions: In this paper, we argue that sparse, high dimensional, orthogonal embeddings are superior to their dense low dimensional counterparts. In this regard, we make two interesting design choices: 1) We design the label embeddings (*e.g.*products in the catalogue) to be high dimensional, super-sparse, and orthogonal vectors. 2) We fix the label embeddings throughout the training process and learn only the input embeddings (one-sided learning), unlike typical dense models, where both the input and label embeddings are learned. Since we use a combination of **S**parse, **O**rthogonal, **L**earned **a**nd **R**andom embeddings, we code-name our method **SOLAR**. We provide a theoretical premise for SOLAR by showing that one-sided and two-sided learning are mathematically equivalent. Our choices manifest in a five-fold advantage over prior methods:

- **Matrix Multiplication to Inverted-Index Lookup:** Sparse high dimensional embeddings can obtain a subset of labels using a mere inverted-index (8) lookup and restrict the computation and sorting to those labels. This enhances the inference speed by a large margin.
- **Load-balanced Inverted Index:** By forcing the label embeddings to be near-orthogonal and equally sparse (and fixing them), we ensure that all buckets in an inverted index are equally filled and we sample approximately the same number of labels for each input. This omits the well-known imbalanced buckets issue where we sub-sample almost all the labels for popular inputs and end up hurting the inference speed.
- **Lower Embedding Memory:** Dense embedding models need to hold all label embeddings in GPU memory to perform real-time inference. This is not a scalable solution with millions of labels (which is a practical industry requirement). On the contrary, SOLAR needs to store only few integers indices per label which is very memory efficient with modern sparse array support on all platforms. These vectors can also be used with Locality Sensitive Hashing based indexing systems like FLASH (34).
- **Zero-communication:** Our unique construction of label embeddings enables distributed training over multiple GPUs with zero-communication. Hence, we can afford to train on a 1.67 M book recommendation dataset and three largest extreme classification datasets and outperform the respective baselines on all 4 of them on both precision and speed.
- **Learning to Hash:** An Inverted-Index can be perceived as a hash table where all the output classes are hashed into a few buckets (18; 33). By fixing the label buckets and learning to map the inputs to the corresponding label buckets, we are doing a ‘partial learning to hash’ task in the hindsight (more on this in Appendix A).

2 Related Work

SNRM: While there have been a plethora of dense embedding models, there is only one prior work called SNRM (Standalone Neural Ranking Model) (40) that trains sparse embeddings for the task of suggesting documents relevant to an input query (classic web search problem). In SNRM, the authors propose to learn a high dimensional output layer and sparsify it using a typical L1 or L2 regularizer. However, imposing sparsity through regularization causes lopsided inverted-index with imbalanced loads and high inference times. As we see in our experiments later, these issues lead to the poor performance of SNRM on our 1.67M product recommendation dataset.

GLaS: Akin to SOLAR’s construction of near-orthogonal label embeddings, another recent work from Google (11) also explores the idea of enforcing orthogonality to make the labels distinguishable and thereby easier for the classifier to learn. The authors enforce it in such a way that frequently co-occurring labels have high cosine-similarity and the ones that rarely co-occur have low cosine similarity. This imposition was called a **G**raph **L**aplacian **a**nd **S**preadout (**GLaS**) regularizer. However, this was done entirely in the context of dense embeddings and cannot be extended to our case due to the differentiability issue. We show the comparison of SOLAR against dense embedding models with and without GLaS regularizer later on in section 5.1.

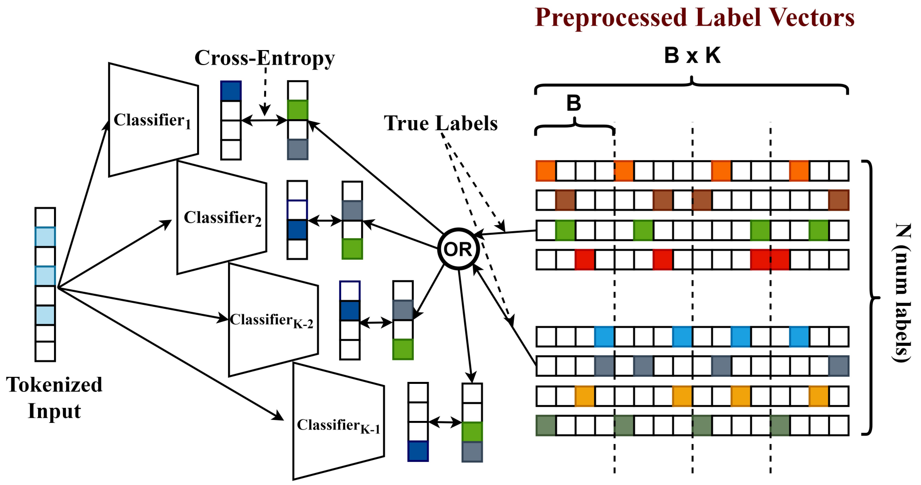


Figure 1: Schematic diagram for label vector construction (on the right) and the training process (on the left). Each label vector is $B \times K$ dimensional divided into K components of length B. Each vector is K-sparse with exactly one non-zero index in each component (colored on the right). The components are separated by dotted vertical lines. For a given input, we perform an 'OR' operation over the true label vectors and feed the resultant pieces to independent small classifiers.

Fix your Classifier: The idea of fixing label vectors was explored in (13; 24). In (13), the authors propose to initialize the last weight matrix of popular CNN architectures with Hadamard matrices and only train the preceding layer weights. With minimal loss in precision, the number of trainable parameters can be greatly reduced. However, 'Fix your Classifier' does not scale to the the huge number of labels in the tasks of our interest as the model cannot be elegantly distributed across independent workers. As shown later in section 5.2, we observe huge performance degradation too with similar network configurations as SOLAR.

Sparsifying Dense Embeddings: Several prior works have proposed to project pre-trained dense embeddings to a sparse high dimensional vectors using techniques like over-complete dictionaries (10), denoising k-sparse auto-encoders (28; 19), permutation maps on unit spheres (4). While all these approaches vindicate the superiority of sparse vectors, none of them learn end-to-end high dimensional sparse vectors and have been confined to tasks with lot fewer labels.

All other embedding models (5; 29; 3; 7) primarily optimize a pairwise similarity-based loss function for query-label pairs, differing in the choice of projection functions. Pairwise training needs negative sampling (12) to avoid degenerate solutions and has large training times as the number of training instances (query-label pairs, both relevant and irrelevant) effectively blows up.

SOLAR, in addition to being sparse, also solves these challenges by learning a classifier instead of a similarity based loss function, encapsulating all labels of an input at once. Since a classifier has intrinsic negative sampling, the number of effective training samples is much lower.

