Learning and Inference via Maximum Inner Product Search

Stephen Mussmann MUSSMANN@STANFORD.EDU
Stanford University, 450 Serra Mall, Stanford, CA 94305 USA

Stefano Ermon ERMON@CS.STANFORD.EDU
Stanford University, 450 Serra Mall, Stanford, CA 94305 USA

Abstract

A large class of commonly used probabilistic models known as log-linear models are defined up to a normalization constant.Typical learning algorithms for such models require solving a sequence of probabilistic inference queries. These inferences are typically intractable, and are a major bottleneck for learning models with large output spaces. In this paper, we provide a new approach for amortizing the cost of a sequence of related inference queries, such as the ones arising during learning. Our technique relies on a surprising connection with algorithms developed in the past two decades for similarity search in large data bases. Our approach achieves improved running times with provable approximation guarantees. We show that it performs well both on synthetic data and neural language models with large output spaces.

1. Introduction

A large class of probabilistic models used in machine learning, physics and statistics are defined up to a normalization constant. This constant, often known as the partition function, ensures that the model is a valid probability distribution. Models involving a normalization constant include undirected graphical models, such as Markov Random Fields, Conditional Random Fields (Lafferty et al., 2001), and extremely common special cases such as multinomial logistic regression (softmax) (Koller & Friedman, 2009; Murphy, 2012).

Defining probabilities up to a normalization constant gives some extra modeling flexibility, however, it comes at a price. Nearly all inference tasks, including evaluating the probability of a state according to the model, require evaluating the partition function. Unfortunately, the problem is generally intractable in the worst case, with complexity scaling linearly in the number of states (Roth, 1996). Problems with large output spaces arise naturally in many fields including NLP and computer vision (Bengio et al., 2006; Andreas et al., 2015; Devlin et al., 2014), and pose numerous challenges to inference and learning algorithms (Ermon et al., 2013a). Learning the parameters from data, in particular, typically requires solving a sequence of probabilistic inference queries. This is often a major bottleneck for learning models with large output spaces.

While sequences of inference queries (arising at learning time, but also at inference time, after the model has been trained) are often treated independently and solved separately, they share some common structure. This observation has been exploited before, e.g., in the so-called persistent chains used in training RBMs (Tieleman, 2008), where a Markov Chain is initialized using samples from a related model (or training data) to speed up convergence. In this work, we take an alternative view and propose an new approach to *amortize* the cost of a sequence of related inference queries (Gershman & Goodman, 2014). Specifically, the idea is to pay upfront an overhead cost to construct a special data structure. This data structure will later allow us to answer inference queries in a much more efficient way, thus amortizing the initial cost and overall reducing the total computational cost.

Our technique relies on a surprising connection with algorithms for similarity search in large data bases. These algorithms have been critical to scale up information retrieval systems in the past two decades, e.g., to efficiently detect duplicate documents at internet scale (Henzinger, 2006; Manku et al., 2007). In this work, we provide a novel algorithm to sample from general log-linear models based on solving a *randomly perturbed* Maximum Inner Product Search (MIPS) instance (Shrivastava & Li, 2014). We show that our approach produces samples with the correct marginals when an exact MIPS method is used, and

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

we are able to bound the error induced by an approximate MIPS solution technique. Crucially, a large literature exists on how to exactly or approximately solve sequences of MIPS instances very efficiently (in an amortized sense). One class of approaches are space-partitioning methods (Ram & Gray, 2012; Koenigstein et al., 2012), similar to k-d trees (Friedman & Tukey, 1974). One such advance is a way to reduce MIPS to Maximum Cosine Similarity Search (MCSS) (Neyshabur & Srebro, 2015). MCSS can be solved using Locality Sensitive Hashing (LSH) techniques such as the Signed Random Projections hash (Charikar, 2002) or a Hamming distance LSH on vectors transformed by Signed Random Projections (Ravichandran et al., 2005). There is also a recent clustering approach to solve MCSS (Auvolat et al., 2015) which uses spherical k-means (Zhong, 2005). Our framework allows us to leverage all these results to efficiently solve a sequence of related sampling problems, where for example the parameters of the model are allowed to change over time (a common situation in a learning loop). Our approach can leverage any existing MIPS solution technique as a block-box, and inherits improved running times with provable approximation guarantees. We show that it performs well both on synthetic data and neural language models with large output spaces.

2. Background

2.1. Log-linear Models

There is a large class of probabilistic models in machine learning where the probabilities are defined up to a normalization constant known as the partition function. This paper focuses on inference and sampling in log-linear models.

Log-linear models are very common in machine learning and statistics, and include as special cases multinomial logistic regression, Conditional and Markov Random fields (Lafferty et al., 2001; Koller & Friedman, 2009; Murphy, 2012). Suppose we wish to assign probabilities to a set of states $\mathcal{X}$, for instance corresponding to all possible assignments to a set of variables.

The defining characteristic of log-linear models is that the log unnormalized probability is a linear combination of a set of *features* or sufficient statistics $\phi : \mathcal{X} \mapsto \mathbb{R}^d$. Log-linear models define the following probability distribution:

$$P_\theta(x) = e^{\theta \cdot \phi(x)} Z_\theta^{-1} \tag{1}$$

where θ is a vector of weights. Z_θ is a normalization constant known as partition function:

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

The partition function ensures that $P_\theta(x)$ is a valid probability distribution that sums to one.

2.1.1. Inference and Learning in Log-linear Models

The method presented in this paper provides estimates of the partition function and samples from the model for different parameter values. This section highlights describes why these tasks are important.

Evaluating the probability of a state (or some data) in a log-linear models with parameters θ requires computing the partition function (see equation 2). In general, computing the partition function requires summing over all possible states and is known to be computationally intractable in the worst-case (Koller & Friedman, 2009). In practice, it can be expensive even if the state space can be enumerated but is moderately large (millions or billions of possible states).

