SUPERBLOOM: BLOOM FILTER MEETS TRANSFORMER

Anonymous authors
Paper under double-blind review

ABSTRACT

We extend the idea of word pieces in natural language models to machine learning tasks on opaque ids. This is achieved by applying hash functions to map each id to multiple hash tokens in a much smaller space, similarly to a Bloom filter. We show that by applying a multi-layer Transformer to these Bloom filter digests, we are able to obtain models with high accuracy. They outperform models of a similar size without hashing and, to a large degree, models of a much larger size trained using sampled softmax with the same computational budget. Our key observation is that it is important to use a multi-layer Transformer for Bloom filter digests to remove ambiguity in the hashed input. We believe this provides an alternative method to solving problems with large vocabulary size.

1 INTRODUCTION

In natural language processing, one recent development, made popular by Wu et al. (2016) is to use a smaller sub-word vocabulary (Sennrich et al., 2016), or so called *word piece* model. In such a model, only frequent words and word pieces are kept in the vocabulary. Each word is then segmented as a sequence of word pieces. Both the input and the prediction are then represented in the smaller word piece space.

The word piece model has multiple benefits. Besides its generalizability and compact size, one crucial benefit is that we can afford to compute the full softmax loss on its much smaller vocabulary. This leads to more precise predictions, (measured e.g. using recall at k for small values of k), compared to alternative approaches such as the sampled softmax method (Bengio & Sénécal, 2003; 2008) or the hierarchical softmax (Morin & Bengio, 2005). Word pieces have been shown to work well for natural language understanding (NLU) tasks. For example, the recent break-through of BERT (Devlin et al., 2018) uses a vocabulary of about 30K word pieces. The goal of this paper is to extend this idea to machine learning tasks where we have to model a large number of categorical values, which are represented by opaque ids (e.g. product ids, video ids) or named entities (e.g. Wikipedia or Knowledge Graph entities).

While word pieces are a natural way for breaking up words, it is unclear how this could be done for a set of arbitrary categorical values (referred to as vocabulary throughout the paper). We propose to use random hashing to achieve this goal. Similarly to a Bloom filter (Bloom, 1970), we use multiple hashing functions to map each id to multiple hash tokens in a smaller space. The other motivation of our approach is based on the promise that Transformer models (Vaswani et al., 2017; Devlin et al., 2018) can disambiguate word meanings well using context. A hashed token can be viewed as a word piece with many different meanings. We hope that a Transformer model is also able to remove the ambiguity of hash tokens using the context, i.e. the set of other input tokens.

In this work, we propose *Superbloom* in which we apply a Bloom filter with random hashing scheme to reduce the vocabulary size, then apply a Transformer model to the Bloom filter digest. We demonstrate, through experiments, that Superbloom works well for tasks with a large vocabulary size – it can be efficiently trained and outperforms non-hashed models of a similar size, and larger models trained with sampled softmax with the same computational budget. We highlight the importance of using a multiple-layer Transformer for Bloom filter digests to resolve the ambiguity in the hashed input. For instance, we find that the model quality gap between a one layer and a twelve layer Transformer model is significantly larger when using Bloom filter digests, compared to that when the vocabulary is not hashed. This capability of the Transformer to “unhash” the Bloom digest is a key difference to earlier work on feature hashing (Weinberger et al., 2009) and multiple hashing (Serrà & Karatzoglou, 2017; Svenstrup et al., 2017; Daniely et al., 2017).

1.1 Related work

Learning with a large vocabulary is a well-studied but still open research problem. Weinberger et al. (2009) proposed feature hashing which uses random hashing to reduce the input vocabulary size, and then learns embeddings for hashed ids in the smaller vocabulary. Several follow-up works propose to better resolve collisions by using multiple hashes: Svenstrup et al. (2017) proposed to learn a weighted sum of hashed embeddings; Shu & Nakayama (2018) used an unweighted sum, but proposed instead to learn the hash function itself; and Chen et al. (2018) proposed to learn both the hash function and the combiner, for which they use either a linear function or an LSTM. A key difference with the aforementioned work is that we do not resolve the hashing early at the input of the model; instead, we feed all hashed embeddings to the Transformer and let it learn to resolve the hashing collisions using the context. Our experiments show that multi-layer Transformer models indeed have the capacity to resolve hashing collisions while learning a high quality model.

Besides reducing the input space and memory usage, another set of related work focuses on dealing with large output vocabularies and improving training efficiency. A commonly used method is sampled softmax (Bengio & Sénécal, 2003; 2008) where for each gradient update, only a subset of the output vocabulary is considered. Another line of work is hierarchical softmax where classes are organized in clusters (Goodman, 2001) or in a tree structure (Morin & Bengio, 2005) to allow for efficient pruning of the output vocabulary. Through our experiments, we show that Superbloom, which allows us to train a full softmax on the hashed vocabularies, can lead to more accurate results than using sampled softmax on the larger output vocabulary. Serrà & Karatzoglou (2017) proposed to use Bloom filters as a general tool in deep models, for both the input and output. Our work demonstrates the efficiency of a multi-layer Transformer-like architecture to use contextual information to resolve hash ambiguity. Indeed, we show that shallow models, even with attention, fail.

