Fast Amortized Inference and Learning in Log-linear Models with Randomly Perturbed Nearest Neighbor Search

Stephen Mussmann*, **Daniel Levy***, **Stefano Ermon**
Department of Computer Science
Stanford University
Stanford, CA 94305
{mussmann,danilevy,ermon}@cs.stanford.edu

Abstract

Inference in log-linear models scales linearly in the size of output space in the worst-case. This is often a bottleneck in natural language processing and computer vision tasks when the output space is feasibly enumerable but very large. We propose a method to perform inference in log-linear models with sublinear amortized cost. Our idea hinges on using Gumbel random variable perturbations and a pre-computed Maximum Inner Product Search data structure to access the most-likely elements in sublinear amortized time. Our method yields provable runtime and accuracy guarantees. Further, we present empirical experiments on ImageNet and Word Embeddings showing significant speedups for sampling, inference, and learning in log-linear models.

1 INTRODUCTION

Log-linear models are widely used in machine learning and statistics. These models receive their name from the fact that the log unnormalized probabilities are linear in the parameters and the sufficient statistics. Since the probabilities are defined up to scaling, inference and learning require computing the normalization constant, also known as the partition function (Murphy, 2012).

While defining unnormalized probabilities affords modeling flexibility, it comes at the price of computation time. For factorized models, there are many methods, such as Gibbs sampling and variational inference (Koller & Friedman, 2009), to approximately perform inference. Here we are interested in the setting where the output space is not factorizable, and is large but enumerable (e.g., a few million elements). Such problems with large output spaces occur in many areas including computer vision and natural language processing (NLP) (Joulin et al., 2016; Bengio et al., 2003; Mikolov et al., 2013). While inference is tractable (by brute force, in time linear in the size of the output space), it can be a major bottleneck in learning and even at test time in resource-constrained settings. Clearly, computation time cannot be saved for a *single* inference query as it requires linear time to examine the input. However, as Mussmann & Ermon (2016) establishes, computation time can be saved for a *sequence* of related queries, e.g., sampling from log-linear models with the same sufficient statistics but different (changing) parameters. Such sequences of queries arise naturally in learning and at test time.

In this work, we employ Gumbel random variables to convert sampling into maximizing unnormalized log-probabilities perturbed by Gumbel noise, applied independently to each element in the output space (Hazan et al., 2013; Maddison et al., 2014; Kim et al., 2016). Naively, sampling a Gumbel for each element requires linear runtime which yields no savings. However, we introduce a novel way to lazily instantiate the Gumbel random variables. In order to maximize the Gumbel-perturbed objective, we only examine a (small) subset of the most-likely states and a small number of Gumbel perturbations (the largest ones). This yields asymptotic runtime improvements with provable accuracy guarantees. To find the most likely states, which involves the maximization of the dot product between the parameters and the sufficient statistics, we are able to make use of the large literature on Maximum Inner Product Search (Shrivastava & Li, 2014; Auvolat et al., 2015; Douze et al., 2016; Ram & Gray, 2012; Koenigstein et al., 2012).

The contributions of this work are as follows.

- We present a method to perform sampling using access to the top $O(\sqrt{n})$ most likely states (where n is the number of states), using the Gumbel max trick.

*Both authors contributed equally.

- We present a method to estimate the partition function and expected values using access to the top $O(\sqrt{n})$ values and relying on uniform sampling.
- We present a way to use Maximum Inner Product Search (MIPS) techniques to retrieve the approximate top $O(\sqrt{n})$ elements to provably achieve sublinear amortized query time.
- We demonstrate applications of our method in computer vision and NLP where we achieve 5–10$\times$ per-query speedups compared to the naive method.

2 BACKGROUND

2.1 LOG-LINEAR MODELS

Log-linear models are widely used in machine learning and artificial intelligence. Generally, any exponential family distribution can be written as a log-linear model. As examples, very common models like multinomial logistic regression and maximum entropy models (Koller & Friedman, 2009; Murphy, 2012) are log-linear models. Additionally, the last layer of a neural network (softmax) is a log-linear model, e.g. sampling the next element of a sequence for recurrent neural networks.

In this work, we focus on discrete distributions over a set of states $\mathcal{X}$. For a log-linear model, the log unnormalized probabilities are linear in the parameters. More precisely, if the parameters are θ and the features (sufficient statistics) for an element $x \in \mathcal{X}$ are $\phi(x)$, then,

$$\Pr(x; \theta) \propto e^{\theta \cdot \phi(x)} \tag{1}$$

Note that in order to define a distribution, we must normalize these probabilities by

$$Z_\theta = \sum_{x \in \mathcal{X}} e^{\theta \cdot \phi(x)} \tag{2}$$

which is known as the partition function. Unfortunately, computing the partition function Z is expensive as it requires summing over all elements in $\mathcal{X}$. We can also learn a log-linear model by maximizing the likelihood of some training data where evaluating the gradient requires computing the expected value of the sufficient statistics.

Assumption: In our setting, $\mathcal{X}$ is large but feasibly enumerable, so naively computing the partition function is tractable but computationally expensive.

As an example, in the experimental results section, $|\mathcal{X}| \approx 10^6$. As a negative example, Markov Random Fields can be written as a log-linear model but have an exponentially large $\mathcal{X}$ and thus are not amenable to our method.