Another important task is learning. Suppose we have some data points $D = (x_1, \ldots, x_K)$, $x_i \in \mathcal{X}$, drawn i.i.d. from some underlying distribution and we wish to find a θ to fit a generative log-linear model (1) to this data. One common estimator for θ is the maximum likelihood estimator which can be written as

$$\hat{\theta} = \operatorname*{argmax}_\theta \Pi_{x \in D} P_\theta(x) \tag{3}$$

and found by minimizing the negative log likelihood,

$$L(\theta) = -\frac{1}{|D|} \sum_{x \in D} \theta \cdot \phi(x) + \log(Z_\theta) \tag{4}$$

The negative log likelihood is convex so convex optimization techniques can be used, such as gradient descent.

$$\nabla L(\theta) = -\frac{1}{|D|} \sum_{x \in D} \phi(x) + \mathbb{E}_\theta(\phi(x)) \tag{5}$$

The gradient set to 0 has an intuitive interpretation. The maximum likelihood estimator for log-linear models matches the means of the empirical data distribution and the log-linear model.

Note that the only dependence of the gradient on θ is through the expectation. Further, this expectation can be approximated using a Monte Carlo estimate, if we have access to samples from (1).

2.1.2. Model Averaging

Model averaging is a Bayesian technique to incorporate uncertainty in the parameters. This provides an application of inference in log-linear models for various values of θ. In the model averaging framework, the parameters are a distribution with probability density $P(\theta)$ rather than a point estimate (e.g., a posterior distribution over the parameters given some data). To perform inference on new data points, we integrate over the parameters:

$$P(x) = \int_\theta P_\theta(x) P(\theta) \tag{6}$$

Since this is an expected value $\mathbb{E}_\theta(P_\theta(x))$, we can use Monte Carlo sampling to give an approximation. In particular, we draw samples S from $P(\theta)$ and use the estimate

$$P_\theta(x) \approx \frac{1}{|S|} \sum_{\theta \in S} e^{\theta \cdot \phi(x)} Z_\theta^{-1} \tag{7}$$

2.2. Gumbel Variables

Our solution for the reduction of inference and sampling in log-linear models to MIPS critically relies on properties of Gumbel random variables. A Gumbel random variable G (Gumbel & Lieblein, 1954) is a continuous random variable with the cumulative distribution function (CDF),

$$P(G < x) = \exp(-\exp(-\frac{x-\mu}{\beta})) \tag{8}$$

where μ is the location parameter and β is the scale parameter. In this work, we will use $\mu = 0$ and $\beta = 1$. We can produce samples from a Gumbel distribution by sampling U uniformly from $[0, 1]$ and computing,

$$G = -\log(-\log(U)) \tag{9}$$

2.2.1. Properties of Maximum of Gumbels

Gumbel variables have recently been used for a number of probabilistic inference algorithms (Papandreou & Yuille, 2011; Hazan et al., 2013; Maddison et al., 2014; Kim et al., 2016). These methods all rely on the max-stability of Gumbel random variables and Gumbel processes.

Lemma 2.1. *Let $\{a_i\}_i$ be a set of coefficients, and $\{G_i\}_i$ and G be independent Gumbel random variables, then*

$$\max_i \{a_i + G_i\} \sim \log(\sum_i e^{a_i}) + G \tag{10}$$

$$\operatorname*{argmax}_i \{a_i + G_i\} \sim Multinomial(\{\frac{e^{a_i}}{\sum_i e^{a_i}}\}) \tag{11}$$

Proof. See (Gumbel & Lieblein, 1954; Papandreou & Yuille, 2011; Hazan et al., 2013; Maddison et al., 2014) □

If $a_i = \theta \cdot x_i$ are the log probabilities in a log-linear model, Lemma 2.1 can be used for inference. The solution to the optimization problem in (10) is a Gumbel random variable with the location shifted by the value of the log partition function. Therefore, the value of the log partition function can be estimated with a Monte Carlo estimate. Note that from (11), the argmax will be a sample from the model. More precisely, if we index $\mathcal{X}$ as $\{x_i\}_{i=1}^N$ and sample independent Gumbel variables $\{G_i\}_{i=1}^N$,

$$s = \operatorname*{argmax}_i \theta \cdot \phi(x_i) + G_i \tag{12}$$

is a sample from the model (Papandreou & Yuille, 2011), i.e., $s \sim P_\theta(x)$. For these reasons, Gumbel variables can be used to reduce inference and sampling problems into (randomly perturbed) optimization problems. This idea is closely related to other recent inference techniques based on random projections (Chakraborty et al., 2013; Ermon et al., 2014; Zhu & Ermon, 2015; Hadjis & Ermon, 2014; Achlioptas & Jiang, 2015; Hsu et al., 2016), and will be one of the key ingredients for our new inference approach.

2.3. Maximum Inner Product and Cosine Similarity Search

Maximum Inner Product Search (MIPS) is a standard computational task where we are given a set of vectors V and a query vector q, and the goal is to find the vector in V that has the maximum inner product with the query vector q (Shrivastava & Li, 2014; Ram & Gray, 2012; Shen et al., 2015). In this work, we reduce sampling and inference in log-linear models to the MIPS task. More precisely,

Definition 2.1. *Given a set of vectors $V = \{v_1, \cdots, v_N\}$ and a query vector q, the MIPS task is to compute*

$$\operatorname*{argmax}_{v \in V} q \cdot v \tag{13}$$

A geometric interpretation of this problem is to find the vector v that has the maximum projection onto a query vector q. This problem has numerous applications, including webpage duplication detection and image querying, where V is a set of vectorized web pages or images and q is a search query or webpage under question (Manku et al., 2007; Henzinger, 2006; Kulis & Grauman, 2009).

In almost all cases, one is interested in solving a large number of queries for the *same* set of vectors V. The typical approach is to perform a preprocessing step on V in order to quickly respond to a series of multiple queries. Thus, *the objective is to minimize the average query time since the preprocessing time will be amortized for a large enough number of queries*.

Maximum Cosine Similarity Search (MCSS) is very similar to MIPS except that the objective is slightly different.

Definition 2.2. *Given a set of vectors $V = \{v_1, \cdots, v_N\}$ and a query vector q, the MCSS task is to compute*

$$\operatorname*{argmax}_{v \in V} \frac{q \cdot v}{||q||||v||} \tag{14}$$

where $||\cdot||$ denotes the Euclidean norm.

