Mutual Information Estimation using LSH Sampling

Ryan Spring[*] and **Anshumali Shrivastava**
Rice University, Houston, Texas, USA
rdspring1@rice.com , anshumali@rice.edu

Abstract

Learning representations in an unsupervised or self-supervised manner is a growing area of research. Current approaches in representation learning seek to maximize the mutual information between the learned representation and original data. One of the most popular ways to estimate mutual information (MI) is based on Noise Contrastive Estimation (NCE). This MI estimate exhibits low variance, but it is upper-bounded by $log(N)$, where N is the number of samples.

In an ideal scenario, we would use the entire dataset to get the most accurate estimate. However, using such a large number of samples is computationally prohibitive. Our proposed solution is to decouple the upper-bound for the MI estimate from the sample size. Instead, we estimate the partition function of the NCE loss function for the entire dataset using importance sampling (IS). In this paper, we use locality-sensitive hashing (LSH) as an adaptive sampler and propose an unbiased estimator that accurately approximates the partition function in sublinear (near-constant) time. The samples are correlated and non-normalized, but the derived estimator is unbiased without any assumptions. We show that our LSH sampling estimate provides a superior bias-variance trade-off when compared to other state-of-the-art approaches.

1 Introduction

The goal of representation learning is to learn a high-level representation from raw data that is useful for a variety of downstream, supervised learning tasks. Instead of training a separate network for each task, we will use the common high-level representation for all tasks. This paradigm is very successful for natural-language (NLP) tasks [Devlin *et al.*, 2018]. State-of-the-art results are obtained from transformer models that are pretrained on large text corpuses and then fine-tuned on different supervised NLP tasks.

There is great interest in applying this self-supervised pretraining paradigm to other domains [Oord *et al.*, 2018a; Hjelm *et al.*, 2018; Bachman *et al.*, 2019]. Current approaches in representation learning seek to maximize the mutual information between the learned representation and original observations, inspired by the infoMax principle [Linsker, 1988; Bell and Sejnowski, 1995]. In this setting, the raw observations are viewed as samples from an underlying distribution, $x \sim p(x)$. The goal is to learn a probabilistic mapping $p(y|x)$ that retains information about the original data x.

One of the most effective approaches to maximize mutual information (MI) uses the Noise Contrastive Estimation (NCE) loss function. Essentially, the objective is to distinguish the single positive sample from the other negative samples. By minimizing the NCE loss function, we will maximize the mutual information between the representation and data. The NCE estimate of the mutual information exhibits low variance but is biased. Its estimate is upper-bounded by $log(N)$ where N is the number of negative samples. If the true mutual information value is very high, then an extremely large number of samples is necessary for an accurate estimate.

For NCE MI estimation, there are $O(N^2)$ evaluations for each Monte-Carlo estimate. The diagonal of the matrix contains the positive samples, while the remaining examples are treated as the negative samples. The positive and negative samples are summed together in the partition function.

In an ideal scenario, the remaining items in the entire dataset would be the negative samples. However, using such a large number of samples is computationally prohibitive. Our proposed solution is to decouple the upper-bound of the MI estimate from the batch size by estimating the partition function for the entire dataset using importance sampling (IS). For our importance sampling (IS) estimate, the proposal distribution is locality-sensitive hashing (LSH).

Our Contributions. Our algorithm utilizes LSH tables for fast, adaptive sampling. This work is an auspicious example of the power of using an algorithmic data structure for efficient and accurate statistical estimation. The samples, obtained from the LSH tables, are correlated and unnormalized. This unusual property is not a hurdle and that there exists a simple, unbiased estimator of the partition function using these samples. Fast LSH Sampling allows us to estimate the partition function accurately in near-constant time. This algorithm opens a new direction for sampling and unbiased estimation. We leverage the striking utility of two-decades of LSH research for statistical estimation tasks.

[*] Contact Author

Our sampling scheme has several favorable theoretical properties. In particular, the probability values associated with every sample is monotonic in the probability density assigned by the log-linear distribution. Thus, the proposal has same modes as the target distribution. This property makes our proposal very informative for estimation.

We show that our LSH sampling estimate provides a superior bias-variance trade-off when compared to other approaches such as I_α [Poole *et al.*, 2019] that interpolates between I_{NCE} [Oord *et al.*, 2018b] and I_{NWJ} [Nguyen *et al.*, 2010]. Our LSH sampling algorithm is more accurate at estimating the partition function than the Uniform IS method while being equally fast.

2 Related Work

2.1 Classic Importance Sampling

The partition function can be approximated in an unbiased fashion using Importance sampling (IS). It estimates the partition function Z_θ by drawing samples y from a tractable proposal distribution $y \sim g(y)$, and then averaging the importance weights $f(y)/g(y)$ across the samples. Therefore, an unbiased estimate is returned by using the weight samples, instead of all the items in the dataset.

The simplest proposal distribution is to select items from the dataset uniformly. However, despite the speed of uniform sampling, it requires a large number of samples to obtain an accurate estimate. Also, the estimate often has a high variance when the proposal distribution is very different from the target distribution.

2.2 Mutual Information Estimation

Informally, mutual information measures the information about a random variable gained from observing another random variable. is defined as:

$$I(X;Y) = \mathbf{E}_{p(x,y)}\left[log\frac{p(x|y)}{p(x)}\right] = \mathbf{E}_{p(x,y)}\left[log\frac{p(y|x)}{p(y)}\right]$$

Figure 1: The formal definition of Mutual Information between random variables X and Y.

Recent approaches leverage neural network function approximators to estimate the mutual information between two random variables. In this paper, we focus primarily on two mutual information estimators - InfoNCE [Oord *et al.*, 2018b] and Interpolate I_α [Poole *et al.*, 2019].