2 Superbloom Model Architecture

Given discrete sets $\mathcal{S}^I, \mathcal{S}^O$, representing respectively the input and output spaces (e.g. word tokens or entities), the goal is to model a function that maps a sequence of n elements[1] in $\mathcal{S}^I$, to a sequence of probability distributions over $\mathcal{S}^O$. The space of probability distributions over a set $\mathcal{S}$ will be denoted by $\Delta(\mathcal{S}) = \{p \in \mathbb{R}_+^{|\mathcal{S}|} : \sum_{s\in\mathcal{S}} p_s = 1\}$.

The input and output entities are typically represented using embedding matrices $E^I \in \mathbb{R}^{|\mathcal{S}^I|\times d}$ and $E^O \in \mathbb{R}^{|\mathcal{S}^O|\times d}$, which map each entity to an embedding vector of dimension d. This makes training and inference expensive if the number of entities is very large. In order to reduce the model size and improve efficiency, we use a Bloom filter to represent input and output sequences.

A Bloom filter is a probabilistic data structure used to efficiently represent a subset of a given set $\mathcal{S}$. In its simplest form, it can be described by m hash functions $h_j : \mathcal{S} \to \mathcal{H}$, $j \in \{1, \dots, m\}$. To represent a subset $S \subset \mathcal{S}$, one then stores its digest $b(S) = \{h_j(s) : s \in S, j \in \{1, \dots, m\}\}$. Typically, the cardinality of $\mathcal{H}$ is much smaller than $\mathcal{S}$. In Superbloom, each element in $\mathcal{S}^I$ (respectively $\mathcal{S}^O$) is represented by a Bloom filter digest, which allows us to reduce the vocabulary size and thus the size of embedding matrices.

We decompose the Superbloom model architecture into $M = O \circ (T_L \circ \cdots \circ T_1) \circ I$, as illustrated in Figure 1. It consists of three components: an input layer (Sec. 2.1) $I : (\mathcal{S}^I)^n \to \mathbb{R}^{mn\times d}$ which maps each item in the sequence to m embeddings of dimension d; L transformer layers (Sec. 2.2) $T_i : \mathbb{R}^{mn\times d} \to \mathbb{R}^{mn\times d}$ which apply transformations in the embedding space; and an output layer (Sec. 2.3) $O : \mathbb{R}^{mn\times d} \to \Delta(\mathcal{H}^O)^n$ mapping the embeddings to probability distributions. Since the model predicts distributions over $\mathcal{H}^O$ instead of $\mathcal{S}^O$, both training (Sec. 2.4) and inference (Sec. 2.5) need to be adapted accordingly.

2.1 Input layer $I : (\mathcal{S}^I)^n \to \mathbb{R}^{mn\times d}$

The input layer consists of m hash functions $h_j : \mathcal{S}^I \to \mathcal{H}^I$, $j \in \{1, \dots, m\}$ and an embedding matrix $E^I \in \mathbb{R}^{|\mathcal{H}^I|\times d}$. Each element s is mapped to the m embeddings $(E_{h_1(s)}, \dots, E_{h_m(s)})$. The

[1] We assume a fixed sequence length for simplicity. This is also a useful assumption for practical implementation on a TPU, which requires fixed input dimensions.

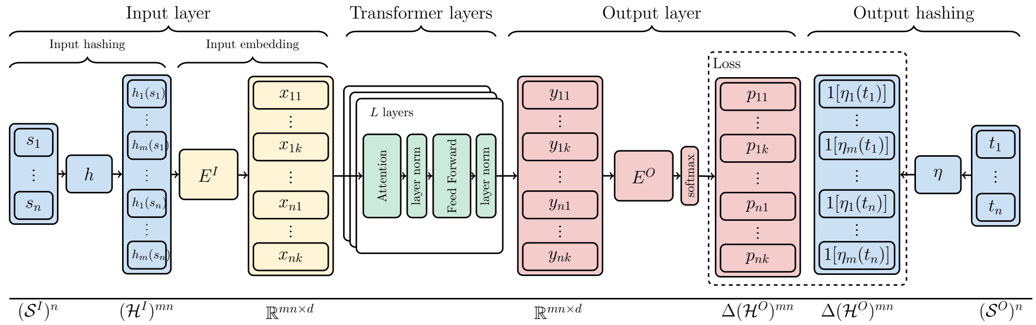

Figure 1: Superbloom model architecture

embeddings for all elements in a sequence $(s_1, \dots, s_n)$ are packed into a matrix $X \in \mathbb{R}^{mn\times d}$. With some abuse of notation, we will identify this matrix with the sequence $\{x_{i,j}\}_{j=1,\dots,m}^{i=1,\dots,n}$, where $x_{i,j} \in \mathbb{R}^d$ is the j-th embedding of element i.

2.2 Transformer layers $T : \mathbb{R}^{mn\times d} \to \mathbb{R}^{mn\times d}$

The Transformer is an attention-based model that was initially proposed for sequence transduction tasks, and that has been used in various other settings such as BERT. For the intermediate layers of Superbloom, we use the same architecture as the original transformer model (Vaswani et al., 2017), which we briefly summarize in Appendix A. Each transformer layer is a function $T : \mathbb{R}^{mn\times d} \to \mathbb{R}^{mn\times d}$ which maps a sequence of mn embeddings in $\mathbb{R}^d$ to another sequence in the same space.