3 Our Method: SOLAR

In this section, we describe in detail the workflow of our algorithm SOLAR. First, we will discuss the pre-processing phase where we construct random sparse label vectors (figure 1) and an inverted-index of the labels (figure 2). Then, we move to the training phase where we split the label vectors into independent contiguous components and train each of them in parallel (figure 1). In the end, we show the inference procedure where we obtain the piece-wise query vector in parallel and sparsify by retaining only top buckets from each piece. We then look up the saved inverted index to retrieve and score the candidate labels to sort and predict the best ones (figure 3).

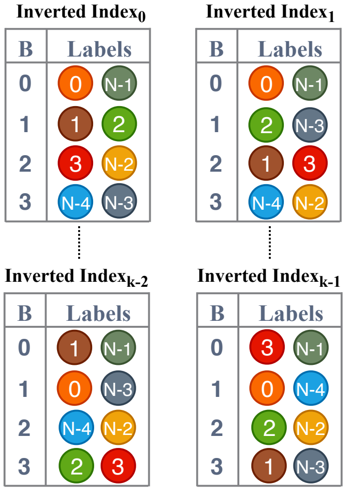
Notations: N denotes the total number of labels. D is the sparse vector dimension. K is the number of non-zeros in label vectors. $B = \frac{D}{K}$ is the number of buckets in each component of the vector.

1) Pre-processing: Construction of Label Embeddings and Inverted-Index: As presented in figure 1, let there be N labels (N is large, in the order of a million). We intend to construct K-sparse (having K non-zero indices) high dimensional vectors for each label. As noted earlier, a large output dimension makes training a cross-entropy loss prohibitively expensive. Therefore, inspired by recent work on zero-communication Model Parallelism (20), we partition the large dimensional vector into K subsets and train each one independently. Each subset of the partition comprises of B buckets with exactly one non-zero index. The colored blocks on the right side in figure 1 denote the non-zero indices for each label vector.

To adhere to our design principle of load-balancing, for each label, we pick the non-zero index randomly in the range of B for each of the K components. To be precise, for any label, we randomly generate K integers in the range of B. As in most of our experiments, set $K = 16$ and $B = 30K$. This makes the overall dimension $D = B \times K = 480K$ and a sparsity ratio of 0.000533 (0.0533%). As an example, let the generated integers be $\{18189, 8475, 23984, ..., 17924, 459\}$. Then the non-zero indices of the overall vector are simply B-shifted , i.e., $\{18189, 38475, 83984, ..., 437924, 450459\}$. Although any random number generator would work fine, we pick our non-zero indices using *sklearn*'s *murmurhash* function. It is rather straightforward to see that these vectors are near-orthogonal. The expected dot-product between any two label vectors l_i and l_j is,

$$E({l_i}^T * l_j) \;=\; \sum_k p(h_k(i) = h_k(j)) \;=\; \frac{K}{B} \approx 0. \tag{1}$$

Figure 2 shows the toy inverted index for the label vectors shown in figure 1. Since we train K independent models, each model is expected to predict its own 'buckets of high relevance'. Hence we maintain K separate inverted-indexes. For any input, we accumulate the candidates from each of the K inverted-indexes and take a union of them for scoring and sorting. It is noteworthy that two unrelated labels might be pooled into the same bucket. While this sounds rather jarring from a learnability perspective, it is essential for the load-balance and also to learn a positive-only association of input tokens and true-label buckets (more on this Appendix B).



Inverted Index$_0$

B	Labels
0	0, N-1
1	1, 2
2	3, N-2
3	N-4, N-3

Inverted Index$_1$

B	Labels
0	0, N-1
1	2, N-3
2	1, 3
3	N-4, N-2

Inverted Index$_{k-2}$

B	Labels
0	1, N-1
1	0, N-3
2	N-4, N-2
3	2, 3

Inverted Index$_{k-1}$

B	Labels
0	3, N-1
1	0, N-4
2	2, N-2
3	1, N-3

Figure 2: Inverted-Index construction for the label vectors shown in figure 1. We construct one index for each of the K chunks. Each bucket will have the same number of labels by design (Load-Balanced).

2) Training: Figure 1 also depicts the training process (on the left side). In a multilabel learning problem, each input has a variable number of true labels. We lookup all the true label vectors for an input and perform an 'OR' operation over the respective sparse label vectors. Please note that at the level of sparsity we are dealing, even with zero pairwise collisions among the non-zero indices of label vectors, we still have a super-sparse representation for the resultant 'OR' vector. We partition this combined-label vector into K parts just like before and train individual classifiers (simple feed forward neural networks with 1 hidden layer) with a binary cross entropy loss function with the B dimensional few-hot vectors. Please note that these models do not communicate with each other. Since there is no overhead of parameter sharing, training can be embarrassing parallellized across multiple GPUs (Zero-Communication Model Parallellism).

Input Feature Hashing: Usually, naive input tokenization like bag-of-words (BoW) leads to a very high dimensional input. This in turn makes the first layer of the network intractable. Hence, an elegant solution for this problem is to hash the token indices to a lower dimension (called Feature Hashing (35)). In our case, we use a different random seed for each of the K models and hash the input indices to a feasible range. Although we lose some input-information in each individual model (due to feature hash collisions), the variation in random seed minimizes this loss when all the models are collectively taken into account.

3) Inference One of the key advantages of SOLAR over dense embedding models is faster inference. As mentioned earlier, the primary reason for this is the replacement of matrix-multiplication and sorting with simple lookups and aggregations. This workflow is depicted in figure 3. Given a tokenized input, we pass it through all the K models in parallel and obtain the respective B dimensional probability vectors. We then sort these probabilities and obtain the top-m (m varies among 50 and 100) buckets for each model. These $m \times K$ integers constitute the non-zero indices of the SOLAR embedding for the input query. We can query the K inverted-indexes with the respective top-m buckets for candidates. A union of all these candidates is our target set of labels. For each of these candidates, we sum up the predicted probability scores from the corresponding buckets and sort for the top results.

Filtering Noisy Labels: Random initialization of embeddings will inevitably result in unrelated labels being assigned the same bucket in any of the K subsets of the partition. Hence, a lot of the candidate labels obtained from the inverted-indexes would be irrelevant to the query. However, it is unlikely that an unrelated label would appear in the top m buckets in more than one of teh K models. Hence, we can omit the labels that appear in the top-m buckets less than a threshold d times. The larger d is, the more precise the retrieved candidates will be. However, setting a large d would mean fewer candidates and thereby lower recall. Later in section 5.1 (in table 3), we experiment with multiple values of d and corroborate that $d = K/2$ is an optimal trade-off between precision and candidate set size.