The goal of Noise Contrastive Estimation (NCE) is to distinguish the target class from samples from the noise distribution. For mutual information estimation, the positive sample is drawn from the joint distribution $p(x, y)$, while the negative samples are drawn from the marginal distributions $p(x)p(y)$. The upper-bound for the InfoNCE estimate is determined by the number of samples. Our solution is to estimate the partition function with all the items in the dataset, which will increase the upper-bound of the MI estimate without significantly increasing the computational cost.

$$I(X;Y) \geq log(N) + \mathbf{E}_{p(x,y)}\left[log\frac{e^{f(x_i,y_i)}}{\sum_j^N e^{f(x_j,y_i)}}\right]$$

Figure 2: Noise Contrastive Estimation provides a biased estimate of the mutual information between X and Y. This bias is upper-bound by the number of samples N used for the estimate.

The Interpolate I_α estimator merges the InfoNCE and I_{NWJ} estimators together. This approach achieves the best of both estimators, since InfoNCE exhibits low variance but high bias while I_{NWJ} has high variance and low bias. I_{NWJ} [Nguyen *et al.*, 2010] is an unnormalized tractable mutual information estimator by Nguyen, Wainwright, and Jordan (NWJ). For more information about mutual information estimation, [Poole *et al.*, 2019] provides a framework of different mutual information estimators using variational bounds.

3 Locality Sensitive Hashing (LSH)

Locality-Sensitive Hashing (LSH) is a popular, sub-linear time algorithm for approximate nearest-neighbor (ANN) search [Andoni and Indyk, 2004; Gionis *et al.*, 1999]. The high-level idea is to place similar items into the same bucket of a hash table with high probability. An LSH hash function maps an input data vector to an integer key — $h(x) : \mathbb{R}^D \mapsto [0, 1, 2, \ldots, N]$

A collision occurs when the hash values for two data vectors are equal - $h(x) = h(y)$. The collision probability of LSH hash functions is generally a $\mathcal{M}$ monotonic function of the similarity metric for the LSH family.

$$Pr[h(x) = h(y)] = \mathcal{M}(sim(x, y))$$

Essentially, similar items are more likely to collide with each other under the same hash fingerprint. In this paper, we focus on the SimHash LSH family and the Cosine Similarity metric.

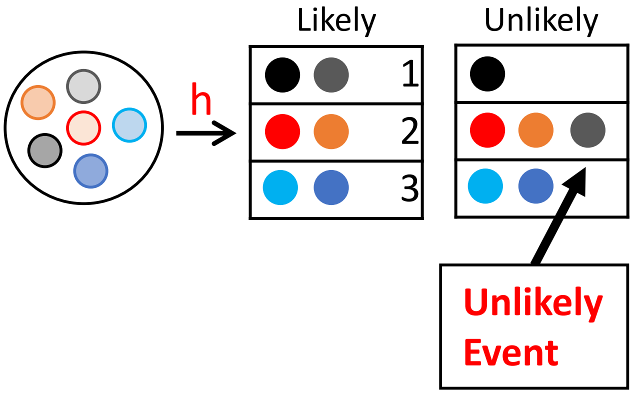

Figure 3: **Locality Sensitive Hashing.** The LSH hash function distributes items based on their similarity. Similar items collide in the same bucket with high probability. However, sometimes items are occasionally placed in a bucket with dissimilar items, which is an unlikely event.

The algorithm uses two parameters - (K, L). We construct L independent hash tables from the collection $\mathcal{C}$. Each hash table has a meta-hash function H that is formed by concatenating K random independent hash functions from $\mathcal{F}$. Given a query, we collect one bucket from each hash table and return the union of L buckets. Intuitively, the meta-hash function makes the buckets sparse (less crowded) and reduces the amount of false positives because only valid nearest-neighbor items are likely to match all K hash values for a given query. The union of the L buckets decreases the number of false negatives by increasing the number of potential buckets that could hold valid nearest-neighbor items.

4 Key Observation: LSH is an Efficient, Informative Sampler in Disguise

The traditional LSH algorithm retrieves a subset of potential candidates for a given query in sub-linear time. For each neighbor in this candidate subset, we compute its actual distance to the query and then report the closest nearest-neighbor. A close observation reveals that an item returned as candidate from a (K, L)-parameterized LSH algorithm is sampled with probability $1 - (1 - p^K)^L$ where p is the collision probability of LSH function [Leskovec *et al.*, 2014]. The precise form of p is defined by the LSH family used to build the hash tables. We can construct a MIPS hashing scheme such that $p = \mathcal{M}(q \cdot x) = \mathcal{M}(\theta_y \cdot x)$ where $\mathcal{M}$ is a monotonically increasing function.

However, the traditional LSH algorithm does not represent a valid probability distribution $\sum_{i=1}^{N} Pr(y_i) \neq 1$. Also, due to the nature of LSH, the sampled candidates are likely to be very correlated. It turns out that there is a simple, unbiased estimator for the partition function using the samples from the LSH algorithm. Please see Appendix A for more information on LSH Sampling and other related applications. We take a detour to define a general class of sampling and partition function estimators where the LSH sampling is a special case.

5 Locality-Sensitive Sampling (LSS) and Unbiased Partition Function Estimator

Here is the description of the sampling process: Assume there is a set of states $Y = [y_1 \dots y_N]$. We associate a probability value p_i with each state y_i. We flip a Bernoulli coin m_i with probability p_i for each state y_i. The sample set S contains all of the states accepted by the Bernoulli sampling process.

$$m_i \sim P(m_i = 1 | p_i) \qquad y_i \in S \iff m_i = 1$$

Note, the probabilities are not required to sum to 1, and that the sampling process is correlated. LSH sampling is a special class of this idealized process with sampling probability:

$$p_i = 1 - (1 - \mathcal{M}(\theta_{y_i} \cdot x)^K)^L$$

Given the sample set S, we have an unbiased estimator for the partition function Z_θ.