2.3 Output layer: $O : \mathbb{R}^{mn\times d} \to \Delta(\mathcal{H}^O)^{mn}$

Similarly to the input layer, we have m hash functions $\eta_j : \mathcal{S}^O \to \mathcal{H}^O$, $j \in \{1, \dots, m\}$ for the output space. We modify the original goal of predicting distribution over $\mathcal{S}^O$ for the n elements, to predicting the hashes of these elements, i.e. the mn distributions over $\mathcal{H}^O$. More formally, if $Y \in \mathbb{R}^{mn\times d}$ is the output of the last transformer layer, then $O(Y) = \sigma(Y(E^O)^\top) \in \mathbb{R}^{mn\times d}$, where $E^O \in \mathbb{R}^{|\mathcal{H}^O|\times d}$ is an output embedding matrix, and σ is the row-wise softmax function (defined in Appendix A). In some problems, the input and output spaces coincide, so it can be advantageous to use identical input and output hash functions, $h_j = \eta_j$, and the same embedding matrices $E^I = E^O$.

2.4 Training

If the target sequence in $\mathcal{S}^O$ is $(t_1, \dots, t_n)$, then we define mn target distributions in $\Delta(\mathcal{H}^O)$, given by $\{1[\eta_j(t_i)]\}_{j=1,\dots,m}^{i=1,\dots,n}$, where $1[\cdot]$ is the indicator function. Let $\{p_{i,j}\}_{j=1,\dots,m}^{i=1,\dots,n}$ denote the output of the model, and let $\ell : \Delta(\mathcal{H}^O) \times \Delta(\mathcal{H}^O) \to \mathbb{R}$ denote the loss function, e.g. cross-entropy loss. Then the training objective is defined as $\sum_{i=1}^n \sum_{j=1}^m \ell(p_{i,j}, 1[\eta_j(t_i)])$. Note that we can pre-process the training data to map the elements in the original spaces $(\mathcal{S}^I)^n, (\mathcal{S}^O)^n$ to the hash spaces $(\mathcal{H}^I)^{mn}, (\mathcal{H}^O)^{mn}$, and training proceeds entirely in the hash spaces.

Model size and efficiency Compared to a model trained on the original space, the main advantage of Superbloom is a reduction in the size of the embedding matrices E^I, E^O. For instance, if a α-to-one hashing is used (i.e., each hash bucket contains α elements), then $|\mathcal{H}| = |\mathcal{S}|/\alpha$ and the size of the input matrices is reduced by a factor α. This not only reduces the memory cost of the model, but may also improve the efficiency of gradient updates during training. Consider a cross-entropy loss, for each training example, all elements in the output space have a non-zero gradient due to the partition function in softmax, and thus the full matrix E^O needs to be updated at each step, unless approximate methods such as sampled softmax (Bengio & Sénécal, 2003) are used. Our experiments (see Section 3.3) show that the cost of updating E^O dominates that of training, and a reduction in vocabulary size allows us to significantly reduce training time without resorting to negative sampling.

Algorithm 1 Approximate and exact inference in Superbloom

```
1:  Input: Model output $p_j \in \Delta(\mathcal{H}^O)$, $j = 1, \dots, m$, and a beam width $B$.
2:  For each $j \in \{1, \dots, m\}$, find the top $B$ hash values $h \in \mathcal{H}^O$ sorted by $p_j(h)$, call this set $H_j^B$
3:  Let $S^B = \eta_1^{-1}(H_1^B) \cup \dots \cup \eta_m^{-1}(H_m^B)$.
4:  Score all candidates in $S^B$. Let $s^\star = \arg\max_{s \in S^B} \gamma(\rho(s))$.
5:  if Approximate inference then
6:      Return $s^\star$
7:  else
8:      Let $p_j^B$ be the $B$-th largest value in $p_j$, and define $\underline{\gamma}^B := \gamma(p_1^B, \dots, p_m^B)$.
9:      Find $B^\star$ the smallest $B' \geq B$ such that $\underline{\gamma}^{B'} \leq \gamma(\rho(s^\star))$.
10:     Reapply the search with $B$ replaced by $B^\star$, and return the resulting $s^\star$.
```

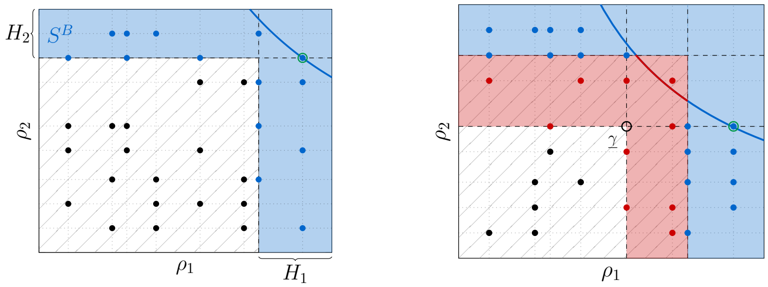

Figure 2: Illustration of approximate and exact inference, with a number of hashes $m = 2$, a four-to-one hashing scheme, and a beam width $B = 2$. The scoring function is $\gamma(\rho) = \log \rho_1 + \log \rho_2$.

2.5 Inference

For each position i in the sequence, the model outputs m distributions $\{p_{i,j}\}_{j=1,\dots m} \in \Delta(\mathcal{H}^O)^m$, which defines a vector of m scores for each candidate $t \in \mathcal{S}^O$, that we denote by $\rho(t) = \{p_{i,j}(\eta_j(t))\}_{j=1,\dots,m}$. We can sort the candidates according to an aggregated score $\gamma(\rho(t))$, where the function $\gamma : \mathbb{R}^k \to \mathbb{R}$ induces a total ordering over $\mathbb{R}^k$. Consider for example $\gamma(\rho(t)) = \sum_j \log \rho_j(t)$, or $\min_j \rho_j(t)$. In order to return the top prediction[2] $\arg\max \gamma(\rho(t))$, one can exhaustively score all elements $t \in \mathcal{S}^O$, which is expensive for a large output space.