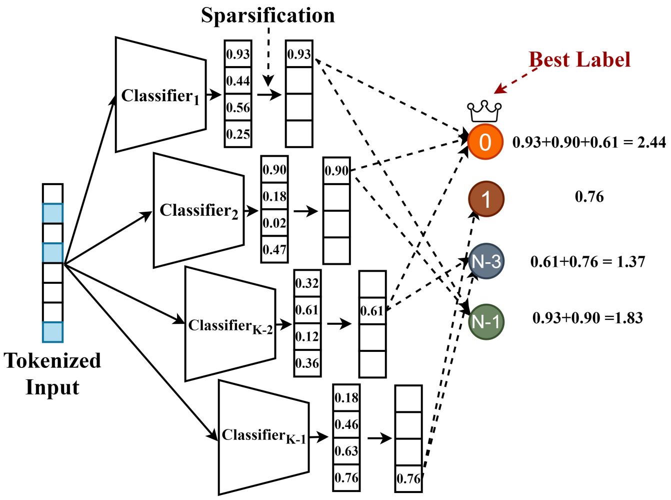


Figure 3: Schematic diagram for Inference. We first get K probability vectors of B dimensions each. Then we only retain the top-m buckets after sparsification (m=1 in above figure. For our experiments, m varies among 50 and 100). We accumulate the candidate labels based on inverted-index for these top-buckets and aggregate their scores and identify the best labels.

Alternate Scoring Methods: It is quite customary to sum up the log of probabilities across the K models for every candidate label as it represents the log-likelihood. Another potential strategy is to sum the logit values directly (as logits have a wider range and are more expressive). In our case, we chose to assign the sum of predicted probabilities for each candidate (as sum is just a scaled version of mean). The rationale behind this is shown in the following analysis assuming a multi-class classification setting.

For an input x, denote $Pr(y = i) = p_i;\ i \in \{0, 1, 2, ..., N\}$. Let $Pr(y = b|\theta_k) = P_b^j;\ b \in \{0, 1, 2, ..., B\}\ and\ k \in \{0, 1, ..., K\}$. Since our hash-functions are random, we have

$$P_{h(i)}^j = p_i + \sum_{k \neq i} 1_{h(k)=h(i)} p_k$$

Hence $E(P_{h(i)}^j) = p_i + \frac{1}{B}\sum_{k\neq i} p_k = p_i + \frac{(1-p_i)}{B}$. After rearrangement, we get

$$p_i = \frac{B * E(P_{h(i)}^j)}{(B-1)} - \frac{1}{B-1}$$

This analysis shows that the original label probabilities are linearly monotonic with expected value of the respective bucket probabilities. And since expected value is proportional to the sum across all K models, summing probabilities is a principled scheme to preserving ranking. While we primarily report the precision with sum of probabilities, we compare it against the other two heuristics (summing log-probabilities and logits) too in our experiments (in table 2).

Time-Complexity: Since our inverted indexes are load-balanced, each bucket accommodates $\frac{N}{B}$ labels. Hence, the top-m buckets contribute $\frac{mN}{B}$ candidates. The candidates retrieved from K models are bound to have some overlapping labels, accompanied by some unrelated counterparts from the respective buckets (since we randomly initialized the label vectors).

In the worst case of zero overlaps, the total number of candidates would be $\frac{KmN}{B}$. The aggregation of scores and frequencies of occurrence in top-m buckets is a by-product of the candidate selection step. We then omit the candidates with frequency < d. Finally, we sort the aggregated scores for the remaining candidates. Including the two sorting steps: 1) to get top-m buckets 2) to get top 5 labels, the total number of operations performed is $B \log m + \frac{KmN}{B} + \frac{KmN}{B} \log 5$. A dense embedding model on the other hand needs $NmK + N \log 5$ steps (assuming dense vectors of dimension $d = mK$, since SOLAR also has mK non-zeros after inference). For the scale of N and mK we are dealing with ($N = 1M$, $K = 16$, $m = 50$, $B = 30K$), SOLAR supposedly has $\frac{B}{1+\log 5}$ times faster inference. However, matrix multiplications have specialized processes with hardware acceleration unlike SOLAR and the practical gains would be in the order of 5x (more in section 5.1).

4 Analysis: Is one-sided learning all we need?

As argued before, we need distributed training to learn ultra-high dimensional embeddings. Our distributed process requires randomly initializing and fixing label embeddings, which might seem to be a questionable approach. However, it turns out that this method of learning is mathematically equivalent to the standard two-sided learning approach with orthogonality constraints. Enforcing such orthogonality in the embedding space for information gain is a common practice (41; 11). More specifically, we prove that the following two processes are mathematically equivalent under any kernel: 1) learning standard two-sided embedding (for both inputs and labels) with orthogonality constraint on the label embedding 2) starting with any fixed orthogonal embedding in the label space and learning the input embeddings. This is a nuanced analogy to the 'Principle of Deferred Decision'.

An embedding model comprises of two functions f_I and f_L which map the inputs and labels respectively to a common vector space. Let X be the set of all inputs and Y be the set of all labels. For some $x \in X$ and $y \in Y$, we seek to optimize f_I and f_L such that the inner product $\langle f_I(x), f_L(y) \rangle$ (or any kernel) is the desired similarity metric $S(x, y)$. Typically, f_I and f_L are neural networks that map tokenized sentences to a vector space and $S(x, y) = \langle f_I(x), f_L(y) \rangle = \frac{f_I(x)^T f_L(y)}{\|f_I(x)\|_2 \|f_L(y)\|_2}$ (cosine similarity). A special case of the embedding models are the popular Siamese Networks (17; 23; 22) in which $f_I = f_L$, i.e., both inputs and outputs learn a shared embedding space.

The imposition of orthogonality on the label vectors supposedly learns a function f_L such that

$$\langle f_L(y_i), f_L(y_j) \rangle = \delta_{ij} \; \forall y_i, y_j \in Y$$

where δ_{ij} is the Kronecker-delta function. Hence, $\{f_L(y_1), f_L(y_2), f_L(y_3), ...\}$ form an orthonormal basis (since we ensure that the learned vectors are unit norm). The following theorem states that both the aforementioned learning methods are equivalent.