2.2 GUMBEL VARIABLE

In the context of extremal statistics, Gumbel & Lieblein (1954) defines the Gumbel distribution as

$$\Pr(G < x) = \exp(-\exp(-x)) \tag{3}$$

We can sample a Gumbel random variable using the following scheme,

$$U \sim \text{Uniform}(0, 1) \tag{4}$$

$$G = -\ln(-\ln(U)) \tag{5}$$

Our use of the Gumbel distribution is motivated by the so-called "Gumbel Trick" which involves adding Gumbel noise to the log unnormalized probabilities to turn sampling from a log-linear model into finding the maximizing element.

Proposition 2.1 ((Hazan et al., 2013; Maddison et al., 2014))**.** *For Gumbel variables G_x sampled i.i.d. for each data point x,*

$$\operatorname*{argmax}_x \theta \cdot \phi(x) + G_x \sim \text{Categorical}(\{\frac{e^{\theta \cdot \phi(x)}}{Z_\theta}\}_x) \tag{6}$$

2.3 MAXIMUM INNER PRODUCT SEARCH

A common computational task is retrieving the nearest neighbor to a query from a database of vectors. More specifically, we are given a database of vectors on which we can perform preprocessing and build a data structure, and then, we receive a sequence of queries $\{q_i\}_i$, and for each query, we use the data structure to compute the element in the database that is most similar to q_i. Note that the structure of this problem depends on the similarity measure between vectors.

If n is the number of vectors, we can trivially create an $O(n)$ algorithm (per query): for every query q, iterate through the entire database and find the vector that is most similar to q. Remarkably, for Euclidean distance and cosine similarity, it is possible to achieve *amortized sublinear* query runtime (Indyk & Motwani, 1998; Charikar, 2002).

Because of applications in log-linear models, we will be interested in using the inner product as the similarity measure. This is known as the Maximum Inner Product Search (MIPS) task.

Definition 2.1 (Maximum Inner Product Search)**.** *Given a set of vectors $V = \{v_1, ..., v_n\}$ the MIPS task is to respond to a query vector q with*

$$\operatorname*{argmax}_{v \in V} q \cdot v \tag{7}$$

One common class of techniques for solving MIPS are space-partitioning methods such as k-d trees (Bentley, 1975). Ram & Gray (2012) and Koenigstein et al. (2012) introduce space-partitioning methods based on a branch and bound technique to solve the MIPS problem. Unfortunately, it has been observed that such tree-based methods suffer from the curse of dimensionality (Shrivastava & Li, 2014).

Clustering is another approach for solving the MIPS task (Auvolat et al., 2015; Douze et al., 2016). For this technique, the database vectors are clustered during the preprocessing step. Then, at query time, the algorithm searches the clusters near q for the most similar vector.

Another common class of techniques for MIPS are based on Local Sensitive Hashing (Shrivastava & Li, 2014; Neyshabur & Srebro, 2014), a method introduced by Indyk & Motwani (1998). LSH only requires a family of hash functions with collision probabilities that are monotonic in the similarity. LSH works by combining these basic hashes to form longer hashes, and then building a hash table for each longer hash. Then, at query time, LSH hashes the query, retrieves elements from the colliding hash buckets, and computes the maximum over such elements. More precisely, define $\mathrm{Sim}(x, y)$ as the similarity between x and y and an S-neighbor to a query q as a point x such that $\mathrm{Sim}(q, x) \geq S$.

Theorem 2.1. *Given a set V of size n with a similarity measure and hash family $\mathcal{H}$ such that for scalars $S_1 > S_2$ and $p_1 > p_2$,*

- *For any $x, y \in V$ where $\mathrm{Sim}(x, y) \geq S_1$, $\mathrm{Pr}_{h \in \mathcal{H}}[h(x) = h(y)] \geq p_1$*
- *For any $x, y \in V$ where $\mathrm{Sim}(x, y) \leq S_2$, $\mathrm{Pr}_{h \in \mathcal{H}}[h(x) = h(y)] \leq p_2$*

one can construct a data structure which, given any query q, does the following with high probability: if there exists a S_1-neighbor of q in V, it returns a S_2-neighbor of q in V. Further, this can be done with $O(n^\rho \log n)$ query time and $O(n^{1+\rho})$ space where $\rho = \frac{\log p_1}{\log p_2} < 1$.

Proof. See Indyk & Motwani (1998) □

This theorem states that if there is an S_1-close neighbor, the algorithm will find an S_2-close neighbor. Intuitively, this means that each LSH instance is "tuned" to a different similarity value. We can build a series of LSH instances "tuned" to different values so that we can find the largest element with high probability, no matter the similarity of the nearest neighbor. The theorem states that this can be done in sublinear time.

For the sublinear theoretical guarantees in this paper, we will rely on the reduction from MIPS to Maximum Cosines Similarity Search presented in Neyshabur & Srebro (2014) which adds a single dimension to make all the database vectors have the same norm. For the cosine similarity search problem, we will rely on LSH techniques for cosine similarity search presented in Charikar (2002) based on Signed Random Projections, a binary hash function based on the sign of the dot product with a random vector.

3 METHOD

Suppose we have a log-linear model over a set of n elements $\mathcal{X}$. We wish to perform sampling and inference in sublinear time. This cannot be done for a single value of the parameters θ, but with preprocessing on $\mathcal{X}$, we can achieve sublinear amortized query time.