Note that MCSS is a special case of MIPS if all of the vectors in V are of unit length.

2.3.1. Conversion of MIPS to MCSS

Several recent methods for MIPS rely on a reduction from MIPS to MCSS. The methods in (Auvolat et al., 2015) and (Shrivastava & Li, 2015) rely on a particular approximate reduction of MIPS to MCSS proposed in (Shrivastava & Li, 2015). However, it appears that the reduction presented in (Bachrach et al., 2014) and used in (Neyshabur & Srebro, 2015) is simpler, faster, and more accurate.

This reduction is summarized here. Because scaling doesn't affect the argmax, without loss of generality, let q be a unit vector and the vectors V be unit length or smaller. Denote vector concatenation as $(\cdot, \cdot)$. Define the transformation: $r(v) = (v, \sqrt{1 - ||v||^2})$. Let $q' = r(q) = (q, 0)$ and $V' = r(V)$. q' and V' are all vectors of unit length since $||r(v)|| = 1$. Finally, the dot product is unchanged: $r(q) \cdot r(v) = q \cdot v$. Thus, a solution to the converted MCSS problem will be a solution to the original MIPS problem.

2.3.2. Methods for MIPS and MCSS

A simple method to solve the MIPS problem is to iterate through V to find the vector with maximum dot product with q. The runtime of this method is simply $O(N)$, where $N = |V|$. Note that this naive method is optimal for a single query q since we must examine all vectors in V. However, improvements can be made for the amortized query time of multiple queries if we perform preprocessing on V.

A number of techniques attempt to optimize the amortized complexity. These include branch-and-bound techniques (Ram & Gray, 2012; Koenigstein et al., 2012), which are similar to k-d trees (Friedman & Tukey, 1974). These were recently generalized to max kernel search (Curtin et al., 2013). However, as (Shrivastava & Li, 2014) notes, these methods suffer from the curse of dimensionality.

There are also several *approximate* methods that seek to return a vector such that $q \cdot v$ is close to the maximum, but not necessarily the maximum. Spherical k-means (Zhong, 2005) clusters V in a hierarchical structure as a preprocessing step. To answer queries, the algorithm recurses down the cluster hierarchy tree to find the element closest to q (Auvolat et al., 2015). Note that there are parameters in this model that trade off the runtime and the accuracy.

Another group of methods are based on Locality Sensitive Hashing (Indyk & Motwani, 1998). These methods require a family of hash functions where the collision probability is monotonic in the similarity. The core idea is to create many hash tables with a concatenation of the hash function to amplify the collision probability for similar elements but not for dissimilar ones. One such method for MCSS uses Signed Random Projections (Charikar, 2002) as the hash function. This hash function can be thought of as sampling a random plane that passes through the origin and assigning 0 to one side and 1 to the other. Another method (Ravichandran et al., 2005) transforms the input vectors to binary vectors using a concatenation of Signed Random Projections, then uses a hamming distance hash on the resulting transformed vectors. This method benefits from an adaptive technique for a hamming distance hash presented at the end of Charikar (2002). LSH methods have probabilistic theoretical guarantees for the results of the queries.

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

- *If $Sim(x, y) \geq S_1$, then $P_{h \in \mathcal{H}}[h(x) = h(y)] \geq p_1$*
- *If $Sim(x, y) \leq S_2$, then $P_{h \in \mathcal{H}}[h(x) = h(y)] \leq p_2$*

one can construct a data structure which, given any query q, does the following with probability $1 - \delta$: 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 $\rho = \frac{\log(p_1)}{\log(p_2)}$.

Proof. See (Indyk & Motwani, 1998) and (Shrivastava & Li, 2014) □

Intuitively, this theorem states that if we have a hash function with collision probabilities dependent on similarity, we can design a data structure that responds to queries in sublinear time. Note that we can vary the difference between S_1 and S_2 to trade off accuracy and runtime. Some possible examples are $S_2 = cS_1$ or $S_2 = S_1 - c$. We will refer to the later as additive error.

3. Sampling and Inference in Log-linear Models via MIPS

Our main contribution is a new approach for sampling and inference in log-linear models based on solving a sequence of MIPS randomly perturbed by Gumbel random variables.

In order to use log-linear models, it is often necessary to provide samples from the model or to estimate Z_θ^{-1} for various values of θ. For example, samples can be used to estimate an expectation for gradient descent (see equation 5) and Z_θ^{-1} can be used for inference, such as model averaging (see equation 7).

Here we present a method to accomplish these tasks using a combination of MIPS and Gumbel variables. As seen in equation 12, we can use the argmax property of the Gumbel (equation 11) to get t samples from the model.

Theorem 3.1. *Using the previous notation, let*

$$H = \max_i \{\theta \cdot \phi(x_i) + G_i\} \tag{15}$$

then e^{-H} is an exponential variable with rate Z_θ.

Proof. From the maximum property of the Gumbel (equation 10), H has the same distribution as $\log(Z_\theta) + G$ where G is a Gumbel variable. Thus, if U is a uniform variable on the unit interval,

$$H \sim -\log(-\log(U)) + \log(Z_\theta) \tag{16}$$

which implies

$$e^{-H} \sim -\log(U) Z_\theta^{-1} \tag{17}$$

which is an exponential random variable with rate Z_θ. □

Because of this theorem, e^{-H} has mean Z_θ^{-1} and variance Z_θ^{-2}. Thus, we can take t samples of e^{-H} to estimate Z_θ^{-1}.

One may wonder why such an estimate is necessary and if there is another suitable estimate. We may observe that $\mathbb{E}[H] = \log(Z_\theta) + \gamma$ where γ is the mean of a Gumbel, the Euler-Mascheroni constant. From this, we might attempt to to estimate Z_θ^{-1} by $\exp(-(\overline{H} - \gamma))$ where $\overline{H}$ is the average of a sample of several H. While this is a consistent estimator, it is not unbiased and has a asymptotic variance of $(\pi^2/6) Z_\theta^{-2}$ which is $\pi^2/6 \approx 1.645$ times the variance of our proposed estimator. Another estimate we might consider is e^H. However, this won't behave well as it does not have a finite expectation.