Theorem 1. Law of Deferred Orthogonality of Learned Embedding *Given any positive semi-definite kernel $S(.,.)$, and functions f_I and f_L, where f_L is orthogonal. For any orthogonal function R with the same input and range space as f_L, there always exist a function f, such that $S(f_I(x), f_L(y)) = S(f(x), R(y))$.*

Proof. Please refer to Appendix C for detailed proof. The result mainly follows from lemma 2. □

Lemma 1. *For any two orthonormal basis matrices, $\mathbf{A} = [\overline{\mathbf{a_1}} \; \overline{\mathbf{a_2}} \; \overline{\mathbf{a_3}} \; ... \; \overline{\mathbf{a_n}}]$ and $\mathbf{B} = [\overline{\mathbf{b_1}} \; \overline{\mathbf{b_2}} \; \overline{\mathbf{b_3}} \; ... \; \overline{\mathbf{b_n}}]$ in the same vector space, there exists an orthogonal matrix $\mathbf{P}$ such that $\mathbf{A} = \mathbf{PB}$.*

Note that, since we construct fixed norm label vectors, orthogonal vectors form orthonormal basis vectors (up to a constant norm factor multiplication).

Power of Randomness: The popular law of deferred decision in probability allows us to pre-generate random numbers for efficiency. In an analogous fashion, Theorem 2 allows us to pick any orthogonal transformation R in advance and only learn one-sided embedding function f. By exploiting this flexibility we design random, binary, and ultra-sparse label vectors which make training affordable at $500K$ dimensions.

5 Experiments

We now validate our method on two main tasks 1) Product to Product Recommendation on a 1.67M book dataset. This dataset simulates the typical product recommendation task in most search engines.

Model	epochs	P@1	P@5	P@10	Rec@100	Train Time (hrs)	Eval Time (ms/point)
SOLAR (m=100)	10	**35.24**	**29.71**	**26.98**	**34.19**	**2.65**	**0.96**
DSSM (d=1600)	5	31.34	27.55	24.41	32.71	25.27	1.77
GLaS (d=1600)	5	32.51	28.31	25.41	33.17	37.14	1.77
SNRM (d=30K)	5	1.59	2.01	1.93	2.41	-	-
AnnexML (d=800)	10	26.31	22.22	19.37	26.13	16	3.06

Table 1: Comparison of SOLAR against DSSM, DSSM+GLaS, and SNRM baselines. SOLAR's metrics are better than the industry-standard DSSM model while training 10x faster and evaluating 2x faster (SOLAR-CPU vs DSSM-GPU evaluation). GLaS regularizer improves the metrics but still lags behind SOLAR.

scoring method	P@1	P@5	P@10	Rec@100
sum probabilities	35.24	29.71	26.98	34.19
sum log-probabilities	35.89	30.13	27.11	34.69
sum logits	**35.93**	**30.17**	**27.14**	**34.77**

Table 2: Comparison of different scoring schemes. SOLAR inference admits any aggregation scheme that is monotonic as a function of true label probabilities. We observe that summing probabilities, log-probabilities and logits all give nearly the same precision with logits being slightly better.

2) Extreme Classification with the three largest public datasets. Most of the deployed models on modern search engines show promising results on these public datasets in addition to their respective private datasets (25; 14; 20). Additionally, we also train SOLAR embeddings to retrieve a set of target words (unordered) given the source sentence which can perform translation effectively when coupled with a target Language Model with copy mechanism. Please refer to Appendix D for details.

5.1 Product to Product Recommendation

Dataset: This dataset is curated from the raw Amazon Book dataset on extreme classification repository (XML-Repo) (30). This dataset comprises of 1604777 training books whose titles serve as input queries. Each query book is mapped to a set of target books serving as labels (related products recommendation problem). There are a total of 1675657 label books with titles. After parsing both the query titles and label titles, the total vocabulary comprises of 763265 words. The average query length is $\sim$ 9 for both input titles and label titles. There are additionally 693K eval books.

Hyperparameters: As mentioned before, we train 480K dimensional SOLAR embeddings split into $K = 16$ chunks of $B = 30K$ buckets each. The label embeddings are fixed to be exactly 16-sparse while the learned query embeddings are evaluated with 1600 non-zero indices (by choosing $m = 100$ top buckets). We feature hash the 763265-dimensional BOW inputs to 100K dimensions. Each independent model is a feed-forward network with an input layer with 100K nodes, one hidden layer with 4096 nodes, and an output layer with $B = 30K$ nodes. For minimizing the information loss due to feature hashing, we choose a different random seed for each model. Note that these random seeds have to be saved for consistency during evaluation.

Machine and Frameworks: We train with Tensorflow (TF) v1.14 on an DGX machine with 8 NVIDIA-V100 GPUS. We use TF Records data streaming to reduce GPU idle time. During training, we use a batch size of 1000. During inference, except getting the sparsified probability scores, all

min. freq. d	candidates	P@1	P@5	P@10	Rec@100
1	88495.015	34.62	28.50	25.25	25.34
2	2483.83	35.19	29.62	26.85	33.08
4	96.425	35.22	29.65	26.89	33.54
8	**43.885**	**35.24**	**29.71**	**26.98**	**34.19**

Table 3: Filtering Noisy Candidates: We only retain candidates which appear in at-least d of the K models in the top-m buckets. For $K = 16$, $m = 100$ and $B = 30000$, precision and average candidate set size is listed above. Recall improves a lot when we filter out noisy candidates.

other steps are performed on CPU using python's *multiprocessing* module with 48 threads. Our metrics of interest are standard precision and recall. We measure precision at 1,5,10 and recall at 100 (denoted by P@1, P@5,P@10 and Rec@100).

Baselines: The most natural comparison arises with recent industry standard embedding models, namely Amazon's Deep Semantic Search Model (DSSM) (23) and Facebook's Deep Learned Recommendation Model (DLRM) (22). DLRM takes mixed inputs (dense and sparse) and learns token embeddings through a binary classifier for ad prediction. On the other hand, DSSM trains embeddings for semantic matching where we need to suggest relevant products for a user typed query. We compare against DSSM as it aligns with this dataset. Additionally, we impose orthogonality in DSSM using GlaS (11) regularizer. We also compare against other embedding models AnnexML (29) and SNRM (40). Under any reasonable setting, we could not train a 'Fix your Classifier' (FyC) (13) baseline for 1.67M labels due to model memory constraints. However, we do compare against FyC for other datasets in section 5.2. Please refer to Appendix E for baseline settings.

Results: Table 1 compares SOLAR against all the aforementioned baselines. The top row corresponds to the case where we pick the top-100 buckets in each of the 16 models (and hence an 1600 sparse vector and a sparsity ratio of 1600/480K = 0.333%). We notice that on all the metrics, SOLAR is noticeably better than DSSM and its improved variant with a GLaS regularizer. SNRM clearly underperforms due to reasons mentioned in section 2. An interesting point to note is that the evaluation for DSSM was totally done on GPU while SOLAR spends most of its time on CPU. Despite that, SOLAR infers much faster than other baselines.