Our method generally works for any distribution where

$$\Pr(i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{8}$$

which encompasses all distributions with strictly positive probability mass. The requirement for our method is that we have access to the largest $O(\sqrt{n})$ values of y_i in sublinear time. In particular, this method works for log-linear models where $y_i = \theta \cdot \phi(x_i)$ and we use Maximum Inner Product search techniques to access the top values.

3.1 SAMPLING

Recall the Gumbel max technique from the background section. In particular, if we can compute the maximum element and value of $y_i + G_i$ for Gumbel variables G_i, the maximum element will be a sample from the model. We can construct a naive strategy as follows: sample a Gumbel G_i for each y_i and iterate over all elements to find the maximum (perturbed) element. However, this algorithm's runtime is linear and provides no savings.

Ideally, we would like to find a way to preprocess $\{y_i\}_{i=1}^n$ so that we can draw samples and perform inference quickly. Mussmann & Ermon (2016) achieves this by performing preprocessing on fixed Gumbel samples and using MIPS techniques. This "frozen" Gumbel noise makes the samples very correlated; in fact, there are a small fixed number of possible samples for a given parameter value. We wish to find a way that allows us to sample fresh Gumbels for every sample, but only a sublinear number of them.

Intuitively, for an element to maximize $y_i + G_i$, either y_i needs to be large or G_i needs to be large, so we only need to examine indices where either y_i or G_i is large. We

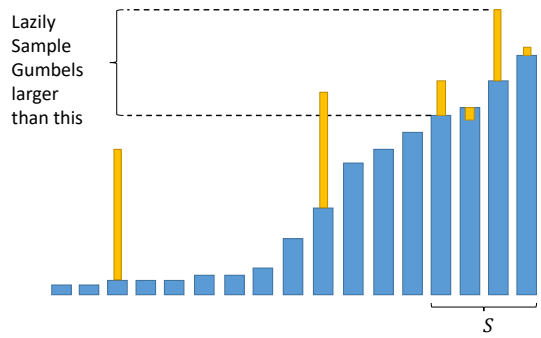

Figure 1: The values of y_i are shown sorted in blue while the Gumbel noise is shown in yellow. We sample a Gumbel for each element of the set S of the largest y_i. Then, we compute the minimum value that a Gumbel must have to yield a candidate solution, represented by the difference between the dotted lines. Finally, we lazily sample Gumbels larger than this value for elements not in S.

can find the largest y_i by performing preprocessing (such as maximum inner product search) and we will find the largest G_i by incorporating a lazy evaluation strategy for the Gumbel variables to only require an expected number of $O(\sqrt{n})$ samples.

First, we describe the method intuitively and with a figure before diving into the details. Let S be the set of the largest $O(\sqrt{n})$ elements of $\{y_i\}_{i=1}^n$. First, we will sample Gumbel values for these S largest elements. Note that the minimal y_i in S is an upper bound on the y_i value of elements not in S. Further, for an element not in S to have the overall maximal $y_i + G_i$, it must exceed the maximal $y_i + G_i$ for elements in S, which is quickly computable. Thus, we have a lower bound on what the value of a Gumbel must be to perturb a point not in S to be the overall maximum. We can lazily sample large Gumbels that exceed this gap, which we will show there will not be too many in expectation. Then, we randomly assign these large Gumbels to the tail of the distribution and check if any of them exceed the maximal $y_i + G_i$ from the largest elements S. See Figure 1.

Note that our method requires the top $k = O(\sqrt{n})$ elements of $\{y_i\}_{i=1}^n$ which we will refer to as S. For lazy sampling the large Gumbels, we will use the fact that a Gumbel can be represented as $G_i = -\ln(-\ln(U_i))$. Then we can sample the number of Gumbels that exceed a threshold B by sampling the number of U_i such that $U_i > \exp(-\exp(-B))$ and then can conditionally sample $U_i > \exp(-\exp(-B))$. Precisely, our method involves several steps shown in Algorithm 1.

Theorem 3.1. *For Algorithm 1, $\hat{x}$ is an exact sample from* $\Pr(i) \propto e^{y_i}$.

Algorithm 1 Fast Sampling with Lazy Gumbels

Input: $\{y_i\}_{i=1}^n$, S as the top k values of y_i
Sample k Gumbel variables G_i for $i \in S$
Compute $M = \max_{i \in S} y_i + G_i$
Compute $S_{\min} = \min_{i \in S} y_i$
Compute the Gumbel cutoff $B = M - S_{\min}$.
Sample $m \sim \text{Binomial}(n-k, 1-\exp(-\exp(-B))$ as the number of $|\mathcal{X} \setminus S|$ Gumbels with value $> B$
Uniformly sample m points from $\mathcal{X} \setminus S$ and denote T
Sample Gumbels that are conditionally $G_i > B$ for the points $i \in T$ (sample $U_i \sim \text{Uniform}(\exp(-\exp(-B)), 1)$)
$\hat{x} = \text{argmax}_{i \in S \cup T} y_i + G_i$
return Sample $\hat{x}$

Proof. This theorem would follow from Proposition 2.1 if we prove that we are finding the maximum of $y_i + G_i$. Note that we do not evaluate the Gumbel for all of the elements in $\mathcal{X} - S - T$. Thus, the only way the lazy sampling strategy will fail is if one of these points is the true maximum. However, these points have Gumbel $G_i < B$ and since they aren't in S, $y_i < S_{\min}$. Together, this implies that $y_i + G_i < S_{\min} + B = M$ which is a value attained by a point in S. Therefore points not in $S \cup T$ cannot be the maximum. □

3.1.1 Runtime

Further, the runtime will be composed of two parts: retrieving the top k elements S and the runtime of Algorithm 1. Let the cost of retrieving the top k elements be $f(n,k)$. For Algorithm 1, including the cost of retrieving S, the runtime will be $O(f(n,k) + m)$ and m has a reasonable expected value.