3.1. MIPS Formulation

The key contribution of this work is to formulate this as a MIPS problem. Denote vector concatenation by $(\cdot, \cdot)$. Intuitively, we can concatenate a series of k Gumbel variables to the end of the feature vectors and concatenate an elementary vector to the end of the query to access just one Gumbel variable at a time. Let, $q_j = (\theta, e_j)$ and $v_i = (\phi(x_i), \{G_{i,j}\}_j)$ where $j = 1..t$ and e_j is the j^{th} elementary vector. If we define,

$$S_j = \operatorname*{argmax}_i q_j \cdot v_i \tag{18}$$

$$H_j = \max_i q_j \cdot v_i \tag{19}$$

then $\{S_j\}_j$ are independent samples from the model and $\{H_j\}_j$ can be used for a Monte Carlo estimate of the inverse partition function, as described above. Thus, our query is q_j and we perform preprocessing on $V = \{v_i\}_i$.

This method is effective in situations where the input data $\mathcal{X}$ is fixed but we wish to perform sampling or inference for various values of θ. In these cases, the preprocessing time is amortized over the various values of θ and thus our inference and sampling have the time complexity of the query time, which can be sub-linear for MIPS methods.

Intuitively, we reduce the problem of sampling and inference to a well-studied optimization problem (Ermon et al., 2013a;b;c; Achlioptas & Jiang, 2015; Hsu et al., 2016). Thus, for situations where the same $\mathcal{X}$ are used with various values of θ, we solve a similar optimization problem. This opens up the problem to be solved by MIPS techniques. While the naive, standard method performs a computation on $\mathcal{X}$ for each value of θ, our method allows for MIPS preprocessing to avoid wasting computation.

One thing to note about this method is that the estimates for various θ will have the correct marginal distribution (with respect to drawing the initial Gumbels) but will not be independent. In particular, there will be a dependence for very similar or equal values of θ. It should be noted that samples obtained using Markov Chain Monte Carlo methods are also typically not independent. In our case, this dependence can be mitigated by sampling kt sets of N Gumbels (each set has one Gumbel for each state) instead of just t. Then, use a random sample of t sets each time we wish to perform sampling or inference. This works empirically as shown in our experiments.

Algorithm 1 MIPS-Gumbel Initialization

Input: data $\{\phi(x_i)\}_{i=1}^N$, t, k
for $i = 1$ **to** N **do**
  for $j = 1$ **to** kt **do**
    Sample Gumbel and save to $G_{i,j}$
  end for
  Set $v_i = (\phi(x_i), \{G_{i,j}\}_j)$
end for
Create MIPS data structure M from $\{v_i\}_{i=1}^N$

Algorithm 2 MIPS-Gumbel t Samples

Input: MIPS data structure M, Parameters θ
Randomly choose subset J of size t from $\{1, .., kt\}$
Initialize $S = \emptyset$
for j **in** J **do**
  $q_j = (\theta, e_j)$
  Query M with q_j and add the argmax to S
end for
Return: S

3.2. Algorithm

Our proposed approach has two phases. The first one is a preprocessing or initialization phase, that sets up the appropriate data structures to later speed up a sequence of MIPS queries. The pseudocode is presented in Algorithm 1. This algorithm only needs to be run once for input data $\mathcal{X}$. The sampling algorithm is shown as Algorithm 2. This returns t independent samples, which, for instance, could be used to estimate the expectation of $\phi(x)$. The inverse partition estimation algorithm is shown as Algorithm 3. This algorithm returns a numerical estimate from averaging t samples.

Algorithm 3 MIPS-Gumbel Inverse Partition Estimate

Input: MIPS data structure M, Parameters θ
Randomly choose subset J of size t from $\{1, .., kt\}$
Initialize $S = \varnothing$
for j **in** J **do**
  $q_j = (\theta, e_j)$
  Query M with q_j and save the max as H_j
  add e^{-H_j} **to** S
end for
$Estimate = \frac{\sum_{s \in S} s}{t}$
Return: $Estimate$ (Estimate for Z_θ^{-1})

3.3. Analysis

3.3.1. Runtime

Theorem 3.2. *Suppose we have a MIPS technique with $O(f(N))$ query time. Then the MIPS reduction technique can produce t exact samples from the model or estimate the inverse partition function with t unbiased Monte Carlo samples with known variance in $O(f(N)t)$ time.*

Proof. Once the data structure is built, to gain t samples from the model or t Monte Carlo, we require querying the MIPS data structure t times. Since each query takes $O(f(N))$, the runtime is $O(f(N)t)$. With exact MIPS, the t samples are exact and the inverse partition function estimate is unbiased and has known variance. □

Since t doesn't increase with N, any sub-linear MIPS technique will improve the runtime for large N. As some examples, $f(N) = O(N^\rho \log(N))$ for LSH methods, and $f(N) = O(\log N)$ for spherical k-means hierarchical clustering. The space-partitioning in (Ram & Gray, 2012) does not provide asymptotic runtime complexity but gives speedups on empirical datasets between 2 and 20,000.

3.3.2. Effect of Approximate MIPS

There has been recent work in improving approximate MIPS rather than exact MIPS. One way to do this is by relying on Locality Sensitive Hashing (Shrivastava & Li, 2014; 2015; Ravichandran et al., 2005).

If an approximate MIPS technique has an additive error guarantee (as is the case for LSH methods, see Theorem 2.1), we can prove bounds for how the error is propagated through our method. Suppose we have an approximate MIPS method that returns a vector with inner product at most c less than the true maximum. Here, bounds are shown regarding the partition function estimation and sampling probabilities.

Theorem 3.3. *For an approximate MIPS method with additive error $c > 0$, let the inverse partition function estimate using Gumbels and exact MIPS be $\hat{Z}^{-1}$ and the estimate with approximate MIPS be $\widetilde{Z}^{-1}$. Then,*

$$\hat{Z}^{-1} \leq \widetilde{Z}^{-1} \leq e^c \hat{Z}^{-1} \tag{20}$$

Thus, using an approximate MIPS method will not make the estimate smaller but could increase it by up to a factor of e^c.