Under the natural assumption that the function $\gamma(\cdot)$ is increasing, in the sense that if $\rho \succeq \rho'$ element-wise then $\gamma(\rho) \geq \gamma(\rho')$, we propose a simple algorithm (Algorithm 1) which performs approximate inference efficiently. First, observe that each element ρ_j in the vector ρ can only take $|\mathcal{H}^O|$ values, and it is inexpensive to find hash values that have a high ρ_j, for each j. Given a beam width B, this defines a subset of candidates S^B to score. The second observation is that the set of elements that are *not scored* is exactly the set $\{s \in \mathcal{S} : \rho(s) \preceq p^B\}$ (where p^B is defined on line 8), and under the assumption that γ is increasing, this guarantees that $\gamma(s) \leq \gamma(p^B)$ for all unscored elements. Thus, if $\gamma(\rho(s^\star)) \geq \gamma(p^B)$, we have a certificate that $s^\star$ is indeed the maximizer over all $\mathcal{S}^O$.

An example of this procedure for $m = 2$ and four-to-one collisions (four elements share the same hash value along each dimension) is illustrated in Figure 2. For a beam width of 2, the subset of candidates to score is highlighted in blue. In the left figure, the top element $s^\star$ is circled, and the solid line shows its γ level set. Since this does not intersect the shaded area (unscored elements), we have a certificate that $s^\star$ is the exact maximizer. The right figure shows a different configuration where the $s^\star$ level set does intersect the shaded area. To find the exact maximizer, a second step is performed where we extend the search region (red region). This step is guaranteed to yield the exact maximizer, but may result in searching a much larger set. In Appendix B, we investigate the effect of the beam width on model quality.

The computational cost of approximate inference consists of two parts: first, finding the top B elements in $\mathcal{H}^O$ along each ρ_j (line 2); second, scoring candidates in S^B (line 4), that is, $\mathcal{O}(m|\mathcal{H}^O| \log B + B\alpha)$, which can be significantly cheaper than scoring all candidates $\mathcal{O}(\alpha|\mathcal{H}^O|)$.

[2] In the experiments, we extend the algorithm to return the top-k elements, but we derive it here for $k = 1$ for simplicity.

3 Wikipedia entity prediction

We apply Superbloom to the Wikipedia entity prediction task, in which we use surrounding links on a Wikipedia page to predict a held-out link. This task is derived from the same data set as many NLU tasks, but uses entities instead of natural language. We believe this study is complementary to previous NLU models trained on Wikipedia, that focus on modeling language. Indeed, we show through examples that the model can learn entity relations well and demonstrates a strong use of contextual information.

The task needs to model about 5.3 million entity pages on Wikipedia. This vocabulary size is two orders of magnitude larger than in previous work that applies a Transformer model with full softmax loss (Devlin et al., 2018; Zhang et al., 2018; Sun et al., 2019). Other works, such as Zhang et al. (2019) and Soares et al. (2019), train a Transformer model with a large number of entities using sampled softmax, with either in-batch or in-example negative sampling. But as we shall show, sampled softmax, even with a large number of 128K negative samples, results in much worse quality.

3.1 Task

We take all the entity pages on the website en.wikipedia.org. For each page, we obtain the URL links to other Wikipedia entity pages. We only use "raw" links, i.e. links that explicitly appear on the page. We obtain 5,281,889 pages and 462,588,415 links. Since the Wikipedia site usually removes duplicates of links on each page, the distribution of pages is rather long tail. For example, the top 100 most frequent pages represent only 3.8% of the total links, and the top 10% most frequent pages represent about 60% of the total links.

We hold out 10% random entity pages for testing. For the training data, we apply a masking similar to BERT – from each page, we take a random contiguous segment of entities, of length up to $n = 32$, and mask 15% of the segment. The task is then to predict the masked entities. We also apply the same input perturbation, where for the input, each masked out link is either replaced with a special [MASK] entity (with 80% probabilty), replaced with a random entity (with 10% probability), or left unchanged (with 10% probability). For evaluation, we hold out one random entity from a random segment on a test page. For quality evaluation, we use recall at k metric (abbreviated as rec@k below), which represents the chance the held out entity is in one of the top k predictions.

3.2 Model

To apply Superbloom, we first create m hash maps from entities to hash tokens with a given hash density α. Each hash map is obtained by applying a random permutation to the vocabulary and map every consecutive α entities to the same token. This way we guarantee each hash token to have the same number of collisions α.[3] Special tokens [CLS], [MASK], [SEP], are each mapped to m tokens with no collisions. For example we create $[\text{MASK}_1], .., [\text{MASK}_m]$ tokens corresponding to [MASK].

We apply the hashing to the input and target, to map each entity to m tokens as described in Section 2. We then apply the Transformer model to the input to predict the masked tokens. Unlike in BERT, we do not use position embeddings, in other words, we treat the input as a set instead of a sequence. Since the input and output spaces coincide, we use the same hash functions and the same embedding matrices in the input and output layer.

We carry out experiments on both the full vocabulary as well as a smaller subset consisting of the top 500K entity pages. On the smaller vocabulary, we are able to train a baseline model with large capacity, with no hashing and no sampling, which is useful for understanding the best achievable model quality.