Theorem 5.1. *Assume that every state y_i has a weight given by $f(y_i)$ with partition function $Z_\theta = \sum_{y_i \in Y} f(y_i)$. Then we have the following as an unbiased estimator of Z_θ:*

$$Est = \sum_{y_i \in S} \frac{f(y_i)}{p_i} = \sum_{i=1}^{N} \mathbf{1}_{[y_i \in S]} \cdot \frac{f(y_i)}{p_i} \tag{1}$$

$$\mathbb{E}[Est] = \sum_{i=1}^{N} f(y_i) = Z_\theta \tag{2}$$

Theorem 5.2. *The variance of the partition function estimator is:*

$$Var[Est] = \sum_{i=1}^{N} \frac{f(y_i)}{p_i}^2 - \sum_{i=1}^{N} f(y_i)^2 \tag{3}$$

$$+ \sum_{i \neq j} \frac{f(y_i) f(y_j)}{p_i p_j} \mathrm{Cov}(\mathbf{1}_{[y_i \in S]} \cdot \mathbf{1}_{[y_j \in S]}) \tag{4}$$

If the states are selected independently, then we can write the variance as:

$$Var[Est] = \sum_{i=1}^{N} \frac{f(y_i)}{p_i}^2 - \sum_{i=1}^{N} f(y_i)^2 \tag{5}$$

Note 1. In general, this sampling process is inefficient. We need to flip coins for every state in order to generate the sample set S. For log-linear models with feature vector x and function $f(y_i) = e^{\theta_{y_i} \cdot x}$, we show a particular form of probability $p_i = 1 - (1 - \mathcal{M}(\theta_{y_i} \cdot x)^k)^L$ for which this sampling scheme is very efficient. In particular, we can efficiently sample for a sequence of queries x in amortized near-constant time.

Note 2. In our case, where the probabilities p_i are generated from LSH or Asymmetric LSH for Maximum Inner Product Search (MIPS), the term

$$\sum_{i \neq j} \frac{f(y_i) f(y_j)}{p_i p_j} \mathrm{Cov}(\mathbf{1}_{[y_i \in S]} \cdot \mathbf{1}_{[y_j \in S]})$$

contains very large negative terms. For each dissimilar pair y_i, y_j, the term $\mathrm{Cov}(\mathbf{1}_{[y_i \in S]} \cdot \mathbf{1}_{[y_j \in S]})$ is negative. When $\mathbf{1}_{[y_i \in S]} = 1$ and $\mathbf{1}_{[y_j \in S]} = 1$, it implies that y_i and y_j are both similar to the query. Therefore, they are similar to each other due to triangle inequality [Charikar, 2002]. Thus, for random pairs y_i, y_j, the covariance will be negative. If y_i is sampled, then y_j has less chance of being sampled and vice versa. Hence, we can expect the overall variance with LSH-based sampling to be significantly lower than uncorrelated sampling. The exact analysis is significantly challenging and data dependent.

5.1 Why is MIPS the correct LSH function?

The term $\sum_{i=1}^{N} \frac{f(y_i)}{p_i}^2$ in the variance is similar in nature to the $\chi^2(f||p)$ term in the variance of Importance Sampling (IS) [Liu *et al.*, 2015]. The variance of the IS estimate is high when the target f and the proposal p distributions are peaked differently. In other words, they give high mass to different parts of the sample space or have different modes. Therefore, for similar reasons as importance sampling, our scheme is likely to have low variance when f and p_i are aligned. It should be noted that there are very specific forms of probability p_i for which sampling is efficient. We show that with the MIPS LSH function, the probabilities p_i and the function $f(y_i) = e^{\theta_{y_i} \cdot x}$ align well.

We have the following relationship between the probability of each state p_i and the unnormalized target distribution $P(y|x, \theta)$.

Theorem 5.3. *For any two states y_1 and y_2:*

$$P(y_1|x;\theta) \geq P(y_2|x;\theta) \iff p_1 \geq p_2$$

where $p_i = 1 - (1 - \mathcal{M}(\theta_{y_i} \cdot x)^K)^L$ and $P(y|x, \theta) \propto e^{\theta_y \cdot x}$

Corollary 5.3.1. *The modes of both the sample and the target distributions are identical.*

6 Methodology

The combination of these observations is a fast, scalable approach for estimating the partition function. The pseudo-code for this process is shown in Algorithms 1 and 2.
Here is an overview of our LSH sampling process:

1. **Preprocesing:** During the preprocessing phase, we use randomized hash functions to build hash tables from the weight vectors θ_y for each state $y \in Y$.
2. **Estimation:** For each partition function estimate, we retrieve the weight vectors from the hash tables with probability $p = 1 - (1 - \mathcal{M}(\theta_{y_i} \cdot x)^K)^L$, which is monotonic w.r.t. the unnormalized density of the weight vector for the state and the feature vector, i.e., monotonic in $e^{\theta_y \cdot x}$.
3. For each weight vector θ_y in the retrieved set S, we calculate this $p = 1 - (1 - \mathcal{M}(\theta_{y_i} \cdot x)^K)^L$, the probability of retrieving the element given the query feature vector.
4. The partition function estimate for the feature vector x is the sum of each unnormalized density $e^{\theta_y \cdot x}$ in the sample set S weighted by the inverse retrieval probability.

$$\hat{Z}_\theta = \textstyle\sum_{i=1}^{N} 1_{[y_i \in S]} \cdot \frac{f(y_i)}{p_i}$$

Running Time. The total running time includes the $K \times L$ hash computations, followed by evaluating the formula over the samples returned from the LSH hash tables, which has complexity $O(|S|)$. The first thing to observe is that the parameters K and L control the sample size S. By increasing K linearly, there is an exponential drop in the sample size. We can fix K, the number of bits in the hash fingerprint, such that the expected number of samples in each table is a small constant. LSH theory [Indyk and Motwani, 1998] states that we can ensure a constant number of samples from each hash table when $K = \log n$. Thus, with $K = \log n$ and a small number of hash tables L, we can easily obtain a constant sample size, independent of the total number of states n. The total computational cost is on the order of $\log n$.