Theorem 3.4. *For an approximate MIPS method with additive error $c > 0$, let the sample with the approximate MIPS be $\widetilde{i}$. Then,*

$$e^{-c} P_\theta(x_k) \leq P(\widetilde{i} = k) \leq e^c P_\theta(x_k) \tag{21}$$

Thus, sampling with an approximate MIPS method at most affects the sampling probabilities by a factor of e^c. The proofs of these two theorems are in the appendix.

4. Experimental Results

The main purpose of our empirical evaluation is to demonstrate that our MIPS reduction using Gumbels (MRG) doesn't affect the accuracy of sampling or inference. We show the results of the reduction on model averaging and learning via gradient descent, two tasks introduced in the Background section. We also show the empirical speedup achieved using a particular MIPS technique.

We compare the results of the exact methods with the results of the MRG presented in this work for gradient descent and model averaging. We use naive exact MIPS as the black box MIPS solver for MRG. The runtime depends on the particular MIPS technique used. As noted in Section 3.3.1, we use $O(f(N)t)$ time as opposed to $O(N)$ time for each gradient computation or Monte Carlo estimate, where $f(N)$ depends on the specific MIPS technique used. In the case of approximate MIPS, $f(N)$ can be chosen to trade off accuracy vs. computational cost, as in Theorem 2.1. We emphasize that we can use any existing MIPS technique as a black box within our framework. To empirically show an example of the speedup for one particular MIPS technique, we run gradient descent with the hierarchical spherical k-means (HSKM) approximate MIPS technique (Auvolat et al., 2015).

4.1. Data

We use synthetic data and real data in these experiments. The synthetic data was normally distributed with 200 dimensions and N=10,000 states.

The real data we use is the word2vec dataset, a word embedding dataset released by Google (Mikolov et al., 2013a;b). This dataset is made by generating tokens for words and short phrases and then learning a probabilistic

model to find vector embeddings. We used the Continuous Bag of Words (CBOW) model which has an implicit log linear model (Mikolov et al., 2013a). The log linear model attempts to predict a word from the surrounding context and the resulting embeddings can be used as a meaningful featurization of the words. Our results are run on a dataset with $N = 681,321$ data points and 200 dimensions.

4.2. Model Averaging

For the model averaging task, we have the states $\mathcal{X}$ and a fixed θ_0. We can compute the probabilities of the elements of $\mathcal{X}$ according to the log-linear model using θ_0 as the parameters (Equation 1). For this task, we wish to introduce model uncertainty by averaging over a distribution of θ. In this case, we use a normal distribution with mean θ_0.

The two methods we compare is a Monte Carlo estimate using exact probabilities and a Monte Carlo estimate using our inference technique with our MIPS Reduction using Gumbels (MRG) using exact MIPS. We used 10,000 Monte Carlo samples. For the MIPS reduction, $k = 5$ and $t = 100$. In general, a larger k will make the samples for different θ less dependent and a larger t will decrease the variance of the estimate.

We show that the two methods provide similar results in our experiments, implying that our method is accurate. We evaluate the techniques by plotting the ratio of the probability of the model averaging methods to the probability computed without model averaging, i.e., evaluated using the MAP parameter estimate θ_0. Ideally, we would like to have similar ratios for the two methods (meaning that their estimates are similar).

4.2.1. Synthetic Data

For the synthetic data, we sample θ_0 in a random direction from a normal distribution.

The results can be seen in Figure 1. To evaluate the performance on a mix of states with high and low probabilities, results are shown for the 1^{st}, 2^{nd}, 3^{rd}, 10^{th}, 100^{th}, and 1000^{th} most likely states. We will refer to this as the probability ranking of the states. The probabilities of these points without model averaging are in Table 3 in the Appendix.

The horizontal black line is the value without model averaging. To plot on the same axes, the probabilities for each point were scaled to the value without model averaging. It can be seen that the results of our MRG method closely align to the exact method, and both give results different from those without model averaging.

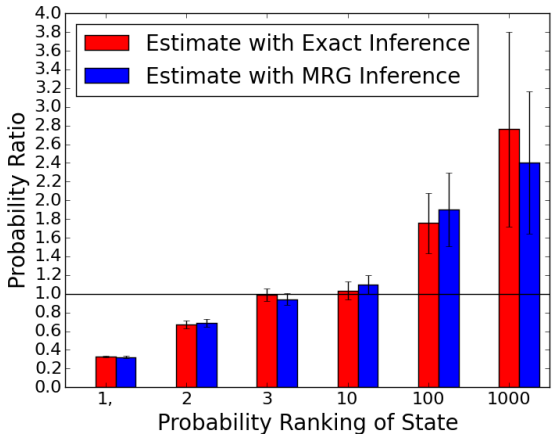

Figure 1. The estimated probabilities (as a ratio to the probability w.r.t. MAP parameter estimate) for various elements of $\mathcal{X}$ in the synthetic dataset using exact inference and inference with MRG. The horizontal line is without model averaging and two standard deviations are shown as error bars. Our MRG technique achieves the same level of accuracy.

4.2.2. Word2vec Data

The results for the word2vec data were created in the same way as in the synthetic data. See the resulting probability ratios in Figure 2. Again, we see that our MRG method matches the exact method, and that both are different from the results without model averaging.

The θ_0 was computed according to the implicit log-linear model using the context of the sentence that was chosen beforehand for evaluation: "it has survived the stresses _____ flight to put it through". Intuitively, this is a probabilistic model for the word used to fill the gap. We show the results for the five most likely words, and then powers of ten, i.e. for the states (words) with probability ranking 1, 2, 3, 4, 5, 10, 100, 1000, 10000, and 10000. See Table 4 in the appendix for the probabilities of these states and the corresponding word or short phrase.

4.3. Gradient Descent with word2vec Data

We also show the word2vec results for the gradient descent algorithm. Each θ defines a probability distribution over the words in the dataset using the word embeddings as the features $\phi(x)$. The task is to maximize the likelihood of a subset of words from a specific category, where a Gaussian prior is put on the parameters. We achieve this by using Equation 5 with an extra term for the Gaussian prior.