Table 2 shows the precision performance for different score aggregation schemes. We observe that SOLAR is robust to multiple aggregation strategies as long as the scores are monotonic functions of actual label probabilities. Table 3 also shows the effect of filtering out noisy candidates based on the number of occurrences in the top-m buckets across K models. We can observe that the number of active candidates falls exponentially with d and the precision and recall improve.

Please refer to Appendix E for some sample evaluation queries which show that DSSM works well for frequently queried books while it falters for the tail queries. SOLAR on the other hand is more robust to this phenomenon.

5.2 Extreme Classification Datasets

Let us now shift our focus to the 3 largest multi-label learning datasets available on the XML-Repo, namely Amazon-3M, Amazon-670K and Wiki-500K datasets with 3M, 670K and 500K labels respectively. The statistics of these datasets are available on the repository.

Hyper-parameters: For Amazon-3M, the hyper-parameters remain the same as the book recommendation dataset. For the Wiki-500K and Amazon-670K datasets, we use the latest versions with dense 512-dimensional inputs as outlined in (14). Since the input is now much lower-dimensional than the sparse versions of the same datasets, we train $K = 32$ models in parallel with $B = 20K$, thereby making the overall dimension 640K as opposed to the 480K before. We report the standard P@1, P@3, and P@5 metrics and perform two levels of sparsification for all 3 datasets; $m = 50$ in addition to the previous $m = 100$.

		Embedding Models				Other Baselines			
Dataset	Metric	SOLAR (m=100)	SOLAR (m=50)	AnnexML	SLEEC	FyC	Parabel	Pfastre XML	SLICE
Wiki-500K	P@1	**60.92**	60.52	56.81	30.86	46.86	59.34	55.00	59.89
	P@3	**46.94**	46.56	36.78	20.77	31.29	39.05	36.14	39.89
	P@5	**45.32**	45.28	27.45	15.23	25.17	29.35	27.38	30.12
Amz-670K	P@1	**34.37**	34.19	26.36	18.77	24.47	33.93	28.51	37.77
	P@3	**32.71**	32.51	22.94	16.5	20.44	30.38	26.06	33.76
	P@5	**32.55**	32.46	20.59	14.97	17.13	27.49	24.17	30.70
Amz-3M	P@1	**44.89**	44.61	41.79	-	-	47.51	43.83	-
	P@3	**42.36**	42.08	38.24	-	-	44.68	41.81	-
	P@5	**41.03**	40.69	35.98	-	-	42.58	40.09	-

Table 4: SOLAR vs popular Extreme Classification benchmarks. Embedding models AnnexML and SLEEC clearly underperform compared to SOLAR. SOLAR even outperforms the state-of-the-art non-embedding baselines like Parabel and Slice. The gains in P@5 are particularly huge (45.32% vs 31.57%). SLEEC and SLICE do not scale up to 3M labels (corroborated on XML-Repo).

Dataset		SOLAR (m=100)	SOLAR (m=50)	SLICE	Parabel	Pfastre XML
Wiki-500K	Training (hrs)	2.52	2.52	**2.34**	6.29	11.14
	Eval (ms/point)	1.1	**0.76**	1.37	2.94	6.36
Amz-670K	Training (hrs)	**1.19**	**1.19**	1.92	1.84	2.85
	Eval (ms/point)	2.56	**1.58**	3.49	2.85	19.35
Amz-3M	Training (hrs)	5.73	5.73	-	**5.39**	15.74
	Eval (ms/point)	2.09	1.87	-	**1.72**	4.05

Table 5: Training and Evaluation speeds against the fastest baselines.

Baselines: Since the labels do not have annotations for these datasets, we cannot aspire to train Siamese models like DSSM and SNRM here. Hence we choose to compare against the popular embedding models AnnexML (29) and SLEEC (3) in addition to other Extreme Classification benchmarks like the pure tree-based PfastreXML (15), tree and 1-vs-all model Parabel (25) and also against the recent NN-graph based SLICE (14) which is the state-of-the-art for the first 2 datasets.

Results: Comparison of precision for SOLAR against both embedding and non-embedding baselines are shown in table 4. We use the reported numbers for all available baselines. However, AnnexML and SLEEC do not have reported scores for the 2 smaller datasets with dense inputs. We run the official C++ and MATLAB packages for either of them. It is noteworthy that the training and evaluation for SLEEC are rather slow even though the model size is smaller with dense inputs. Hence, SLEEC could not be scaled to the large 3M class dataset. This fact is independently verified on the repository. It is clear that SOLAR outperforms the state-of-the-art baselines, including SLICE which is noteworthy because SLICE has been very successful on Bing Search for query suggestion (improving the trigger coverage by 52%, mores so for tail queries).

The speed comparison for SOLAR against the fastest baselines is shown in table 5. We can see that SOLAR either matches or surpasses the best ones on both training and inference. Parabel closes in on SOLAR on Amz-3M while SOLAR is much faster on Wiki-500K. PfastreXML is the slowest.

Ablation Study on Choice of B and K: Choosing an appropriate number of partitions K and buckets B is very important as it causes a trade-off between precision and speed. We usually have diminishing gains in precision with increasing B and K at a cost of training and inference speeds. Hence, to strike a good balance, we start with $B = 10K$ buckets and increment by $10K$ until the gains are insignificant. Similarly, we increase K in powers of 2 until convergence in precision performance. Table 6 shows the performance trend for Amazon-670K dataset. We notice that $B = 20K,\ K = 32$ is an optimal setting for this dataset.

	K=8				K=16				K=32			
	P@1	P@3	P@5	Time (ms)	P@1	P@3	P@5	Time (ms)	P@1	P@3	P@5	Time (ms)
B=30K	31.34	29.23	28.45	0.647	33.27	31.34	30.90	0.908	**33.99**	**32.39**	**32.31**	**1.65**
B=20K	30.2	28.25	27.31	0.621	32.55	30.69	29.99	0.96	33.74	32.06	31.78	1.52
B=10K	27.99	25.58	24.36	0.97	29.72	27.85	27.04	1.326	32.07	30.22	29.65	1.39

Table 6: Effect of B, K on $P@1/3/5$ for Amz-670K dataset (with $m = 25$). We increment B linearly and K exponentially and choose an optimal trade-off between precision and inference time (shown in ms/point).