Theorem 3.2. *For Algorithm 1,* $\mathbb{E}[m] \leq \frac{n}{k}$

The proof is in the appendix. Thus, the expected runtime for our method will be $O(f(n,k) + \frac{n}{k})$ which is sublinear if $k = \sqrt{n}$ and $f(n, \sqrt{n})$ is sublinear.

3.1.2 Fixed B

Note that the technique above has a reasonable expected runtime but no runtime guarantees with high probability. To address this, we can fix B to be a constant so that the value of m is concentrated. Additionally, the technique shown in Algorithm 1 only works if $S_{\min}$ is an upper bound on elements not in S which is brittle to errors in the MIPS technique.

To address these issues, we define a related algorithm with a fixed Gumbel cutoff of $B = -\ln(-\ln(1 - l/n))$ so that there are on average l Gumbel variables that exceed the cutoff. See Algorithm 2.

Algorithm 2 Fast Sampling with Fixed B

Input: $\{y_i\}_{i=1}^n$, S as the top k values of y_i, l
Sample k Gumbel variables G_i for $i \in S$
Set $B = -\ln(-\ln(1 - l/n))$
Sample m as the number of $|\mathcal{X} - S|$ Gumbels with value $> B$
Uniformly sample m points from $\mathcal{X} - S$ and call them T
Sample Gumbels that are conditionally $G_i > B$ for the points $i \in T$
$\hat{x} = \text{argmax}_{i \in S \cup T}\, y_i + G_i$
return Sample $\hat{x}$

Algorithm 3 Partition Function Estimation

Input: $\{y_i\}$, S as the top k values of y_i, l
Uniformly sample l elements with replacement from $[1, n] \setminus S$ and call it T
$\hat{Z} = \sum_{i \in S} e^{y_i} + \frac{n-|S|}{|T|} \sum_{i \in T} e^{y_i}$
return Partition function estimate $\hat{Z}$

Note that for $|S| = k$, the total runtime is $O(f(n, k) + m)$ where $f(n, k)$ is the runtime of gathering the top k elements. Further $m \sim \text{Binomial}(n, l/n)$ so with very high probability, $m < 2l$ and the runtime is $O(f(n, k) + l)$ which will be sublinear if $f(n, k)$ and l are sublinear.

Theorem 3.3. *For Algorithm 2, the sample is an exact sample with probability* $1 - \delta$ *for* $kl \geq n \ln(1/\delta)$.

The proof is in the appendix. Thus, we can set $k = l \geq \sqrt{\ln(1/\delta)}\sqrt{n}$.

3.2 PARTITION FUNCTION ESTIMATION

Similar to sampling, we can estimate the partition function by using the top $k = O(\sqrt{n})$ elements S and a uniform sample T of $l = O(\sqrt{n})$ elements from the remaining elements. We combine these two sets to form an estimate of the partition function with relative error ϵ. See Algorithm 3.

Theorem 3.4. *Algorithm 3 returns an unbiased estimate* $\hat{Z}$ *and for* $kl \geq \frac{2}{3}\frac{1}{\epsilon^2} n \ln(1/\delta)$*, then with* $1-\delta$ *probability,*

$$\frac{|\hat{Z} - Z|}{Z} \leq \epsilon \tag{9}$$

The proof is in the appendix. If we set $k = l$, then the runtime is $O(\frac{1}{\epsilon}\sqrt{n}\sqrt{\ln(1/\delta)})$.

This is closely related to the heuristic presented in Rastogi & Van Durme (2015) as MIMPS. However, this is the first work that provides theoretical guarantees for the method and yields a theoretical understanding for the choice of k and l.

Algorithm 4 Expectation Estimation

Input: $\{y_i\}$, bounded function values f_i, S as the top k values of y_i, l
Uniformly sample l elements with replacement from $[1, n] \setminus S$ and call it T
$\hat{Z} = \sum_{i \in S} e^{y_i} + \frac{n-|S|}{|T|} \sum_{i \in T} e^{y_i}$
$\hat{J} = \sum_{i \in S} e^{y_i} f_i + \frac{n-|S|}{|T|} \sum_{i \in T} e^{y_i} f_i$
$\hat{F} = \hat{J}/\hat{Z}$
return Expectation estimate $\hat{F}$

3.3 EXPECTED VALUE ESTIMATION

In this section we show a way to estimate an expected value with respect to the distribution $\Pr(i) \propto e^{y_i}$. In particular, for bounded function values $\{f_i\}_{i=1}^n$ where $|f_i| \leq C$ we can define the expectation

$$F = \sum_i \frac{e^{y_i}}{Z} f_i \tag{10}$$

where $Z = \sum_i e^{y_i}$. The algorithm we use to create an estimate is very similar to the partition function estimate. More specifically, we compute the largest S values of $\{y_i\}_{i=1}^n$ and then draw uniform samples from the remaining elements and call it T. Then we compute an expected value using S and T (and upweighting the estimate from T). See Algorithm 4.