We compare the results of gradient descent with the exact gradient and of gradient descent using samples from our MIPS reduction. Ideally, the most likely words under the learned model will be from the category. However, for a

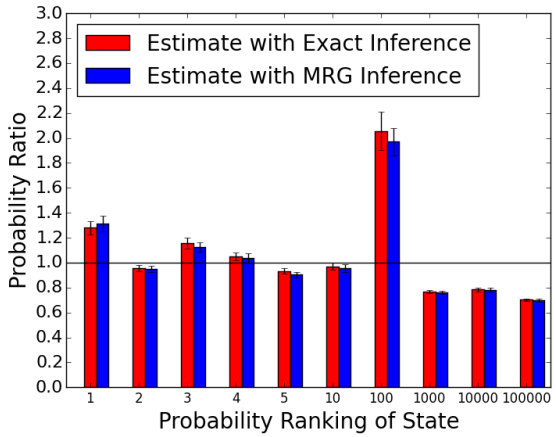

Figure 2. The estimated probabilities (as a ratio to the probability w.r.t. MAP parameter estimate) for various output labels in the word2vec dataset using the exact method and our MRG method. The horizontal line is without model averaging and two standard deviations are shown as error bars. Our MRG technique achieves the same level of accuracy.

good evaluation of whether the MIPS reduction hurts the accuracy, there needs to be a difference between solving with the MLE and a naive method. Note that the mean of the input words is a simple heuristic that works well in general, but fails in some cases. Such a case is the basketball category, which can be confused with other sports. We use this category and choose a number of basketball words that can be found in the appendix in Table 5.

For this experiment, we also include results from using the hierarchical spherical k-means algorithm (HSKM) (Auvolat et al., 2015) as the MIPS method. This is not meant to give an exhaustive comparison of the MIPS techniques, but rather show that speedups that are possible.

We compare the results of the mean heuristic to the exact method and our MRG method, both with exact MIPS and with HSKM. We computed the top 50 words from the four methods and report the number of non-basketball mistakes in Table 1. The full list of words can be seen in Table 6 and Table 7 in the appendix. It can be seen that the heuristic performs poorly while both exact and and our MRG with exact MIPS work well. Introducing the approximate MIPS technique adds error so that it yields results between the exact and the mean heuristic.

As another comparison, the log posterior probability of the data and parameters can also be seen in Table 1. Note that a scaling constant was optimized for a fair comparison with the mean heuristic. We can see that our MRG method with exact MIPS yields almost the same value as the exact method, while the mean heuristic and our MRG method with HSKM perform significantly worse.

Method	# Mistakes	Log Prob.
Exact Gradient	3	-7.15
MRG Method: Exact MIPS	3	-7.18
MRG Method: HSKM MIPS	7	-8.70
Mean Heuristic	13	-8.91

Table 1. A table of the mistakes and posterior probability of the specification words for the basketball category. We show the result of gradient descent with the exact method; gradient descent with our MRG method, with exact MIPS and HSKM MIPS, and the mean heuristic. We can see that our MRG method with exact MIPS has similar results to the exact method. HSKM allows us to trade off accuracy for speed in a principled way.

Method	Seconds per Iteration
Exact Gradient	4.7
MRG Method: HSKM MIPS	0.73

Table 2. Runtime for each gradient descent iteration for exact method and our MRG method using hierarchical spherical k-means approximate MIPS.

Finally, we present the time per gradient iteration in Table 2 to show a 6X-7X speedup. The times are both for a single core running in python. It should be noted that the HSKM comes with around 10 minutes of preprocessing (that could be amortized for multiple runs) and that the MRG iterations are slightly weaker since they are somewhat stochastic (highly dependent on gradient descent specifics). Even when these effects are included for our experiment, the MIPS reduction with HSKM is still faster.

5. Conclusion

In conclusion, we propose an novel way to reduce inference and sampling problems with large output spaces to randomly perturbed MIPS. This method is particularly useful in settings where we wish to perform inference or sampling on a fixed set of states for various parameters.

Our method crucially relies on MIPS, a well studied problem with a number of efficient solutions that perform well both in theory and practice. Our method thus provides a way to take advantage of recent advances in algorithms for MIPS to speed up sampling, partition function estimation, model averaging, and training. Using approximate MIPS techniques, we can trade off accuracy and speed in a principled way, thanks to our novel error analysis.

Future work will examine the how this method interacts with various MIPS techniques. It would be important to understand the tradeoffs between exact MIPS techniques, LSH-based methods, and other approximate MIPS techniques through empirical evaluation.

References

Achlioptas, Dimitris and Jiang, Pei. Stochastic integration via error-correcting codes. In *Proc. Uncertainty in Artificial Intelligence*, 2015.

Andreas, Jacob, Rabinovich, Maxim, Jordan, Michael I, and Klein, Dan. On the accuracy of self-normalized log-linear models. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1774–1782, 2015.

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

Bachrach, Yoram, Finkelstein, Yehuda, Gilad-Bachrach, Ran, Katzir, Liran, Koenigstein, Noam, Nice, Nir, and Paquet, Ulrich. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender systems*, 2014.

Bengio, Yoshua, Schwenk, Holger, Senécal, Jean-Sébastien, Morin, Fréderic, and Gauvain, Jean-Luc. Neural probabilistic language models. In *Innovations in Machine Learning*, pp. 137–186. Springer, 2006.

Chakraborty, Supratik, Meel, Kuldeep, and Vardi, Moshe. A scalable approximate model counter. In *Proc. of the 19th International Conference on Principles and Practice of Constraint Programming (CP)*, pp. 200–216, 2013.

Charikar, Moses. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th annual ACM Symposium on Theory of Computing (STOC)*, 2002.

Curtin, Ryan, Ram, Parikshit, and Gray, Alexander. Fast exact max-kernel search. In *Proceedings of SDM*, 2013.

Devlin, Jacob, Zbib, Rabih, Huang, Zhongqiang, Lamar, Thomas, Schwartz, Richard M, and Makhoul, John. Fast and robust neural network joint models for statistical machine translation. In *ACL (1)*, pp. 1370–1380. Citeseer, 2014.

Ermon, Stefano, Gomes, Carla P, Sabharwal, Ashish, and Selman, Bart. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *ICML*, 2013a.