6 Conclusion

This paper proposes SOLAR to learn high-dimensional sparse vectors against dense low-dimensional ones. Sparse vectors are conducive to efficient data structures like inverted-indexes for large scale Information Retrieval. However, training high dimensional vectors has major bottlenecks. Through some elegant design choices, SOLAR ensures that the high dimensional embeddings are trainable in a distributed fashion with Zero-Communication. Additionally, SOLAR enforces near equal inference times for all queries by load-balancing the inverted-indexes. When applied to a multitude of product recommendations and extreme classification datasets, SOLAR outperforms the respective state-of-the-art methods by a large margin on precision and speed.

References

[1] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the 22nd international conference on World Wide Web*, pages 13–24, 2013.

[2] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[3] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. Sparse local embeddings for extreme multi-label classification. In *Advances in neural information processing systems*, pages 730–738, 2015.

[4] Avradeep Bhowmik, Nathan Liu, Erheng Zhong, Badri Bhaskar, and Suju Rajan. Geometry aware mappings for high dimensional sparse factors. In *Artificial Intelligence and Statistics*, pages 455–463, 2016.

[5] Wei Bi and James Kwok. Efficient multi-label classification with many labels. In *International Conference on Machine Learning*, pages 405–413, 2013.

[6] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[7] Yao-Nan Chen and Hsuan-Tien Lin. Feature-aware label space dimension reduction for multi-label classification. In *Advances in Neural Information Processing Systems*, pages 1529–1537, 2012.

[8] W Bruce Croft, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*, volume 520.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint:1810.04805*, 2018.

[10] Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A Smith. Sparse overcomplete word vector representations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1491–1500, 2015.

[11] Chuan Guo, Ali Mousavi, Xiang Wu, Daniel N Holtmann-Rice, Satyen Kale, Sashank Reddi, and Sanjiv Kumar. Breaking the glass ceiling for embedding-based classifiers for large output spaces. In *Advances in Neural Information Processing Systems*, pages 4944–4954, 2019.

[12] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[13] Elad Hoffer, Itay Hubara, and Daniel Soudry. Fix your classifier: the marginal value of training the last weight layer. In *International Conference on Learning Representations*, 2018.

[14] Himanshu Jain, Venkatesh Balasubramanian, Bhanu Chunduri, and Manik Varma. Slice: Scalable linear extreme classifiers trained on 100 million labels for related searches. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019.

[15] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944, 2016.

[16] RichardSocher JeffreyPennington and ChristopherD Manning. Glove: Global vectors for word representation. Citeseer.

[17] Gregory Koch. Siamese neural networks for one-shot image recognition. 2015.

[18] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in neural information processing systems*, pages 1042–1050, 2009.

[19] Alireza Makhzani and Brendan Frey. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

[20] Tharun Kumar Reddy Medini, Qixuan Huang, Yiqiu Wang, Vijai Mohan, and Anshumali Shrivastava. Extreme classification in log memory using count-min sketch: A case study of amazon search with 50m products. In *Advances in Neural Information Processing Systems 32*, pages 13265–13275. 2019.

[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[22] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

[23] Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. Semantic product search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2876–2885, 2019.

[24] Federico Pernici, Matteo Bruni, Claudio Baecchi, and Alberto Del Bimbo. Fix your features: Stationary and maximally discriminative embeddings using regular polytope (fixed classifier) networks. *arXiv preprint arXiv:1902.10441*, 2019.

[25] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference*, pages 993–1002, 2018.

[26] Yashoteja Prabhu and Manik Varma. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 263–272, 2014.

[27] 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.

[28] Anant Subramanian, Danish Pruthi, Harsh Jhamtani, Taylor Berg-Kirkpatrick, and Eduard Hovy. Spine: Sparse interpretable neural embeddings. *arXiv preprint arXiv:1711.08792*, 2017.

[29] Yukihiro Tagami. Annexml: Approximate nearest neighbor search for extreme multi-label classification. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 455–464, 2017.

[30] Manik Varma. Extreme Classification Repository. `http://manikvarma.org/downloads/XC/XMLRepository.html`, 2014.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[32] Jianfeng Wang, Jingdong Wang, Nenghai Yu, and Shipeng Li. Order preserving hashing for approximate nearest neighbor search. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 133–142, 2013.

[33] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):769–790, 2017.

[34] Yiqiu Wang, Anshumali Shrivastava, Jonathan Wang, and Junghee Ryu. Flash: Randomized algorithms accelerated over cpu-gpu for ultra-high dimensional similarity search. *arXiv preprint arXiv:1709.01190*, 2017.

[35] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1113–1120, 2009.

[36] Jason Weston, Ameesh Makadia, and Hector Yee. Label partitioning for sublinear ranking. In *International conference on machine learning*, pages 181–189, 2013.

[37] Song Xu, Haoran Li, Peng Yuan, Youzheng Wu, Xiaodong He, and Bowen Zhou. Self-attention guided copy mechanism for abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1355–1362. Association for Computational Linguistics, July 2020.

[38] Ian EH Yen, Xiangru Huang, Wei Dai, Pradeep Ravikumar, Inderjit Dhillon, and Eric Xing. Ppdsparse: A parallel primal-dual sparse method for extreme classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 545–553, 2017.

[39] Ian En-Hsu Yen, Xiangru Huang, Pradeep Ravikumar, Kai Zhong, and Inderjit Dhillon. Pd-sparse: A primal and dual sparse approach to extreme multiclass and multilabel classification. In *International Conference on Machine Learning*, pages 3069–3077, 2016.

[40] Hamed Zamani, Mostafa Dehghani, W Bruce Croft, Erik Learned-Miller, and Jaap Kamps. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 497–506, 2018.

[41] Xu Zhang, Felix X Yu, Sanjiv Kumar, and Shih-Fu Chang. Learning spread-out local feature descriptors. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4595–4603, 2017.

A Additional Related Work

A.1 Learning to Hash

Learning to Hash (LTH) (33) has become a very integral research area for Near Neighbor Search recently. LTH learns a function $y = f(x)$ for a vector x in such a way that f accomplishes two goals:

1) y is a compact representation of x

2) nearest neighbor of x, x' has $y' = f(x')$ such that y' and y are as close as possible if not same.

The 2^{nd} condition is a restatement of the typical Locality Sensitive Hashing (LSH) (27) constraint. The major difference between LTH and LSH is the fact that f is learned.

Assume a scenario of learning d dimensional vector representations for product search and using Near Neighbor Search to lookup related products. An LSH algorithm maps each vector to a compact hash code. Many similar vectors can have the same hash code (LSH property). For a lookup, we compute the hash code of a query vector and perform a true Near Neighbor Search with the candidates having the same hash code. All LSH algorithms assume the vectors to be distributed throughout the d-space. For example, the typical Simhash or Signed Random Projection (SRP) (6) algorithm samples a few hyper-planes randomly in the d-space. For each label vector, it performs a dot product with all hyper-planes. For each plane, if the dot product of a label vector is ≥ 0, it assigns a bit '1', otherwise '0'. In this way, if we sample P hyper-planes, we get P-bit hash code for every label vector. These label vectors obey the above two properties. However, if the vectors are not distributed uniformly in the d space, simhash miserably assigns a lot of vectors to the same hash code making the Near Neighbor Search almost exhaustive and slow.