On the contrary, the same LSH needs $K = \log n$ and a significant number of hash tables $L = n^\rho$ to ensure that an approximate near-neighbor is chosen with high probability. In practice, $\rho < 1$ can still be very large and close to 1. We do not require any such near-neighbor guarantee and can work well with a small, constant-sized number of hash tables L.

Algorithm 1 LSS - Preprocessing

```
HT = Create(k, L)
for each $y \in Y$ do
    Insert(HT, $\theta_y$)
end for
Return: HT

k - Number of Bits per Hash Fingerprint
L - Number of Hash Tables
HT - LSH Hash Tables
$[\theta_y]_{y \in Y}$ weight vectors
```

Algorithm 2 LSS - Partition Estimate

```
$total = 0$
$union$ = Query(HT, x)
for each $y \in union$ do
    $p = \mathcal{M}(x, \theta_y)$
    $weight = 1 - (1 - p^k)^L$
    $logit = e^{\theta_y \cdot x}$
    $total \mathrel{+}= \frac{logit}{weight}$
end for
Return: $\hat{Z}_\theta = total$

$p$ - LSH Collision Probability
$x$ feature vector
$\hat{Z}_\theta$ Partition Function Estimate
```

7 Wasteful Nearest-Neighbor Approaches

Recent approaches to approximate the partition function Z_θ rely on retrieving the heavy elements using a near-neighbor query. Those approaches leverage a reasonable assumption that the states Y follow a power law distribution. The main idea [Mussmann *et al.*, 2017] is to retrieve the heavy entries in the partition function exactly and to get a random sample estimate for the leftover tail. We argue that these near-neighbor based approaches are inherently wasteful for partition function estimation.

First, any efficient MIPS subroutine generates a larger candidate set C. Then, the candidate set is filtered to find the top-k elements. The filtering step requires computing $\theta_i \cdot x$ for all of the elements in C, sorting their values, and then reporting the top-k elements. Therefore, we are throwing away candidates after computing its contribution to the partition function, which is clearly sub-optimal. In addition, we have to use random sampling on the leftover states to get an unbiased estimator.

Another thing to note is that near-neighbor search is only efficient when the query is very close to neighbors. See related literature on hardness of near-neighbors [He *et al.*, 2012]. Thus, finding a large number of neighbors is either very expensive or very inaccurate with approximate near-neighbors algorithms. [Mussmann *et al.*, 2017] requires around top $O(\sqrt{n})$ neighbors, but such a high number of neighbors cannot be efficiently computed.

The LSH sampler removes all of the disadvantages shown above. It uses all of the candidates retrieved from the data structure, which our theory suggests is also required for accurate estimation. Furthermore, it utilizes the existing randomness in the LSH algorithm itself to guarantee an unbiased estimator, so there is no need for any additional random sampling. Our experiments clearly show that the speed of the LSH sampler is competitive with uniform sampling in practice yet has significantly higher accuracy. Also, the statistical guarantees on bias and variance are independent of the quality of the retrieved candidates. Therefore, we are not forced to rely on any accuracy guarantees for MIPS or approximate near-neighbor search, which leads to significant savings.

8 Mutual Information - Experiments

A popular MI estimator uses the Noise Contrastive Estimation (NCE) loss function. The NCE loss function was originally used to train language models with large vocabularies [Gutmann and Hyvärinen, 2010], but it is also a lower-bound on the mutual information [Oord *et al.*, 2018b]. To maximize the mutual information between the input data and output representation, the NCE loss function trains the model to distinguish the positive sample from the $N - 1$ negative samples where N is the batch size.

The main limitation of NCE estimator is that the maximum mutual information estimate is upper-bounded by $log(N)$. If the mutual information estimate is very high, we will need an exponential number of samples to get an accurate estimate. However, when using a large batch size, the gradient update becomes slow. The ideal batch size for an unbiased MI estimate is the entire dataset, where every data point becomes a negative sample.

In this experiment, we estimate the partition function using our LSH IS approach, reducing the dependency on the batch size used during training. Importance sampling scales the batch size to the number of elements in our proposal distribution — the LSH data structure. Empirically, we demonstrate that LSH importance sampling generates an unbiased estimate with lower variance than uniform importance sampling.

The code[1] for the experiments is available online. We designed the experiments to answer the following four important questions:

1. Does importance sampling alleviate the dependency on the batch size for estimating mutual information using NCE?
2. What is the bias/variance trade-off for our LSH importance sampling approach?

We compare and contrast the following estimators against our approach:

1. **Noise Contrastive Estimation (NCE) [Oord *et al.*, 2018b]:** The baseline MI estimator that uses multiple samples to reduce the estimator's variance, but has high bias because the estimate is limited by $log(N)$ where N is the number of samples in the mini-batch.
2. **Uniform Importance Sampling:** An IS estimate where the proposal distribution is a uniform distribution U[0, N]. All samples are weighted equally.
3. **Interpolate I_α [Poole *et al.*, 2019]:** This approach interpolates the high-bias, low-variance I_{NCE} and low-bias, high-variance I_{NJW} estimators using the α parameter.

We applied the various estimators to a correlated Gaussian problem [Poole *et al.*, 2019]. We used a separable critic architecture where $f(x, y) = g(x)^\intercal f(y)$ where f and g are neural network functions. The X and Y variables are drawn from a 20-d Gaussian distribution with zero mean and correlation ρ. This problem allows us to calculate the exact mutual information.

$$I(X;Y) = -\frac{d}{2}log(1-\rho^2)$$

Figure 4: The mutual information for correlated Gaussian random variables where ρ is the correlation between X and Y and d is the number of dimensions.

By increasing the correlation ρ, we increase the mutual information between X and Y in a controlled manner from 2 to 10. In the experiment, as the mutual information content changes, we see how the different MI estimators behave.