Ermon, Stefano, Gomes, Carla P, Sabharwal, Ashish, and Selman, Bart. Optimization with parity constraints: From binary codes to discrete integration. In *UAI*, 2013b.

Ermon, Stefano, Gomes, Carla P., Sabharwal, Ashish, and Selman, Bart. Embed and project: Discrete sampling with universal hashing. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2085–2093, 2013c.

Ermon, Stefano, Gomes, Carla P., Sabharwal, Ashish, and Selman, Bart. Low-density parity constraints for hashing-based discrete integration. In *Proc. of the 31st International Conference on Machine Learning (ICML)*, pp. 271–279, 2014.

Friedman, Jerome H. and Tukey, John W. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 1974.

Gershman, Sam and Goodman, Noah D. Amortized inference in probabilistic reasoning. In *Proceedings of the Thirty-Sixth Annual Conference of the Cognitive Science Society*, 2014.

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

Hadjis, Stefan and Ermon, Stefano. Importance sampling over sets: A new probabilistic inference scheme. In *UAI*, 2014.

Hazan, Tamir, Maji, Subhransu, and Jaakkola, Tommi. On sampling from the gibbs distribution with random maximum a-posteriori perturbations. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2013.

Henzinger, Monika. Finder near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.

Hsu, Lun-Kai, Achim, Tudor, and Ermon, Stefano. Tight variational bounds via random projections and I-projections. In *AISTATS*, 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 (STOC)*, 1998.

Kim, Carolyn, Sabharwal, Ashish, and Ermon, Stefano. Exact sampling with integer linear programs and random perturbations. In *Proceedings of AAAI*, 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*, 2012.

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

Kulis, Brian and Grauman, Kristen. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of the IEEE Internation Conference on Computer Vision (ICCV)*, 2009.

Lafferty, John D., McCallum, Andrew, and Pereira, Fernando C. N. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pp. 282–289, 2001. ISBN 1-55860-778-1.

Maddison, Chris, Tarlow, Daniel, and Minka, Tom. A* sampling. In *Proceedings of Advances in Neural Information Processing Systems*, 2014.

Manku, Gurmeet Singh, Jain, Arvind, and Sarma, Anish Das. Detecting near-duplicates for web crawling. In *Proceedings of the 16th International World Wide Web Conference*, 2007.

Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Efficient estimation of word representations in vector space. In *Proceedings of Workshop at ICLR*, 2013a.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*, 2013b.

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

Neyshabur, Behnam and Srebro, Nathan. On symmetric and asymmetric lshs for inner product search. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 1926–1934, 2015.

Papandreou, George and Yuille, Alan. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *Proceedings of ICCV*, 2011.

Ram, Parikshit and Gray, Alexander. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2012.

Ravichandran, Deepak, Pantel, Patrick, and Hovy, Eduard. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL)*, 2005.

Roth, Dan. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1):273–302, 1996.

Shen, Fumin, Liu, Wei, Zhang, Shaoting, Yang, Yang, and Shen, Heng Tao. Learning binary codes for maximum inner product search. In *Proceedings of The IEEE International Conference on Computer Vision (ICCV)*, 2015.

Shrivastava, Anshumali and Li, Ping. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Proceedings of the 27th Advances in Neural Information Processing Systems (NIPS) conference*, 2014.

Shrivastava, Anshumali and Li, Ping. Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). In *In Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015.

Tieleman, Tijmen. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pp. 1064–1071. ACM, 2008.

Zhong, Shi. Efficient online spherical k-means clustering. In *Proceedings of International Joint Conference on Neural Networks*, 2005.

Zhu, Michael and Ermon, Stefano. A hybrid approach for probabilistic inference using random projections. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 2039–2047, 2015.

6. Appendix

6.1. Proofs of Theorems 3.3 and 3.4

Theorem 3.3. *For an approximate MIPS method with additive error $c > 0$, let the inverse partition function estimate using Gumbels and exact MIPS be $\hat{Z}^{-1}$ and the estimate with approximate MIPS be $\widetilde{Z}^{-1}$. Then,*

$$\hat{Z}^{-1} \leq \widetilde{Z}^{-1} \leq e^{c}\hat{Z}^{-1}$$

Proof. Let H be defined as before and let $\hat{H}$ be the corresponding value with the approximate method. Since the approximate method has additive error c,

$$H - c \leq \hat{H} \leq H$$

$$e^{c}e^{-H} \geq e^{-\hat{H}} \geq e^{-H}$$

Using the notation from the theorem statement, $\hat{Z}^{-1} = e^{-H}$ and $\widetilde{Z}^{-1} = e^{-\hat{H}}$. Thus,

$$e^{c}\hat{Z}^{-1} \geq \widetilde{Z}^{-1} \geq \hat{Z}^{-1}$$

□

Theorem 3.4. *For an approximate MIPS method with additive error $c > 0$, let the sample with the approximate MIPS be $\widetilde{i}$. Then,*

$$e^{-c}P_\theta(x_k) \leq P(\widetilde{i} = k) \leq e^{c}P_\theta(x_k)$$

Proof. Let $\widetilde{i}$ be the maximum from the approximate MIPS. For convenience, let $a_i = \theta \cdot \phi(x_i)$. Let G_i be the corresponding Gumbel variables. Note that,

$$P_\theta(x_k) = P(k = \operatorname*{argmax}_j a_j + G_j)$$
$$P_\theta(x_k) = P(a_k + G_k > \max_{j \neq k} a_j + G_j)$$

Let us examine the probability of sampling a particular k with the approximate MIPS method.

$$P(\widetilde{i} = k) \leq P(\max_{j \neq k} a_j + G_j - c < a_k + G_k)$$
$$P(\widetilde{i} = k) \geq P(\max_{j \neq k} a_j + G_j < a_k + G_k - c)$$

The maximum has a distribution of $\log(Z - e^{a_k}) + G'$ where G' has a Gumbel distribution.