Hence, recent LTH Approaches (32) propose to optimize two objective functions: 1) pairwise similarity-preserving 2) bucket balance by maximizing entropy of learned representations. However, no objective function has successfully solved the problem of imbalanced Hash functions.

SOLAR tackles this problem in a unique way by assigning load uniformly to each bucket and then repeating this process multiple times to ensure the distinguishability of dissimilar items. So our hash tables are predetermined and fixed. We learn to map the inputs to respective buckets which might return relevant candidates along with several irrelevant ones too. However, the final step of scoring and sorting eliminates the irrelevant ones and accomplishes the overall functionality of a hash table.

A.2 SNRM

While there have been a plethora of dense embedding models, there is only one prior work called SNRM (Standalone Neural Ranking Model) (40) that trains sparse embeddings for the task of suggesting documents relevant to an input query (classic web search problem). In SNRM, the authors propose to learn a high dimensional output layer and sparsify it using a typical L1 or L2 regularizer. Further, they opt for a weak-labelling strategy wherein each training sample has a query q, documents $d1$ and $d2$ and a binary label l suggesting whether $d1$ is more relevant to q than $d2$ or vice-versa. The query and documents are passed through a Neural Network (RNN for a text input) to obtain their high-dimensional sparse embeddings. These embeddings are concatenated and a hinge-loss is trained to learn the label l. After training, a standard inverted-index is constructed where all the products are partitioned based on the non-zero indexes in their sparse embeddings. During inference, a query is mapped to its sparse vector using the neural network and the non-zero indexes are obtained. All documents that have atleast one matching non-zero index with the query are shortlisted and scored to get the best predictions.

While SNRM looks good to be an efficient sparse alternative to dense embeddings, imposing sparsity through regularization has multiple issues - 1) The training and inference becomes too sensitive to the regularization weight. A larger regularization weight causes the embeddings to be too sparse and in all likelihood, we retrieve zero labels for many inputs. On the other hand, if the regularization weight is very small, we end up retrieving too many candidates defeating the purpose of sparse embeddings. 2) The inverted-index generally has a lopsided label distribution causing imbalanced loads and high inference times. As we see in our experiments later, these issues lead to the poor performance of SNRM on our 1.67M product recommendation dataset.

B Positive-only Association

The random choice of buckets in constructing SOLAR embedding might pool totally unrelated labels into a bucket in each component, For example, two books titled 'Velveteen Rabbit' and 'Gravitational Waves' might be assigned the same bucket in a particular component of the embedding. Even though this pooling appears unconventional, it is necessary to maintain the load balance of each bucket. However, the convergence of cross-entropy loss might come under the scanner in such a scenario.

Going by our training design, we choose to train that specific bucket with '1' if the input query is either related to 'Velveteen Rabbit' or 'Gravitational Waves'. By choosing an 'OR' operation over the true labels, we enforce the model to reasonably fire up the bucket for all queries containing terms 'waves', 'gravity', 'rabbit' etc. This is called 'positive-only association'. Although a single bucket might correspond to unrelated labels, the labels that have high scores in all of the K components would be truly related to the input.

C Law of Deferred Orthogonality of Learned Embedding

Lemma 2. *For any two orthonormal basis matrices,* $\mathbf{A} = [\overline{\mathbf{a_1}}\ \overline{\mathbf{a_2}}\ \overline{\mathbf{a_3}}\ ...\ \overline{\mathbf{a_n}}]$ *and* $\mathbf{B} = [\overline{\mathbf{b_1}}\ \overline{\mathbf{b_2}}\ \overline{\mathbf{b_3}}\ ...\ \overline{\mathbf{b_n}}]$ *in the same vector space, there exists an orthogonal matrix* $\mathbf{P}$ *such that* $\mathbf{A} = \mathbf{PB}$.

Note that, since we construct fixed norm label vectors, orthogonal vectors form orthonormal basis vectors (up to a constant norm factor multiplication).

Proof. Proving the existence is simple. We can perform an orthogonal projection of a_i on to B as:

$$a_i = (a_i \cdot b_1)b_1 + ... + (a_i \cdot b_n)b_n \ = \sum_j b_j(a_i \cdot b_j)$$

Query 1: remington the science and practice of pharmacy 2 volumes	**Query 2:** hello in there a big sisters book of waiting	**Query 3:** beginners guide to american mah jongg how to play the game amp win
SOLAR Top Preds (All Correct)	**SOLAR Top Preds (All Correct)**	**SOLAR Top Preds (only 3 correct)**
pharmacotherapy a patho-physiologic approach 8th edition	im a big sister	winning american mah jongg strategies a guide for the novice player
the sanford guide to antimicrobial therapy sanford guides	you and me new baby	fudge dice black 4 dice in plastic tube
goodman and gilmans the pharmacological basis of therapeutics twelfth edition	look at me new baby	the great mahjong book history lore and play
basic and clinical pharmacology 12e lange basic science	the berenstain bears new baby	beginners guide to american mah jongg how to play the game amp win
GLaS Top Preds (All Correct)	**GLaS Top Preds (All Wrong)**	**GLaS Top Preds (All Correct)**
goodman and gilmans the pharmacological basis of therapeutics twelfth edition	just me and my little brother little critter picturebackr	winning american mah jongg strategies a guide for the novice player
pharmaceutical calculations 13th edition	the day the crayons quit	the great mahjong book history lore and play
basic and clinical pharmacology 12e lange basic science	the invisible boy	beginners guide to american mah jongg how to play the game amp win
the sanford guide to antimicrobial therapy 2013	the name jar	national mah jongg league scorecard large 2014

Table 7: Sample queries and the respective top predictions from SOLAR and DSSM+GLaS. The first query is a relatively frequent one. The second and third are relatively infrequent. We can see that on infrequent queries, SOLAR is more robust than dense embedding models.