This algorithm comes with a guarantee on the additive error.

Theorem 3.5. *Algorithm 4 returns an estimate* $\hat{F}$ *such that* $|\hat{F} - F| \leq \epsilon C$ *with probability* δ *if*

$$k^2 l \geq \frac{8n^2}{\epsilon^2} \log(4/\delta) \tag{11}$$

$$kl \geq \frac{8}{3}\frac{1}{\epsilon^2} n \ln(2/\delta) \tag{12}$$

The proof is in the appendix. If we set $k = l$ then

$$k = O(n^{2/3}(1/\epsilon)\sqrt{\log(1/\delta)}) \tag{13}$$

Then, with a sublinear MIPS technique, the total runtime is sublinear. Note that we can use this to compute the expectation of $\phi(x)$ and thus the gradient of data likelihood. This technique will be used in the experiments section for the learning experiment.

3.4 APPROXIMATE TOP ELEMENTS

Many Maximum Inner Product Search (MIPS) methods, including LSH-based techniques, do not solve the exact nearest neighbor problem, but approximate nearest

neighbor problem. In this work, we define a similar concept of the approximate top $O(\sqrt{n})$ elements that will suffice for our theoretical arguments. Further, we show that we can use LSH instances to retrieve the approximate top k elements in sublinear time.

We say that an algorithm returns the approximate top k if the gap between the smallest element in S and the largest element *not* in S is bounded by a constant.

Definition 3.1 (Approximate Top k)**.** *A set of elements S is an approximate top k if $|S| = k$ and*

$$\max_{i \notin S} y_i - \min_{i \in S} y_i < c \tag{14}$$

We can create a sequence of LSH instances that are "tuned" to a range of similarity values. Then at query time, we can go through the LSH instances in decreasing order of tuned value, gathering elements until we have k elements. It turns out that these elements will be the approximate top k elements (more details in the appendix). This technique will have a total runtime of

$$O(k + (\log(k) + \log(1/\delta)) \log(n) n^{\rho}) \tag{15}$$

where $\rho < 1$. Thus, we have a sublinear approximate top k element MIPS technique. We state this as a theorem and prove it in the appendix.

Theorem 3.6. *For sublinear k, there exists a MIPS technique that returns the approximate top k elements in sublinear amortized time.*

Note that if we have a MIPS technique that returns an approximate top k set S then we can adapt Algorithm 1 to make $B = M - S_{\min} - c$ for an added increase of e^c in the expected value of m, and thus the runtime.

If we have a MIPS technique that returns an approximate top k set S with constant c, then Algorithm 2 and 3 will have an extra factor of $e^{c/2}$ for k and l and Algorithm 4 will have an extra factor of $e^{2c/3}$ for k and l. These extensions are proved in the appendix and the previously stated theorems are special cases with $c = 0$.

4 EXPERIMENTS

In this section, we present an empirical evaluation of our proposed sampling, inference, and expectation techniques. The use case for our method is when there are fixed feature vectors $\{\phi(x)\}_{x \in \mathcal{X}}$, and a sequence of inference or sampling queries with different parameter vectors $\{\theta_i\}$. Although we cannot achieve gains on a single query, through preprocessing we can decrease the amortized query time. We will evaluate **runtime improvements** and **accuracy**.

4.1 PRELIMINARIES

4.1.1 MIPS technique

We present the MIPS technique used to retrieve the top-k values of the unnormalized log-probabilities. We follow the approximate nearest neighbor search method presented in Douze et al. (2016) as well as the publicly available implementation. However, we will not be making use of the compression component, as we do not optimize for memory usage.

This method relies on the use of a k-means clustering. With the same notations as 2.3, given a query q and a set of vectors V, we aim at finding the k highest values of $\{q \cdot v, v \in V\}$.

We first cluster the vectors in V in n_c clusters. For an incoming query vector q, we look at the inner product with the vectors in the cluster q is assigned to as well as n_p neighboring clusters. While this method doesn't have any theoretical guarantees, it has been shown to perform better than LSH in practice as it more advantageously exploits the distribution of the set of vectors.

In our experiments, we use CPU implementations of all algorithms for fair comparison.

4.1.2 Data

We experiments with two datasets from different domains to demonstrate the effectiveness of our method in real-world use.

Word Embeddings We use a set of word embeddings released by Facebook (Bojanowski et al., 2016). Each embedding is a dense vector representing a word in a given vocabulary. These continuous representations are obtained by training log-bilinear models on large text corpora. The embeddings incorporate structure from character n-grams of the words. We retain words containing only letters and scale each vector to be of unit-norm. The data is composed of $N = 2,000,126$ vectors of dimension $d = 300$.

ImageNet The ImageNet dataset (Russakovsky et al., 2015) from the ILSVRC 2012 competition contains 1.2 million natural images divided into 1000 classes. We extract features using a pre-trained residual network (He et al., 2016) trained on this classification task. More precisely, we represent each image by its activation map from the last layer before the linear classification layer of a ResNet-152. The extracted features are of size $7 \times 7 \times 2048$ for each image. We then take the average along the depth dimension and reduce dimensionality using a PCA. We scale each vector to be of unit-norm.