Figure 5 shows the effect of increasing the batch size while using importance sampling to estimate the mutual information. As the batch size increases, the variance for both approaches decreases. However, the LSH Sampling approach is less biased than the Uniform Sampling approach.

The computational cost of the estimators depends on the batch size N and the number of samples K selected by the proposal distribution. For uniform sampling, the batch size and the number of samples are equal $N = K$, so the cost is $O(N^2)$. For LSH Sampling, the number of samples K varies slightly for each example, so the cost is $O(NK)$.

We wanted a fair comparison of the computational cost of the estimators. Therefore, the batch size for uniform IS was selected, so the computational cost between the LSH Sampling and Uniform Sampling approaches were roughly equal.

We compare NCE, Uniform IS, and LSH IS for batch size 50 in Figure 6. NCE is upper-bounded by its batch size $ln(50) = 3.91$. The upper bound for Uniform IS and LSH IS is $ln(50,000) = 10.82$ while using the same amount of computation as NCE. As we increase the mutual information, the InfoNCE estimate plateaus while Uniform IS and LSH IS estimates increase appropriately.

Figure 7 compares LSH IS and the Interpolate estimator for a fixed alpha parameter and different batch sizes. As the batch size increases for the Interpolate I_α estimator, the variance decreases. The LSH IS estimator is more unbiased and has less variance than the I_α estimator for batch size 32 and 64.

9 Conclusion

In this paper, we presented the concept of locality-sensitive sampling (LSS), which uses the locality-sensitive hashing (LSH) data structure as an efficient proposal distribution. The samples are weighted by their LSH probability to obtain an unbiased estimator. Using this LSH sampler, we can estimate the partition function using far fewer samples than the entire state space. We demonstrated the effectiveness of locality-sensitive sampling (LSS) for estimating the mutual information using neural network function approximators.

[1] https://github.com/rdspring1/LSH-Mutual-Information

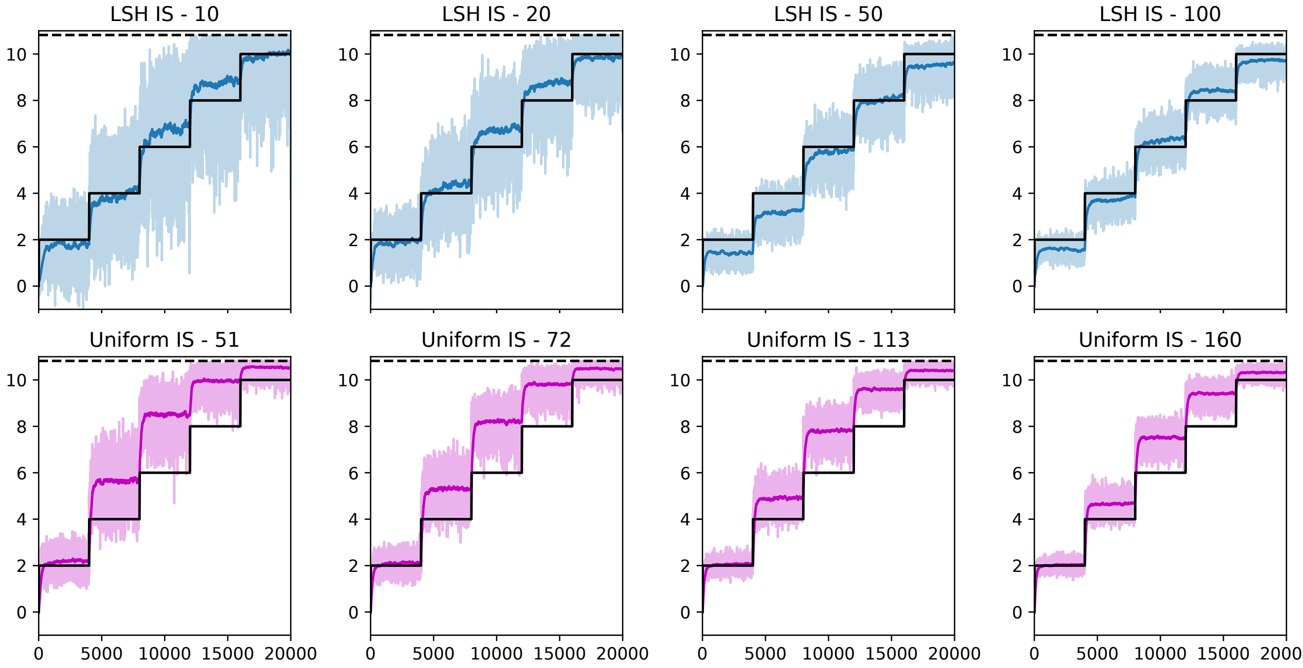

Figure 5: The effect of increasing the batch size while using importance sampling for mutual information estimation - **LSH IS (Top)** and **Uniform IS (Bottom)**. The LSH data structure used $k = 10$ bits and $L = 10$ hash tables. We scaled the size of the sample set to the total size of the dataset - 50K items. Approximately 200-300 samples per query are retrieved from the LSH data structure. The dashed black line is the maximum mutual information estimate. The solid black line is the true mutual information. The batch size and sample size for uniform sampling was selected so that both methods use roughly the same amount of computation.