Hence, a matrix P can be designed as $P = (P_i^j) = (a_i \cdot b_j)$ to obey $A = PB$. For the orthogonality, consider

$$(PP^T)_i^j = \sum_k (b_j \cdot a_k)(a_k \cdot b_i) = \sum_k (b_j^T a_K)(a_k^T b_i) = b_j^T \left(\sum_k (a_K a_k^T) \right) b_i$$

Since A and B have orthonormal columns, we have $\sum_k (a_K a_k^T) = I$ and $(PP^T)_i^j = b_j^T b_i = \delta_{ji}$. This leads to the orthogonality of P.

$$PP^T = I \implies P^T = P^{-1}$$

□

Theorem 2. Law of Deferred Orthogonality of Learned Embedding *Given any positive semi-definite kernel $S(.,.)$, and functions f_I and f_L, where f_L is orthogonal. For any orthogonal function R with the same input and range space as f_L, there always exist a function f, such that $S(f_I(x), f_L(y)) = S(f(x), R(y))$.*

Proof. Lemma 2 states that we can transform one orthonormal basis to another using an orthogonal matrix. Let the SOLAR's label vectors be denoted by $\mathbf{A} = [\overline{\mathbf{R}(y_1)},\ \overline{\mathbf{R}(y_2)},\ ...,\ \overline{\mathbf{R}(y_N)}]$. Here, $\mathbf{R}$ refers to the random initialization matrix that maps label ids to sparse vectors. Similarly, the two-sided learnt label vectors are denoted by $\mathbf{B} = [\overline{f_L(y_1)},\ \overline{f_L(y_2)},\ ...,\ \overline{f_L(y_N)}]$. Let $\mathbf{P}$ be the transformation matrix from $\mathbf{B}$ to $\mathbf{A}$, i.e., $A_i = R(y_i) = \mathbf{P}B_i = \mathbf{P}f_L(y_i)$. By virtue of orthogonality, $\mathbf{P}$ preserves the inner product objective function as follows:

$$\langle f_I(x), f_L(y) \rangle \ = \ \langle \mathbf{P}f_I(x), \mathbf{P}f_L(y) \rangle \ = \ \langle \mathbf{P}f_I(x), \mathbf{R}(y) \rangle$$

Hence learning one function $f_{SOLAR} \ = \ \mathbf{P}f_I$ is equivalent to learning two function f_I and f_L provided the columns of $\mathbf{B}$ are orthonormal.

□

Method	P@1	P@5	Rec@50	Inference time (ms/point)
SOLAR (m=50)	96.71	83.56	63.007	0.548
Parabel	95.85	77.54	48.03	1.16

Table 8: SOLAR vs Parabel on Word Prediction for French to English translation.

D Machine Translation Experiments

In this section, we make an innovative formulation of machine translation task. Translation can be perceived as a classification task where given the input sentence (source language), we aim to predict the words in the target language. This setup is unpopular mainly because a multi-label classifier is agnostic to the order in which the words are predicted. Hence generating meaningful sentences is not possible by a pure classifier.

However, with the advent of Tranformer models with Copy Mechanism (37), we can train a Target language model that takes in a set of scrambled words and generates meaningful sentences from it. Training such an LM is beyond the scope of this paper. Assuming the existence of such a model, SOLAR can effectively solve the prediction of words challenge.

To re-iterate, we decouple translation into 2 segments: 1) given a source language sentence, we first want to predict all the words in the target sentence using SOLAR (and compare against SOTA Extreme Classification Algorithms) 2) use a pre-trained Target Language Model to generate meaningful sentences from predicted words.

To demonstrate this setup, we perform the word prediction task on French to English WMT dataset. The dataset has 2007723 matched sentence pairs from French to English and 2677 additional pairs for evaluation. The source language French has a vocabulary size of 96270 and the target language English has a vocabulary size of 83213. The mean sentence length in both languages is 35.

We use $B = 5000$ and $K = 8$ to train 40000 dimensional sparse SOLAR vectors. For inference, we use $m = 50$ to obtain a 400-sparse representation for each input sentence and retrieve the most relevant target language words. We additionally train Parabel with the same model size and compare the two on P@1/5 and Recall@50 (we truncate sentences to a maximum of 50 words in both languages). The results are in table 8.

E Additional Information for Experiments

Baselines and our settings: The most natural comparison arises with recent industry standard embedding models, namely Amazon's Deep Semantic Search Model (DSSM) (23) and Facebook's Deep Learned Recommendation Model (DLRM) (22). DLRM takes mixed inputs (dense and sparse) and learns token embeddings through a binary classifier for ad prediction. On the other hand, DSSM trains embeddings for semantic matching where we need to suggest relevant products for a user typed query. Because DSSM's goal aligns with this dataset, we compare SOLAR against it. For DSSM, we train a 763265 × 1600 embedding matrix where each word in the vocabulary has a 1600 dimensional dense vector (to be consistent with the query sparsity of SOLAR). This matrix is shared across input and output titles (Siamese Network). We tokenize a title into words and lookup their embeddings and mean-pool them. This is followed by a batch normalization and $tanh$ activation. The resultant embeddings of both input and label titles are optimized to have high cosine similarity. For every input title, we also train to minimize cosine similarity with one irrelevant label title (Negative Sampling).

We additionally incorporate orthogonality on DSSM label embeddings using the recent GLaS (11) regularizer. We shuffle the entire training data, pick one label from each row and obtain the label vector matrix $\mathbf{V}$. Then, we add a 2^{nd} optimization function (in addition to the cosine similarity) of the form:

$$l_{GLaS} = \lambda \|\mathbf{V}^T\mathbf{V} - \frac{1}{2}(\mathbf{C}\mathbf{Z}^{-1} \ + \ \mathbf{Z}^{-1}\mathbf{C})\|_2^2 \tag{2}$$

where, $\mathbf{C}$ is the label co-occurrence matrix and Z is the diagonal component of $\mathbf{C}$. Hence, $\mathbf{C}\mathbf{Z}^{-1}[i,j] = p(i|j)$ and $\mathbf{Z}^{-1}\mathbf{A}[i,j] = p(j|i)$.

Another natural comparison arises with the lone sparse embedding model SNRM (40). We did try to use the available code for SNRM with our data generator. However, for reasons explained in section A.2, we could not get any reasonable metrics even with very low regularization weight. The training would eventually culminate in the learned vectors being absolutely zero and thereby retrieving nothing from the inverted indexes. Even the label vectors end up being empty most of the time. We included SNRM results in the main paper.

Sample queries and predictions for SOLAR vs DSSM: We perform a qualitative assessment of SOLAR and DSSM+GLaS to assess the robustness to infrequent and spurious queries. For each test query, we sum up the frequency of each term in the corpus and sort the queries. We pick a random query from the top half and classify it as a 'frequent query'. Similarly, we pick a random query from the bottom half and classify it as an 'infrequent query'. We manually examined 10 such queries and listed three of them in table 7. The first query was classified as frequent and the rest two as infrequent. We show the top 4 predictions from either algorithm and the number of ground truth labels among them.