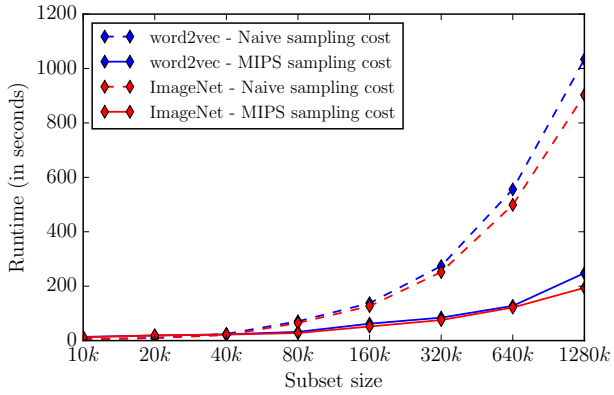

Figure 2: Empirical comparison of the runtime of sampling for $10,000$ randomly chosen θ from a log-linear model on subsets (of varying size) of the datasets. Note the log-scale of the dataset size. This time is the per query runtime and does not include preprocessing.

The data is thus composed of $N = 1,281,167$ vectors of dimension $d = 256$. In the rest of our experiments, we choose the temperature of the log-linear model to be $\tau = 0.05$.

4.2 SAMPLING

In this section, we measure the performance of our method in terms of both sampling quality and speed. We first present empirical results on sampling and then illustrate the efficiency of our method on a specific task: a random walk over ImageNet.

4.2.1 Sampling

Speed We want to evaluate the runtime of our method for sampling on large datasets. Given a dataset $\mathcal{X}$ and a parameter vector θ, we compare the time necessary to sample from $\Pr(x) \propto e^{\theta \cdot \phi(x)}$ using our method or by enumeration (brute force). We compute the sampling time for random vectors $\{\theta_i\}_{i \leq 1000}$ and subsets of varying size for ImageNet ranging from $10,000$ to $1,280,000$. The results are presented in Figure 2.

We can see that the speedup is linear w.r.t the log of the sub-sampled dataset size, achieving up to $5\times$ sampling speedup for the full dataset of size $1,281,167$. If we consider the amortized cost, i.e. including the pre-processing cost of our MIPS data structure, our method starts paying off after approximately $8,600$ samples. The amortized costs are presented in the appendix, in Figure 7.

Accuracy To measure the accuracy of our method, we present a way to establish an upper bound on the total variation distance in closed form for a given θ. Then, we average this upper bound over 100 samples of θ (drawn uniformly from the dataset).

Dataset	Speedup	Total Variation Bound
ImageNet	$4.65\times$	$(2.5 \pm 1.4) \times 10^{-4}$
Word Embeddings	$4.17\times$	$(4.8 \pm 2.2) \times 10^{-4}$

Table 1: Summary of the sampling speedup and bound on the total variation distance for our method on the ImageNet and Word Embeddings datasets.

Note that the lazy sampling strategy is exact unless the true maximum is not in $S \cup T$. Thus, if we can upper bound this probability, it is an upper bound on the total variation distance. For a given threshold x, we can compute the closed form probability that $\max_{i \notin S \cup T} y_i + G_i < x$ and $\max_{i \in S} y_i + G_i > x$. This is the upper bound that we desire and we can optimize x for the tightest upper bound. For both datasets, over 100 samples of θ, the average upper bound was on the order of 10^{-4} proving that our sampling method is accurate even while using an approximate MIPS technique. A summary of our results in terms of accuracy and speedup are provided in Table 1. We provide further empirical evidence in the appendix to show that the distributions closely match on the shown θ.

4.2.2 Random walk over a large set

To showcase the applicability of our method, we perform a random walk over the ImageNet dataset. We define the transition function, i.e. the probability to walk from image j to image i as $\Pr(X_{t+1} = i | X_t = j) \propto e^{\tau \phi(x_i) \cdot \phi(x_j)}$ where τ is the temperature, ϕ is the fixed featurization previously defined, and x_i, x_j are the pixel-values of images i, j. The initial state is sampled uniformly across the dataset. This is similar in spirit to the PageRank algorithm (Page et al., 1999). This setting fits our method because while the MIPS structure can be reused across time steps, no computation can be cached in the naive setting (assuming we do not store the distribution for each element, which would be on the order of Terabytes).

We evaluate the quality of the Markov Chain by comparing the top elements of the empirical sampling distribution. We run two different Markov chains, one with exact sampling and one with our sampling technique. Over one million steps, the two Markov Chains share 73.6% of the top 1000 elements. This percentage looks low because of the finite sampling error. When we compare two different one million element windows within each chain, the top 1000 elements are shared 69.3% and 72.9% for the

Figure 3: Samples of the Markov chain. The samples are spaced out by 20 time steps.