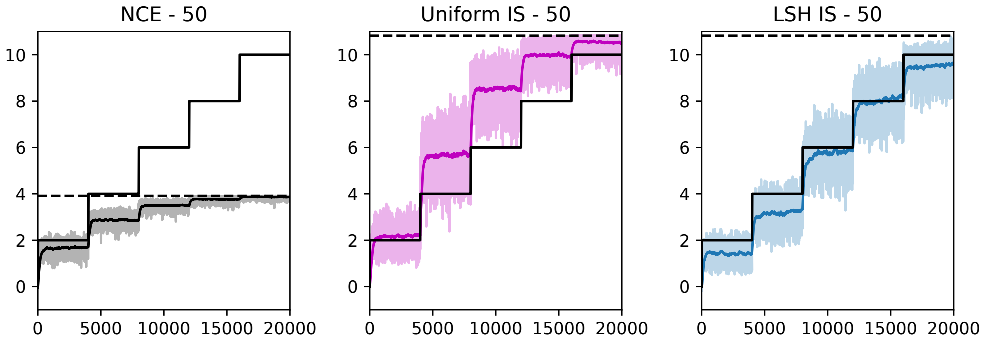

Figure 6: The comparison between NCE, LSH IS, and Uniform IS estimators for batch size 50. NCE is upper-bounded by its batch size — $ln(50) = 3.91$. The upper bound for Uniform IS and LSH IS is $ln(50,000) = 10.82$, while using the same amount of computation as NCE. As the mutual information increases, the NCE estimate plateaus while Uniform IS and LSH IS estimates increase appropriately.

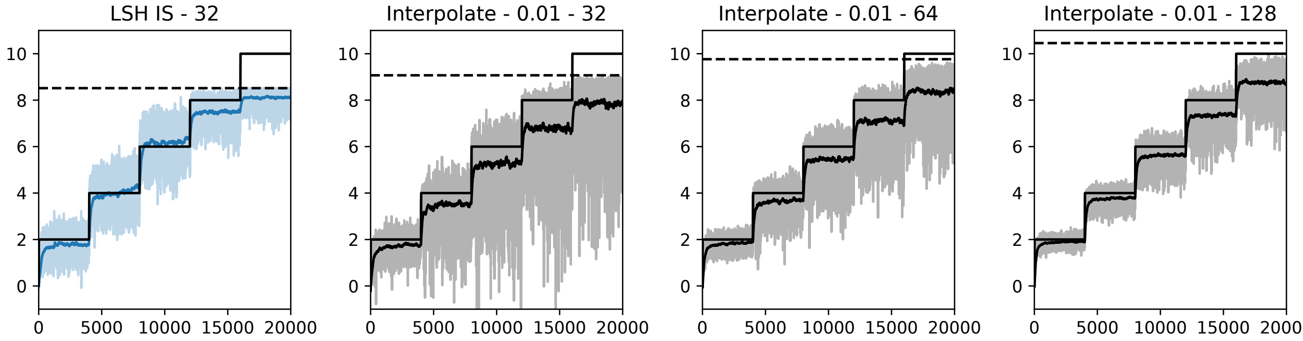

Figure 7: The comparison between LSH IS and Interpolate estimators. The LSH data structure contains 5K items with $k = 8$ bits and $L = 10$ hash tables. The average sample size per query was 91 elements and a 32 batch size. For the interpolate method, $\alpha = 0.01$.

A Locality-Sensitive Sampling

This paper focuses on the new concept of Locality-Sensitive Sampling. Technical portions of this paper appeared in Spring 2017 [Spring and Shrivastava, 2017a]. Traditionally, Locality-Sensitive Hashing (LSH) is used for approximate nearest-neighbor search (ANN). For the ANN algorithm, LSH returns a set of items close to the query. Often times, we filter the returned items to find the closest items to the query.

Locality-Sensitive Sampling views the LSH data structure as a fast, adaptive sampler. Items are inserted into a set of hash tables, according to the locality-sensitive hash function. Then, for a given query, a subset of similar items is retrieved from the hash tables and weighted by the LSH collision probability. Note, the filtering step associated with ANN is not necessary, improving computational performance. Instead of sampling from a probability distribution, we sample items from the LSH data structure. A key difference is that the sampling probabilities for all the items do not sum to 1 with LSH sampling.

A.1 Timeline for LSH Sampling

1. Spring 2016 - Scalable and Sustainable Deep Learning via Randomized Hashing [Spring and Shrivastava, 2017b] — Instead of running all nodes in a neural network, this paper saves computation by sparsely activating a set of nodes sampled adaptively using LSH.
2. Spring 2017 - Scalable Partition Function Estimation via LSH Sampling [Spring and Shrivastava, 2017a] — This paper reduces the computational cost of calculating the normalization constant for the Softmax classifier. It combines Importance Sampling with a SimHash LSH Sampler to find an unbiased estimator for the partition function.
3. FOCS 2017 - Hashing-Based-Estimators for Kernel Density in High Dimensions [Charikar and Siminelakis, 2017] — This paper derives a Hash-Based Estimator (HBE) for Kernel Density Estimation (KDE) in high-dimensions using the Euclidean distance LSH family.
4. AAAI 2018 - Scaling-up Split-Merge MCMC with Locality Sensitive Sampling [Luo and Shrivastava, 2019] — This paper uses a MinHash LSH Sampler as an efficient proposal distribution for Split-Merge MCMC methods.
5. WWW 2018 + WWW 2020 - [Luo and Shrivastava, 2018; Coleman and Shrivastava, 2019] — RACE is a sketching algorithm for kernel density estimation in high-dimensional spaces in a streaming setting. It compresses a set of high-dimensional vectors into an array of integer counters, which is capable of estimating the kernel density for a wide range of problems.
6. NeurIPS 2019 - Fast and Accurate Stochastic Gradient Estimation [Chen *et al.*, 2019] — Locality Gradient Descent (LGD) uses an LSH Sampler to select examples during gradient descent. It is an unbiased estimator of the full gradient and converges faster than Stochastic Gradient Descent (SGD).