We train all of our models on 16 Cloud TPUs. We use a batch size of 1024 for experiments with full vocabulary and 4096 for experiments with 500K vocabulary. All the experiments use the Adam optimizer (Kingma & Ba, 2014), and use a decreasing learning rate sequence with inverse square root decay, and initial learning rate 1e-4 for the full vocabulary and 2e-4 for the 500K vocabulary. All the experiments have been run for more than 1 million steps to reach near convergence.

[3] The procedure described here is for simplicity. If we are concerned with space, we may use some space efficient methods, for example a perfect hash function (Fredman et al., 1984).

3.3 SUPERBLOOM IS MORE ACCURATE

We experiment with two models of similar size: one is a baseline model (`baseline`) with full vocabulary of size N equal to the number of entities; the other is a Superbloom model (`superbloom`) with a heavy 50 to 1 hashing. We set other hyper-parameters (such as the embedding dimension) so both models have a similar size. We also compare to a large model (`sampled-softmax`) trained using sampled softmax. Table 1 lists the hyper-parameters of each model. Recall that α denotes the number of collisions (1 if there is no hashing), d the embedding dimension, n_A the number of attention heads, d_F the dimension of intermediate hidden layers, and L the number of transformer layers. In all of our experiments, we use two hash functions for Superbloom models. Hence their vocabulary size is $2N/\alpha$.

model	α	d	n_A	d_F	L	#parameters	#samples
baseline	1	48	4	1024	12	248M	5.3M
sampled-softmax	1	512	8	2048	12	2.6G	128K
superbloom	50	768	12	3072	12	229M	200K

Table 1: Model parameters. "#samples" lists the number of samples in the softmax loss computation. For baseline and superbloom, since there is no sampling, this number corresponds to the full vocabulary, 5.3M and 200K, respectively. For sampled-softmax, we use 128K samples.

Table 2 shows the recall metrics of the models. For the Superbloom model, we set the beam width to $B = 20$ (our experiments suggest that it is sufficient to set $B = k$ in order to achieve the best rec@k metric, see Appendix B for details).

model	rec@1	rec@10	rec@20
baseline	36.2%	63.1%	68.2%
sampled-softmax	3.1%	36.2%	55.1%
superbloom	51.1%	72.3%	76.5%

Table 2: Recall metrics for different models.

The Superbloom model clearly outperforms, to a large extent, both the baseline and the sampled-softmax model. We note that the sampled-softmax model has much worse rec@k than the other two models, and this gap is larger for smaller k. This is not surprising given the relatively small percentage (2.5%) of negative examples we can afford to sample.

While the Superbloom model performs well overall, there is a possibility that it devotes most of the embedding capacity to the top entities, so it loses accuracy on the less frequent entities. To test this, we plot the rec@1 value as a function of label frequency. In Figure 3, we show the mean rec@1 for every 10 percentile bucket in terms of the label frequency. We can observe that Superbloom is more accurate than the baseline in all the buckets. Another interesting phenomenon is that the most challenging labels are those in the 20 and 30 percentile. One possible reason is that they lack the higher predictability of the most frequent labels, and also the strong regularity of less frequent labels.

Besides the high predictive accuracy, the prediction from the model shows strong semantic ability and context dependency. We show some examples of predictions in Figure 4 in Appendix D. In one set of examples, we pair "Copenhagen" with different entities, and observe that the predictions change accordingly, depending on the context. Another observation is that despite the heavy hashing, there are almost no unrelated entities in the top 10 predictions. The model even exhibits an ability to perform certain analogy tasks (without being trained on such tasks) – for example, given "Tunisia Tunis Thailand", it predicts "Bangkok" as the top result.

3.4 MULIT-LAYER TRANSFORMER IS IMPORTANT FOR SUPERBLOOM

Intuitively, given the large noise introduced by hashing, it is more important for Superbloom to use multiple attention layers in Transformer to "remove" the noise. To test this intuition, we run experiments with a smaller vocabulary size of the top 500K entity pages (about 60% of the links).

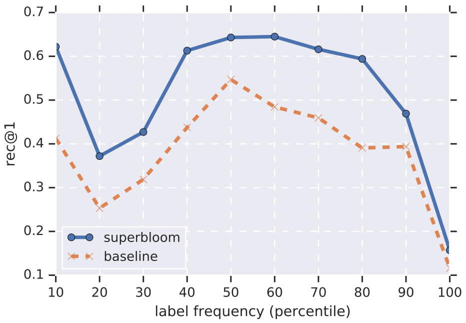

Figure 3: Rec@1 with respect to label frequency, starting from the most frequent labels.

On this smaller vocabulary size, we can afford to run a full softmax model with a larger embedding dimension.

model	α	d	n_A	d_F	L	#parameters	rec@1	rec@10	rec@20
baseline-l1	1	256	1	1024	1	123M	51.0%	70.4%	75.5%
baseline-l12	1	256	8	1024	12	132M	55.0%	73.7%	77.3%
superbloom-d256l1	20	256	1	1024	1	13M	17.8%	35.8%	42.6%
superbloom-d384l1	20	384	1	1536	1	21M	30.6%	52.9%	58.7%
superbloom-d256l12	20	256	8	1024	12	21M	43.4%	60.1%	64.0%

Table 3: Model parameters and recall metrics.