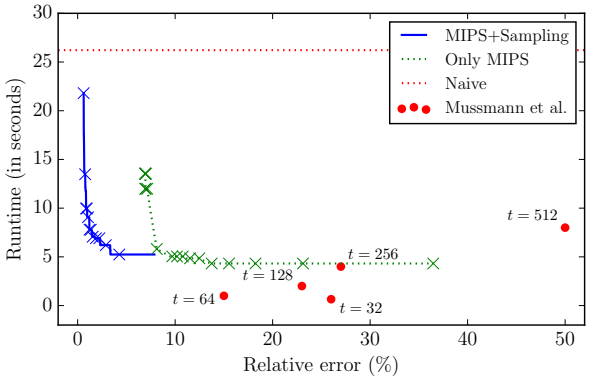

Figure 4: Runtime plotted as a function of relative error of partition function estimate (different points made by varying k and l) on ImageNet (averaged over random values of θ. The red dotted line is the time for the exact partition function computation.

exact sampling and our sampling, respectively. It is seen that the between-chain differences are the same as the within-chain differences, so the Markov chain with our sampling technique yields roughly the same distribution as the chain with exact sampling.

4.3 PARTITION FUNCTION ESTIMATE

We show the performance of our partition function estimate as shown in Algorithm 3. We can trade-off error and runtime by varying k and l. See Figure 4. We average the results over several values of θ, drawn uniformly from the dataset. For comparison, we plot the trade-off for only looking at the top k values and using this as a partition function estimate. Additionally, we compare to the method of Mussmann & Ermon (2016) for different size of noise t. For each value of k, l and t we report the runtime and relative error of the partition function estimate. As shown by the relative error of the top-k estimate, sampling from the tail is necessary to achieve low relative error. We also show that the method from Mussmann & Ermon (2016) cannot come close in terms of relative error, achieving a maximum of 15% relative error for $t = 64$. It is also important to note that their method cannot trade-off speed for accuracy as, when the noise-length t increases, the injected noise destroys the MIPS structure rendering it highly inaccurate.

Method	Log-likelihood	Speedup
Exact gradient	-3.170	$1\times$
Only top-k	-4.062	$22.7\times$
Our method	-3.175	$9.6\times$

Table 2: Log-likelihood and speedup for the learning of a log-linear model on ImageNet. For our method, we picked $k = 10\sqrt{n}, l = 10k$, for the comparison to only weighing the top-k, we chose $k = 100\sqrt{n}$ as well.

4.4 LEARNING

We wish to maximize the likelihood of a subset of the data $\mathcal{D} \subseteq \mathcal{X}$ given $\Pr(\cdot; \theta)$. We aim at finding

$$\theta^* = \arg\max_{\theta} \sum_{x \in \mathcal{D}} \log \Pr(x; \theta) \tag{16}$$

using gradient ascent. Evaluating the gradient requires finding the expectation of the features $\phi(x)$ which can be estimated using our method in Algorithm 4. The features are fixed but θ is updated at each step of the gradient ascent algorithm, fitting well into the setting of our method. We choose a small subset $\mathcal{D}$ of ImageNet as images with a commonality. In particular, we handpick 16 images showing the presence of water. We compare computing the gradient with our method to the computation of the exact gradient and to approximating the gradient by considering the truncated distribution on the top k elements (referred to as top-k gradient). The chosen images are shown in the appendix in Figure 9. We perform gradient ascent for 5000 iterations with learning rate $\alpha = 10$, which we halve every 1000 iterations. The results are reported in Table 2. The learning curves are shown in Figure 5. We also show the 10 most probable samples (outside of the dataset $\mathcal{D}$) according to the log-linear model in Figure 6. We can see that these images are semantically similar to the training set, all containing water, showcasing the expressive power of the ResNet features.

As shown in Figure 5, we can see that the log-likelihood for our method and the exact gradient almost exactly overlap indicating that our estimation of the gradient is very accurate. In contrast, the top-k gradient, while faster, proves to be a poor estimator and thus cannot optimize the log-likelihood. To summarize, **our method converges to the global maximum $9.6\times$ as fast as computing the exact gradient**.

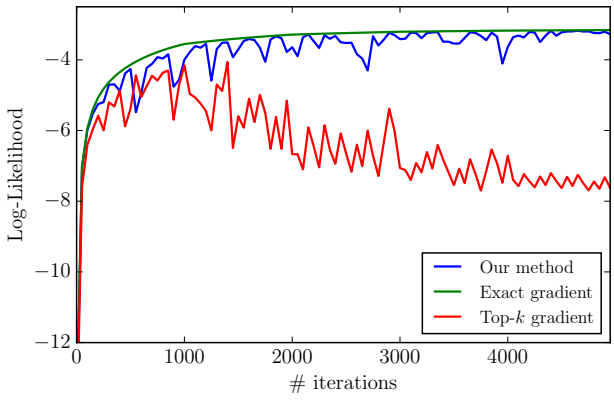

Figure 5: Log-likelihood plotted against the number of iterations for performing gradient ascent on our learning problem for 5000 iterations with a learning rate $\alpha = 10$, halving the learning rate every 1000 iterations.

Figure 6: 10 most probable images (outside of $\mathcal{D}$) from our log-linear model trained to convergence.

5 RELATED WORK

Our method can be viewed in two different comparative perspectives. Our method can be seen as an alternative to only using the top-k most probable elements as is done in Vijayanarasimhan et al. (2014). There, large output spaces for deep learning are handled using Locality Sensitive Hashing. In particular, the top vectors are gathered and the rest of the vectors in the tail are ignored. For spread-out distributions (closer to uniform), this method will fail. Our work provides a scalable method to incorporate the probability mass present in the tail of the distribution by sampling $O(\sqrt{n})$ elements, a small prices compared to retrieving the top elements.