B LSH Partition Function

Theorem B.1. *Assume there is a set of states Y. Each state y occurs with probability $[p_1 \dots p_N]$. For some feature vector x, function $f(y_i) = e^{\theta_{y_i} \cdot x}$ Then, there is a random variable whose expected value is the partition function.*

$$Est = \sum_{i=1}^{N} 1_{[y_i \in S]} \cdot \frac{f(y_i)}{p_i}$$

$$\begin{aligned}
\mathbb{E}[Est] &= \sum_{i=1}^{N} \mathbb{E}[\mathbf{1}_{[y_i \in S]}] \cdot \frac{f(y_i)}{p_i} && (6)\\
&= \sum_{i=1}^{N} p_i \cdot \frac{f(y_i)}{p_i} && (7)\\
&= \sum_{i=1}^{N} f(y_i) && (8)\\
&= Z_\theta && (9)
\end{aligned}$$

Theorem B.2. *The variance of the partition function estimator is:*

$$\begin{aligned}
Var[Est] &= \sum_{i=1}^{N} \frac{f(y_i)^2}{p_i} - \sum_{i=1}^{N} f(y_i)^2 && (10)\\
&+ \sum_{i \neq j} \frac{f(y_i)f(y_j)}{p_i p_j} \text{Cov}(\mathbf{1}_{[y_i \in S]} \cdot \mathbf{1}_{[y_j \in S]}) && (11)
\end{aligned}$$

If the states are selected independently, then we can write the variance as:

$$Var[Est] = \sum_{i=1}^{N} \frac{f(y_i)^2}{p_i} - \sum_{i=1}^{N} f(y_i)^2$$

Proof. The expression for the variance of the partition function estimator is:

$$Var[Est] = \mathbb{E}[Est^2] - \mathbb{E}[Est]^2$$

$$\begin{aligned}
Est^2 &= \sum_{ij} 1_{[y_i \in S]} 1_{[y_j \in S]} \frac{f(y_i)f(y_j)}{p_i p_j} && (12)\\
&= \sum_{i} 1_{[y_i \in S]} \frac{f(y_i)^2}{p_i^2} + \sum_{i \neq j} 1_{[y_i \in S]} 1_{[y_j \in S]} \frac{f(y_i)f(y_j)}{p_i p_j} && (13)
\end{aligned}$$

Notice.

$$\mathbb{E}[1_{[y_i \in S]} 1_{[y_j \in S]}] = \mathbb{E}[1_{[y_i \in S]}]\mathbb{E}[1_{[y_j \in S]}] + \text{Cov}[1_{[y_i \in S]} 1_{[y_j \in S]}]$$

$$\mathbb{E}[1_{[y_i \in S]}] = p_i$$

$$\mathbb{E}[Est^2] = \sum_{i=1}^{N} \frac{f(y_i)}{p_i}^2 + \sum_{i=1}^{N} f(y_i)[Z_\theta - f(y_i)] \tag{14}$$

$$+ \sum_{i \neq j} \frac{f(y_i)f(y_j)}{p_i p_j} \mathrm{Cov}(\mathbf{1}_{[y_i \in S]} \cdot \mathbf{1}_{[y_j \in S]}) \tag{15}$$

$$= \sum_{i=1}^{N} \frac{f(y_i)}{p_i}^2 + Z_\theta^2 - \sum_{i=1}^{N} f(y_i)^2 \tag{16}$$

$$+ \sum_{i \neq j} \frac{f(y_i)f(y_j)}{p_i p_j} \mathrm{Cov}(\mathbf{1}_{[y_i \in S]} \cdot \mathbf{1}_{[y_j \in S]}) \tag{17}$$

Notice.

$$\sum_{i=1}^{N} f(y_i) = Z_\theta$$

$$\sum_{i \neq j}^{N} f(y_j) = \sum_{j=1}^{N} f(y_j) - f(y_i) = Z_\theta - f(y_i)$$

Notice. $\mathbb{E}[Est]^2 = Z_\theta^2$.

$$Var[Est] = \sum_{i=1}^{N} \frac{f(y_i)}{p_i}^2 - \sum_{i=1}^{N} f(y_i)^2 \tag{18}$$

$$+ \sum_{i \neq j} \frac{f(y_i)f(y_j)}{p_i p_j} \mathrm{Cov}(\mathbf{1}_{[y_i \in S]} \cdot \mathbf{1}_{[y_j \in S]}) \tag{19}$$

□

Theorem B.3. *For any two states y_1 and y_2:*

$$P(y_1|x;\theta) \geq P(y_2|x;\theta) \iff p_1 \geq p_2$$

where

$$p_i = 1 - (1 - \mathcal{M}(\theta_{y_i} \cdot x)^K)^L$$

$$P(y|x,\theta) \propto e^{\theta_y \cdot x}$$

Proof. Follows immediately from monotonicity of e^x and $1 - (1 - \mathcal{M}(x)^K)^L$ with respect to the feature vector x. Thus, the target and the sample distributions have the same ranking for all the states under the probability. □

C LSH Interaction with Training

During training, the feature vectors of the dataset are altered through gradient descent. However, a single gradient step does not perturb the features vectors significantly. Therefore, we periodically rebuild the hash tables to minimize the computational overhead. Also, substantial changes to the feature vectors occur at the beginning of training. As training progresses, the changes to the feature vectors diminish. We take advantage of this property by annealing the rate in which we rebuild the hash tables. Also, LSH is embarrassingly parallel, so we generate the hash fingerprints using GPUs and construct the hash tables in parallel with multiple threads.

D Estimating Mutual Information with Importance Sampling

We modified the proof for InfoNCE [Oord *et al.*, 2018b] for our importance sampling MI estimator. Assume there is a set of states $Y = [y_1 \ldots y_N]$. Let $x, y_i \in Y$ be the input and output variables respectively. The optimal value for $f(y,x)$ is $\frac{p(y|x)}{p(y)}$ where f is a neural network function. For each positive example, we sample from our proposal distribution the negative examples Y_{neg}. Let $N-1$ be the total number of states in the proposal distribution excluding the positive example.