We consider different embedding dimensions and model complexity. Table 3 lists the model parameters as well as the recall metrics for each model. We observe that for the baseline models, the quality difference is small between models of different complexity. For example, rec@1 of baseline-l12 (55.0%) is about 8% better than baseline-l1 (51.0%). Since a one layer Transformer is close to a bag-of-words (BOW) model, one may argue that it may be unnecessary to use a Transformer in this case – instead one can use a larger dimension BOW model to achieve a similar accuracy.

However, for Superbloom models, the quality improves significantly with more layers. When increasing the number of layers from 1 (superbloom-d256l1) to 12 (superbloom-d256l12), rec@1 increases from 17.8% to 43.4%. The multi-layer model also performs much better than the single layer model with the same size (superbloom-d384l1). Note that previous work on hashed vocabularies relies on BOW models, which are less expressive than even a single-layer transformer. This highlights one of our key observations that multi-layer Transformer models are more effective for working with hashed vocabularies.

4 Experiments on natural language data

In this section, we apply Superbloom to natural language data. We consider a large vocabulary that contains frequent unigrams and bigrams and use it to tokenize the text, then apply a Bloom filter to reduce the vocabulary size. We show that despite high hash collisions, the model can achieve high accuracy on natural language data. Since many named entities appear in the large vocabulary, we observe that the model seems to make better predictions of named entities than the BERT model.

While each hash id can be regarded as a word piece in an NLU model, there are important differences between hash ids and word pieces. First, hashing causes random collisions, while wordpiece tokenization can be viewed as a special hashing scheme based on the spelling – there is often coherence between words that share a word piece. As suggested by the experiments in Appendix C, random hashing with Superbloom digests may outperform coherent hashing. In addition, as every token in the large vocabulary is hashed, we do not have unambiguous anchors (such as the exact word pieces) to help bootstrap the disambiguation process. Despite these differences, our experiments suggest that even with high hashing collision $\alpha = 40$, the Transformer is capable of resolving, or unhashing, the Bloom filter digest effectively and produces highly accurate predictions and meaningful embeddings.

We construct a vocabulary of size 1M by taking the union of standard BERT word piece vocabulary ($\sim$ 30K) with the most frequent unigrams and bigrams, and follow the same procedure in BERT to create training examples. For Superbloom, we apply random hash maps to the 1M vocabulary similar to the approach described in Section 3.2 to ensure an even number of collisions. The Superbloom architecture is chosen to have a comparable model size to the baseline BERT model.

We compare four models: For the non-hashed baselines, we have a large model with embedding dimension $d = 256$, and a small model with $d = 64$. And we have two Superbloom models with similar model sizes. We list the parameters in Table 4. In Table 5 we list the recall metrics for the models. We observe that with comparable model size, Superbloom outperforms the baseline model in all the recall metrics, and the improvement is more significant for smaller model size.

model	α	d	n_A	d_F	L	#parameters
baseline-h64	1	64	4	256	12	62.6M
baseline-h256	1	256	8	1024	12	254.4M
hash40-h512	40	512	8	2048	12	62.3M
hash20-h1024	20	1024	16	4096	12	246.3M

Table 4: The model parameters.

model name	rec@1	rec@10	rec@20	model name	rec@1	rec@10	rec@20
baseline-h64	28.4%	44.9%	48.6%	baseline-h256	37.2%	57.4%	63.3%
hash40-h512	31.7%	48.3%	52.9%	hash20-h1024	39.2%	58.5%	64.5%

Table 5: Recall metrics.

Since many named entities are included in the larger vocabulary, the Superbloom model shows that it may have better “understanding” or representation of those entities. We show some anecdotal evidence in Appendix D by comparing predictions of pretrained BERT and Superbloom model on some fill-in-the-blanks examples. The BERT model often predicts generic words, seemingly ignoring other named entities in the sentence. The Superbloom model, on the other hand, can often fill in the blank with related entities.

5 Conclusion

Our experiments show that the multi-layer Transformer is effective for achieving high accuracy on hashed inputs, represented using Bloom filter digests. Besides applying it to tasks with large vocabularies, it also points to a few interesting future research directions.

The Transformer model has been mostly studied in natural language settings and for sequence data. In our setup, we show that it can work effectively with sets of hashed entities. We hope that by investigating this simpler setup, it can help us better understand the properties of the Transformer. For example, due to hashing, each token is similar to words with multiple meanings, so its embedding can be viewed as a combination, possibly linear (Arora et al., 2018), of the embeddings of multiple entities. A multi-layer Transformer model may provide a mechanism for iteratively filtering such noisy representations, using the context. It would be interesting to further study this mechanism.

While hashing adds noise to the learned representations, it can also increase the flexibility of these representations – when we hash multiple entities to the same token, the model is free to allocate the corresponding embedding unevenly among entities, which results in a different effective embedding dimension for each entity. Such learned capacity allocation might be more efficient than using a fixed embedding size or frequency-based allocation. Of course, an effective “denoising” model is a pre-requisite for such an approach to work. Perhaps Superbloom, with its strong denoising ability, can help further realize the potential of embedding models on hashed vocabularies.

REFERENCES

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Linear algebraic structure of word senses, with applications to polysemy. *Transactions of the Association for Computational Linguistics*, 6:483–495, 2018.

Yoshua Bengio and Jean-Sébastien Sénécal. Quick training of probabilistic neural nets by importance sampling. In *Proceedings of the conference on Artificial Intelligence and Statistics (AISTATS)*, 2003.

Yoshua Bengio and Jean-Sébastien Sénécal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *Transactions on Neural Networks*, 19(4):713–722, April 2008. ISSN 1045-9227.

Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

Ting Chen, Martin Renqiang Min, and Yizhou Sun. Learning k-way d-dimensional discrete codes for compact embedding representations. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 854–863, 2018.

Amit Daniely, Nevena Lazic, Yoram Singer, and Kunal Talwar. Short and deep: Sketching and neural networks. In *ICLR Workshop*, 2017.

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

Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, June 1984. ISSN 0004-5411.

Joshua Goodman. Classes for fast maximum entropy training. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '01)*. IEEE, 2001.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In Robert G. Cowell and Zoubin Ghahramani (eds.), *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pp. 246–252. Society for Artificial Intelligence and Statistics, 2005.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

Joan Serrà and Alexandros Karatzoglou. Getting deep recommenders fit: Bloom embeddings for sparse binary input/output networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pp. 279–287, 2017.

Raphael Shu and Hideki Nakayama. Compressing word embeddings via deep compositional code learning. In *ICLR*, 2018.

Livio Baldini Soares, Nicholas FitzGerald, Jeffrey Ling, and Tom Kwiatkowski. Matching the blanks: Distributional similarity for relation learning. *arXiv preprint arXiv:1906.03158*, 2019.

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. *arXiv preprint arXiv:1904.06690*, 2019.

D. T. Svenstrup, J. Hansen, and O. Winther. Hash embeddings for efficient word representations. In *Advances in Neural Information Processing Systems*, 2017.

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*, pp. 5998–6008, 2017.

Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alex Smola. Feature hashing for large scale multitask learning. In *ICML*, 2009.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Shuai Zhang, Yi Tay, Lina Yao, and Aixin Sun. Dynamic intention-aware recommendation with self-attention. *arXiv preprint arXiv:1808.06414*, 2018.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. Ernie: Enhanced language representation with informative entities. *arXiv preprint arXiv:1905.07129*, 2019.

A Transformer architecture

We briefly recall the Transformer architecture following Vaswani et al. (2017). Each transformer layer is a function $T : \mathbb{R}^{mn\times d} \to \mathbb{R}^{mn\times d}$ which transforms a sequence of mn embeddings[4] in $\mathbb{R}^d$ to another sequence of mn embeddings in the same space. T can be decomposed into $T = F \circ A$, where

- A is an attention function,

$$A(X) = \sum_{a=1}^{n_A} \sigma\left((XQ_a)(XK_a)^\top/\sqrt{d}_A\right) XV_aW_a^\top, \tag{1}$$

  where $a \in \{1, \dots, n_A\}$ indexes attention heads, $d_A \le d$ is an internal embedding dimension (usually $d_A = d/n_A$), and for each a, $Q_a, K_a, V_a, W_a \in \mathbb{R}^{d\times d_A}$. Finally, $\sigma : \mathbb{R}^{n\times n} \to \Delta([n])^n$ is the row-wise softmax function, given by

$$\sigma(Y)_{ij} = \frac{\exp(Y_{ij})}{\sum_{l=1}^{n} \exp(Y_{il})}. \tag{2}$$

  One interpretation of the attention function is that it forms the i-th output embedding by taking a convex combination of input embeddings weighted by the softmax weights, followed by a low-rank transformation $V_aW_a^\top \in \mathbb{R}^{d\times d}$.
- F is a fully connected feed-forward network given by $F(X) = \text{ReLU}(XU_1 + b_1)U_2^\top + b_2$, where $U_i \in \mathbb{R}^{d\times d_F}$ for some $d_F \ge d$.

A residual connection and layer normalization are also applied at each stage A, F.

B The quality of beam search

Before we report the recall metrics, we first need to make sure that the beam search is accurate for Superbloom models. Table 6 shows rec@k for $k = 1, 10, 20$ for different beam widths B, using a small number of test examples for the Superbloom model. We observe that to obtain highest rec@k metric, it is sufficient to set the beam width $B = k$ in Algorithm 1.

beam width	rec@1	rec@10	rec@20
B=1	53.0%	56.0%	56.0%
B=10	53.2%	68.2%	69.1%
B=20	53.2%	67.9%	71.0%
B=100	53.2%	67.8%	71.5%

Table 6: Recall metrics at different beam width.

C Comparison of different hashing schemes

We have used random hashing functions in Superbloom. One natural alternative is “coherent” hashing, in which we map similar entities to the same hash bucket. A potential benefit of coherent hashing is that it may use embedding capacity more effectively by sharing it among similar entities. However, the downside is that it becomes difficult to distinguish those similar entities.

To create a coherent hashing function, we first run a co-occurrence factorization algorithm and then group similar entities together using the following procedure, designed to guarantee equal-sized hash buckets. For each entity, in decreasing frequency order, we compute the nearest neighbors (scored using cosine similarity), then create a hash bucket that includes the elements and its $\alpha - 1$ nearest neighbors which have not been already assigned a bucket. When creating a second coherent hash function, we add the constraint that any pair of elements that share a bucket for the first hash function

[4] A minor difference with the original Transformer model is that we operate on $\mathbb{R}^{mn\times d}$ instead of $\mathbb{R}^{n\times d}$, since we have m embeddings for each element in the sequence.

cannot be assigned to the same bucket in the second hash. This ensures that no two elements have the same collision in both hash functions.