Our method can also be compared to a different way of combining the Gumbel max trick and Maximum Inner Product Search as presented in Mussmann & Ermon (2016). In that work, Gumbel noise is appended to the database vectors and stored in the MIPS data structure. Then, query vectors are chosen to access the frozen Gumbel noise. That work has several major shortcomings that make it unusable in practice.

The Gumbel noise is re-used, introducing correlated samples and systematic bias in the partition function estimate. In particular, for any fixed value of the parameters, there are a fixed number of samples "frozen" into the stored Gumbel noise. We avoid this issue by sampling $O(\sqrt{n})$ fresh Gumbel variables for every sample. While real world data often has structure that can be exploited by the MIPS techniques, in Mussmann & Ermon (2016), the structure is destroyed by injecting random Gumbel noise. In our technique, we preserve structure in the database vectors by leaving the vectors unchanged. Finally, the method of Mussmann & Ermon (2016) requires accessing the MIPS data structure many times for independent samples and partition function estimates. In this work, we only require accessing the MIPS data structure once per parameter value.

6 CONCLUSION

In conclusion, we have presented several related methods that are based on the key idea of accessing the large elements in a distribution using Maximum Inner Product Search and accessing the tail of a distribution with uniform sampling. This decreases the runtime from $O(n)$ to $O(\sqrt{n})$ plus the runtime for the MIPS technique.

This work is best suited for cases where the output space is large but enumerable, such as those in NLP and computer vision. This work can be expected to give speedups when the feature vectors of a log-linear model are fixed but it is desired to perform inference and sampling for several different values of the the parameters. Note that our method is as flexible as the MIPS method that is employed; the feature vectors need to only be fixed for the MIPS to work. As an example, if a MIPS system allows for sparse updates, our method will also allow for sparse updates. Since our method treats MIPS as a black-box, advances in the speed and accuracy of MIPS techniques automatically improve our method.

When accessing the top elements is not accurate enough, we present a method to include uniform samples from the tail to provide provably good samples and estimates of the partition function. All this, at the small overhead price of uniform sampling.

7 Acknowledgments

This research was supported by Intel Corporation, Future of Life Institute ($\#2016 - 158687$) and NSF grants 1651565, 1649208, 1522054, and DGE-1656518.

We thank Ludwig Schmidt and Moses Charikar for helpful discussions.

References

Auvolat, Alex, Chandar, Sarath, Vincent, Pascal, Larochelle, Hugo, and Bengio, Yoshua. Clustering is efficient for approximate maximum inner product search. *arXiv preprint arXiv:1507.05910*, 2015.

Bengio, Yoshua, Ducharme, Réjean, Vincent, Pascal, and Jauvin, Christian. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb): 1137–1155, 2003.

Bentley, Jon Louis. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

Bojanowski, Piotr, Grave, Edouard, Joulin, Armand, and Mikolov, Tomas. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

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

Douze, Matthijs, Jégou, Hervé, and Perronnin, Florent. Polysemous codes. In *European Conference on Computer Vision*, pp. 785–801. Springer International Publishing, 2016.

Gumbel, Emil Julius and Lieblein, Julius. Statistical theory of extreme values and some practical applications: a series of lectures. 1954.

Hazan, Tamir, Maji, Subhransu, and Jaakkola, Tommi. On sampling from the gibbs distribution with random maximum a-posteriori perturbations. In *Advances in Neural Information Processing Systems*, pp. 1268–1276, 2013.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, 2016.

Indyk, Piotr and Motwani, Rajeev. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613. ACM, 1998.

Joulin, Armand, van der Maaten, Laurens, Jabri, Allan, and Vasilache, Nicolas. Learning visual features from large weakly supervised data. In *European Conference on Computer Vision*, pp. 67–84. Springer, 2016.

Kim, Carolyn, Sabharwal, Ashish, and Ermon, Stefano. Exact sampling with integer linear programs and random perturbations. In *Proc. 30th AAAI Conference on Artificial Intelligence*, 2016.

Koenigstein, Noam, Ram, Parikshit, and Shavitt, Yuval. Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 535–544. ACM, 2012.

Koller, Daphne and Friedman, Nir. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Maddison, Chris J, Tarlow, Daniel, and Minka, Tom. A* sampling. In *Advances in Neural Information Processing Systems*, pp. 3086–3094, 2014.

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

Murphy, Kevin P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Mussmann, Stephen and Ermon, Stefano. Learning and inference via maximum inner product search. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 2587–2596, 2016.

Neyshabur, Behnam and Srebro, Nathan. On symmetric and asymmetric lshs for inner product search. *arXiv preprint arXiv:1410.5518*, 2014.

Page, Lawrence, Brin, Sergey, Motwani, Rajeev, and Winograd, Terry. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Ram, Parikshit and Gray, Alexander G. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 931–939. ACM, 2012.

Rastogi, Pushpendre and Van Durme, Benjamin. Sublinear partition estimation. *arXiv preprint arXiv:1508.01596*, 2015.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Shrivastava, Anshumali and Li, Ping. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pp. 2321–2329, 2014.

Vijayanarasimhan, Sudheendra, Shlens, Jonathon, Monga, Rajat, and Yagnik, Jay. Deep networks with large output spaces. *arXiv preprint arXiv:1412.7479*, 2014.