$$\mathbf{L}_{\text{IS}}^{\text{optimal}} = -\mathbb{E}_Y \log\left[\frac{f(y_i,x)}{f(y_i,x) + \sum_{y_j \in Y_{\text{neg}}} \frac{f(y_j,x)}{p_j}}\right] \tag{20}$$

$$= \mathbb{E}_Y \log\left[1 + f(y_i,x) \sum_{y_j \in Y_{\text{neg}}} \frac{f(y_j,x)}{p_j}\right] \tag{21}$$

$$= \mathbb{E}_Y \log\left[1 + f(y_i,x) \sum_{j=1}^{N-1} \mathbf{1}_{[y_j \in S]} \frac{f(y_j,x)}{p_j}\right] \tag{22}$$

$$\approx \mathbb{E}_Y \log\left[1 + f(y_i,x)\mathbb{E}\big[\sum_{j=1}^{N-1} \mathbf{1}_{[y_j \in S]} \frac{f(y_j,x)}{p_j}\big]\right] \tag{23}$$

$$= \mathbb{E}_Y \log\left[1 + f(y_i,x) \sum_{j=1}^{N-1} p_j \frac{\mathbb{E}[f(y_j,x)]}{p_j}\right] \tag{24}$$

$$= \mathbb{E}_Y \log\left[1 + f(y_i,x) \sum_{j=1}^{N-1} \mathbb{E}[f(y_j,x)]\right] \tag{25}$$

$$= \mathbb{E}_Y \log\left[1 + \frac{p(y_i|x)}{p(y_i)}(N-1)\mathbb{E}_{y_j} \frac{p(y_j|x)}{p(y_j)}\right] \tag{26}$$

$$= \mathbb{E}_Y \log\left[1 + \frac{p(y_i|x)}{p(y_i)}(N-1)\right] \tag{27}$$

$$\geq \mathbb{E}_Y \log\left[\frac{p(y_i|x)}{p(y_i)}N\right] \tag{28}$$

$$= -\mathbf{I}(y,x) + \log(N) \tag{29}$$

$\mathbf{L}_{\text{IS}}^{\text{optimal}} = -I(y,x) + \log(N)$ where N is the number of elements in the proposal distribution. For InfoNCE, the mutual information estimate is bound by the batch size M — $\mathbf{L}_{\text{NCE}}^{\text{optimal}} = -I(y,x) + \log(M)$. By using an efficient proposal distribution such as our LSH data structure, $N \gg M$. Therefore, our importance sampling approach reduces the bias in the MI estimator by effectively scaling the batch size to the total size of the proposal distribution.

Acknowledgements

Ryan Spring and Anshumali Shrivastava were supported by NSF-1652131, NSF-BIGDATA 1838177, AFOSR-YIP FA9550-18-1-0152, and ONR BRC grant for Randomized Numerical Linear Algebra.

References

[Andoni and Indyk, 2004] Alexandr Andoni and Piotr Indyk. E2lsh: Exact euclidean locality sensitive hashing. Technical report, MIT, 2004.

[Bachman *et al.*, 2019] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. In *Advances in Neural Information Processing Systems*, pages 15509–15519, 2019.

[Bell and Sejnowski, 1995] Anthony J Bell and Terrence J Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural computation*, 7(6):1129–1159, 1995.

[Charikar and Siminelakis, 2017] Moses Charikar and Paris Siminelakis. Hashing-based-estimators for kernel density in high dimensions. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1032–1043. IEEE, 2017.

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

[Chen *et al.*, 2019] Beidi Chen, Yingchen Xu, and Anshumali Shrivastava. Fast and accurate stochastic gradient estimation. In *Advances in Neural Information Processing Systems*, pages 12339–12349, 2019.

[Coleman and Shrivastava, 2019] Benjamin Coleman and Anshumali Shrivastava. Sub-linear race sketches for approximate kernel density estimation on streaming data. *arXiv preprint arXiv:1912.02283*, 2019.

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

[Gionis *et al.*, 1999] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. *VLDB*, 99(6):518–529, 1999.

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

[He *et al.*, 2012] Junfeng He, Sanjiv Kumar, and Shih-fu Chang. On the difficulty of nearest neighbor search. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1127–1134, 2012.

[Hjelm *et al.*, 2018] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[Indyk and Motwani, 1998] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, Dallas, TX, 1998.

[Leskovec *et al.*, 2014] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[Linsker, 1988] Ralph Linsker. Self-organization in a perceptual network. *Computer*, 21(3):105–117, 1988.

[Liu *et al.*, 2015] Qiang Liu, Jian Peng, Alexander Ihler, and John Fisher III. Estimating the partition function by discriminance sampling. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 514–522. AUAI Press, 2015.

[Luo and Shrivastava, 2018] Chen Luo and Anshumali Shrivastava. Arrays of (locality-sensitive) count estimators (ace) anomaly detection on the edge. In *Proceedings of the 2018 World Wide Web Conference*, pages 1439–1448, 2018.

[Luo and Shrivastava, 2019] Chen Luo and Anshumali Shrivastava. Scaling-up split-merge mcmc with locality sensitive sampling (lss). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4464–4471, 2019.

[Mussmann *et al.*, 2017] Stephen Mussmann, Daniel Levy, and Stefano Ermon. Fast amortized inference and learning in log-linear models with randomly perturbed nearest neighbor search. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2017.

[Nguyen *et al.*, 2010] XuanLong Nguyen, Martin J Wainwright, and Michael I Jordan. Estimating divergence functionals and the likelihood ratio by convex risk minimization. *IEEE Transactions on Information Theory*, 56(11):5847–5861, 2010.

[Oord *et al.*, 2018a] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[Oord *et al.*, 2018b] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[Poole *et al.*, 2019] Ben Poole, Sherjil Ozair, Aaron Van Den Oord, Alex Alemi, and George Tucker. On variational bounds of mutual information. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5171–5180, 2019.

[Spring and Shrivastava, 2017a] Ryan Spring and Anshumali Shrivastava. A new unbiased and efficient class of lsh-based samplers and estimators for partition function computation in log-linear models. *arXiv preprint arXiv:1703.05160*, 2017.

[Spring and Shrivastava, 2017b] Ryan Spring and Anshumali Shrivastava. Scalable and sustainable deep learning via randomized hashing. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 445–454, 2017.