$$P(\widetilde{i} = k) \leq P(\log(Z - e^{a_k}) + G' - c < a_k + G_k)$$

$$P(\widetilde{i} = k) \geq P(\log(Z - e^{a_k}) + G' < a_k + G_k - c)$$

Rearranging terms,

$$P(\widetilde{i} = k) \leq P(G' - G_k < a_k - \log(Z - e^{a_k}) + c)$$

$$P(\widetilde{i} = k) \geq P(G' - G_k < a_k - \log(Z - e^{a_k}) - c)$$

Since the difference of two Gumbel distributions is a logistic distribution,

$$P(\widetilde{i} = k) \leq \frac{1}{1 + e^{-(a_k - \log(Z - e^{a_k}) + c)}}$$

$$P(\widetilde{i} = k) \geq \frac{1}{1 + e^{-(a_k - \log(Z - e^{a_k}) - c)}}$$

And thus,

$$e^{-c}\frac{e^{a_k}}{Z} \leq P(\widetilde{i} = k) \leq e^{c}\frac{e^{a_k}}{Z}$$

$$e^{-c}P_\theta(x_k) \leq P(\widetilde{i} = k) \leq e^{c}P_\theta(x_k)$$

□

6.2. Tables

Here we present some tables from the experimental results section. The tables with the probabilities without model averaging for the states that were presented in Section 4.2 are shown as Table 3 and 4. The probabilities of the words generally make sense given that the model is a bag-of-words-like model. The "point_b" may appear odd at first, but is often used in conjunction with flights as in the context of the phrase "from point A to point B".

The words used to train the basketball category in Section 4.3 can be seen in Table 5. The top 50 words for the generative models learned by the four different methods mentioned in Section 4.3 can be seen in Table 6 and 7. The training words appear in black black, the other basketball-related words appear in blue, and the non-basketball-related words appear in red. The non-basketball-related words are counted as mistakes.

For the HSKM method, we used $p = 100$ as a beam size and $b = 10$ as a branching factor. We also used $k = 1$ and $t = 100$.

Order	Probability
1	0.5122
2	0.05799
3	0.02643
10	0.01056
100	6.632e-4
1000	1.6336e-5

Table 3. Probabilities (without using model averaging) used for synthetic model averaging inference

Order	Word	Probability
1	point_b	2.8917e-5
2	stresses	2.0855e-5
3	turbulence	2.0143e-5
4	ordeal	1.8597e-5
5	stress	1.7599e-5
10	important_thing	1.5307e-5
100	formal_training	1.0142e-5
1000	gabby's_seat	6.7475e-6
10000	garamba	4.2853e-6
100000	joe_hart_vincent_kompany	2.3600e-6

Table 4. Words and probabilities (without using model averaging) for word2vec model averaging inference

Basketball-related Words
hoop
shoot
basket
dribble
pass
three_pointer
key
sideline
bench
court
coach
player
center
guard

Table 5. The basketball-related words used for training the basketball category for the gradient descent experiment

Rank	Exact Method	MRG: Exact MIPS
1	bench	sideline
2	sideline	dribble
3	dribble	bench
4	guard	hoop
5	hoop	ball
6	three_pointer	three_pointer
7	ball	guard
8	basket	basket
9	layup	loose_ball
10	defender	layup
11	loose_ball	timeout
12	teammate	defender
13	shot_clock	shot_clock
14	jump_shot	pointer
15	pointer	jump_shot
16	coach	an_ncaa_college
17	timeout	teammate
18	dribbling	coach
19	with_seconds_left	free_throw
20	player	dribbling
21	free_throw	scrimmage
22	scrimmage	player
23	an_ncaa_college	shoot
24	shoot	with_seconds_left
25	teammates	buzzer
26	receiver	teammates
27	shooting_guard	receiver
28	pass	pass
29	fast_break	dribbled
30	point_guard	shooting_guard
31	buzzer	fast_break
32	yard_line	yard_line
33	dribbled	midcourt
34	defenders	point_guard
35	midfield	referee
36	free_throws	an_nba_basketball
37	an_nba_basketball	defenders
38	assistant_coach	quickness
39	referee	free_throws
40	midcourt	perimeter
41	locker_room	griner
42	perimeter	midfield
43	quickness	locker_room
44	griner	inbounds_pass
45	scoring	assistant_coach
46	playmaker	jump_shots
47	court	basketball
48	playmaking	court
49	jump_shots	inbounded
50	inbounds_pass	forward

Table 6. Top 50 words according to models learned by gradient descent with exact method and with our Gumbel reduction using exact MIPS. The training words are black, basketball-related words are blue, and non-basketball-related words are red.

Rank	MRG: HSKM MIPS	Mean Heuristic
1	layup	layup
2	three_pointer	pointer
3	pointer	three_pointer
4	ball	ball
5	loose_ball	loose_ball
6	free_throws	free_throws
7	free_throw	end_zone
8	jump_shot	with_seconds_left
9	two_free_throws	yard_touchdown
10	fast_break	free_throw
11	with_seconds_left	two_free_throws
12	shot_clock	an_ncaa_college
13	timeout	puck
14	puck	jump_shot
15	dribble	dunk
16	dunk	timeout
17	basket	yard_line
18	dunks	field_goal
19	layups	yard_field_goal
20	end_zone	fast_break
21	penalty_area	an_alley_oop
22	an_alley_oop	bench
23	three_pointers	sideline
24	yard_touchdown	yard_touchdown_pass
25	jumper	three_pointers
26	driving_layup	basket
27	midfield	th_minute
28	bench	penalty_area
29	field_goal	dunks
30	backboard	midfield
31	reverse_layup	scrimmage
32	yard_line	dribble
33	technical_foul	shot_clock
34	sideline	jumper
35	free_throw_line	touchdown
36	dribbled	technical_foul
37	buzzer	yard_gain
38	scrimmage	yard_pass
39	an_ncaa_college	layups
40	midcourt	driving_layup
41	foul	final_seconds
42	ump_shots	with_seconds_remaining
43	an_offensive_rebound	reverse_layup
44	th_minute	teammate
45	with_seconds_remaining	free_throw_line
46	short_jumper	dribbled
47	point_guard	an_nba_basketball
48	fouls	midcourt
49	foul_line	yard_run
50	teammates	teammates

Table 7. Top 50 words according to models learned by gradient descent with our Gumbel reduction using HKSM and with the mean heuristic. The training words are black, basketball-related words are blue, and non-basketball-related words are red.