We carry out the experiments on the data set with smaller vocabulary (500K). We train different models that all use two hash functions, with the following configurations: both random, one random and one coherent; and both coherent. We also use different hashing densities $\alpha = 10$ and $\alpha = 20$. All the models have the same hyper-parameters as the superbloom-l12 model in Section 3.4. The results are given in the following table.

model	α	#coherent hashing	token rec@1	entity rec@1
hash10-00	10	0	36.32%	52.50%
hash10-01	10	1	38.19%	50.20%
hash10-11	10	2	38.55%	34.70%
hash20-00	20	0	33.39%	43.70%
hash20-01	20	1	36.98%	41.10%
hash20-11	20	2	37.65%	30.20%

Table 7: Random hashing versus coherent hashing.

We observe that with coherent hashing, we get higher accuracy for predicting hash tokens but lower accuracy for predicting entities. And the entity recall@1 is significantly lower when both hash functions are coherent. This indicates that with higher coherence, it becomes increasingly difficult for the model to make finer distinctions between similar items.

D Examples of Wikipedia entity predictions

1. Examples of pairing "Copenhagen" with different entities. The predictions vary according to the context, from Danish cities, to major European cities, to Danish royalty, and Danish culture. There is a one unrelated result (underlined), which disappears in the presence of additional context.

Copenhagen [MASK]
Denmark Oslo Stockholm Paris Berlin Aarhus Danish_language University_of_Copenhagen Sweden Copenhagen

Copenhagen Aarhus [MASK]
Denmark Odense Copenhagen Aalborg Aarhus Oslo Malmö Max_Wilms Stockholm Esbjerg

Copenhagen Paris [MASK]
Berlin Denmark London Oslo Rome Vienna Stockholm New_York_City Brussels Hamburg

Copenhagen Dynasty [MASK]
Denmark Margrethe_II_of_Denmark Danish_language Copenhagen Catholic_Church Rome Christian_V_of_Denmark Jutland When_We_Wake_Up Frederik,_Crown_Prince_of_Denmark

Copenhagen Dynasty Danish_language [MASK]
Denmark German_language Margrethe_II_of_Denmark Catholic_Church Copenhagen English_language Princess_Benedikte_of_Denmark Danish_language Frederik,_Crown_Prince_of_Denmark Christian_V_of_Denmark

2. Examples of Jazz musicians. These relatively long and rare name entities would not appear in the vocabulary of a word piece model.

Miles_Davis [MASK]
Jazz Columbia_Records Miles_Davis John_Coltrane Dizzy_Gillespie Bill_Evans Album Sonny_Rollins AllMusic Charles_Mingus

John_Coltrane [MASK]
Miles_Davis AllMusic Jazz A_Love_Supreme Rolling_Stone Elvin_Jones Albert_Ayler Tenor_saxophone New_York_City Drum_kit

Miles_Davis John_Coltrane [MASK]
Jazz Charles_Mingus Album AllMusic Miles_Davis Dizzy_Gillespie Thelonious_Monk Sonny_Rollins Charlie_Parker Bill_Evans

3. Example showing that the prediction is the set union if two entities are not related.

Miles_Davis Thailand [MASK]
Vietnam Bangkok Japan Miles_Davis Cambodia Malaysia Jazz Indonesia Thai_language Brazil Myanmar Rock_music Dizzy_Gillespie John_Coltrane

4. Examples for completing location analogy task!

Texas Austin,_Texas Florida [MASK]
Miami Houston Orlando,_Florida Dallas Jacksonville,_Florida Fort_Lauderdale,_Florida Tampa,_Florida Georgia_(U.S._state) Tallahassee,_Florida St._Petersburg,_Florida

Tunisia Tunis Thailand [MASK]
Bangkok Philippines Montcau Tokyo Malaysia Singapore Indonesia Pattaya Vietnam Thai_language

Figure 4: Examples of Superbloom model predictions. For each example, we output the top 10 predictions of the model (computed using Algorithm 1 with a beam width $B = 10$). The entity names shown here are obtained by removing the prefix "https://en.wikipedia.org/wiki/" from the entity URL.

E Examples of natural language entity predictions

Miles Davis is a Jazz musician, he is similar to [MASK].

BERT: jazz himself beethoven him davis chopin bowie williams jones
baseline-h256: miles_davis john_coltrane bill_evans charlie_parker louis_armstrong sonny_rollins keith_jarrett thelonious_monk jazz duke_ellington
hash20-h1024: miles_davis john_coltrane charlie_parker thelonious_monk dizzy_gillespie bill_evans billie_holiday duke_ellington humans_is louis_armstrong

Empire state building is an iconic site of [MASK1] , it is close to [MASK2] .

[MASK1]
BERT: architecture chicago manhattan downtown pittsburgh art philadelphia history washington america
baseline-h256: architecture modern_art contemporary_art modern_architecture national_significance new_york art its_day historical_significance the_city
hash20-h1024: the_city new_york lower_manhattan manhattan the_neighborhood downtown wall_street the_area harlem architecture

[MASK2]
BERT: downtown it chicago philadelphia rome london broadway manhattan chinatown campus
baseline-h256: downtown downtown_pittsburgh city_hall new_york the_city times_square columbia_university san_francisco philadelphia the_pentagon
hash20-h1024: central_park city_hall times_square wall_street union_station broadway lower_manhattan the_pentagon fifth_avenue carnegie_hall

Figure 5: Natural language fill-in-the-blank examples. BERT is the base BERT model in Devlin et al. (2018); baseline-h256 and hash20-h1024 are the Superbloom models with 1M vocabulary, with model parameters listed in Table 4.