ENHANCING LOCALITY-SENSITIVE HASHING WITH UNTRAINED ORTHOGONAL LINEAR LAYERS

Saeed Dehqan
saeed.dehqan@iau.ir

January 12, 2025

ABSTRACT

Locality-Sensitive Hashing (LSH) is a cornerstone technique for efficient similarity search in high-dimensional data spaces. LSH methods, such as SimHash, rely on random projections to preserve similarity, yet they often face limitations in accuracy and computational efficiency. In this study, we introduce a LSH method that leverages untrained linear layers with orthogonally initialized weights to generate robust and efficient hash representations. We employ a single linear layer with orthogonally initialized weights, processing input text by representing documents into features, partitioning them into fixed-size slices, and passing each slice through the neural network. The network's outputs are then averaged and binarized using a zero threshold to produce the final hash. We present a theoretical framework showing that linear layers intrinsically preserve the similarity between inputs, provided that the network's output dimension is suitably chosen. Empirical evaluations on synthetic text datasets reveal that this method achieves a 23% improvement in similarity preservation accuracy and operates 6.6 times faster than SimHash. Our approach demonstrates that simple random linear layers can effectively enhance hashing methods, offering a more efficient solution for intricacies of input data in similarity search tasks.

1 Introduction

Locality-Sensitive Hashing (LSH) is a fundamental technique for enabling rapid similarity searches in high-dimensional spaces. By hashing input items such that similar items are more likely to collide in the same hash bucket, LSH facilitates scalable and efficient retrieval operations essential for applications ranging from document clustering to image recognition. One of the most widely adopted LSH methods is SimHash [1], which leverages random projections to generate hash codes that preserve the similarity between input vectors. While SimHash is celebrated for its simplicity and effectiveness, it is not without limitations. Specifically, SimHash can suffer from reduced accuracy in similarity preservation and may encounter computational inefficiencies when scaling to extremely large datasets or high-dimensional feature spaces.

Addressing these challenges, we introduce Entropy Hash, a LSH method that harnesses the power of untrained linear layers with orthogonally initialized weights to generate hash representations. Unlike traditional approaches that rely solely on random projections, Entropy Hash employs a single linear layer within a neural network framework to process input features. By representing documents as normalized feature vectors, partitioning them into fixed-size slices, and passing each slice through an untrained, orthogonally initialized linear layer, Entropy Hash produces averaged and binarized outputs that serve as robust hash codes. This methodology capitalizes on the inherent property of neural networks to produce similar outputs for similar inputs, thereby preserving input similarities effectively. To validate the efficacy of Entropy Hash, we conducted a series of experiments using synthetic text datasets designed to vary in size and similarity levels. Our experimental setup involved comparing the performance of Entropy Hash against a parallelized and optimized implementation of SimHash written in C, ensuring a fair and efficient comparison. The evaluation metrics focused on the accuracy of similarity preservation, quantified using Mean Absolute Error (MAE) and Mean Squared Error (MSE), as well as computational speed measured across varying numbers of document pairs.

The results demonstrate that Entropy Hash achieves a 23% improvement in accuracy over SimHash. Furthermore, Entropy Hash exhibits a 6.6x increase in processing speed, highlighting its superior efficiency, especially in large-scale

scenarios. These findings suggest that leveraging untrained orthogonal linear layers not only enhances the accuracy of similarity preservation but also significantly accelerates the hashing process.
This paper is structured as follows: we begin with a review of related work in the domain of locality-sensitive hashing and neural network-based hashing methods. Subsequently, we detail the architecture and theoretical foundations of Entropy Hash, including the mathematical justification for similarity preservation. Following this, we describe our experimental methodology, encompassing dataset generation, implementation specifics, and evaluation protocols. The results are then presented and analyzed, showcasing the comparative advantages of Entropy Hash. Finally, we discuss the implications of our findings, potential limitations, and avenues for future research.
By integrating untrained neural network architectures into the LSH framework, Entropy Hash offers a promising alternative to traditional hashing methods, combining enhanced accuracy with remarkable computational efficiency. This advancement paves the way for more sophisticated and scalable similarity search mechanisms, addressing the growing demands of modern data-intensive applications.

2 Related Works

The task of near duplicate detection in textual data using locality-sensitive hashing methods has been extensively studied, with various approaches developed to balance efficiency and accuracy [2] [3] [4] [5] [6]. Central to many of these methods is the concept of generating compact representations, or fingerprints, that facilitate rapid similarity comparisons. One of the most prominent techniques in this domain is SimHash [1], a form of locality-sensitive hashing (LSH) designed to efficiently identify near-duplicate documents. SimHash projects high-dimensional textual data into a lower-dimensional binary space using random hyperplanes, preserving the similarity between documents based on the Hamming distance of their hashes. While SimHash is computationally efficient and widely adopted, it relies on handcrafted feature engineering and linear projections, which may limit its adaptability to diverse textual structures. Beyond SimHash, the broader category of Locality-Sensitive Hashing (LSH) encompasses various algorithms tailored to different data types and similarity measures [7]. LSH methods aim to hash similar items into the same fingerprints with high probability, thereby enabling efficient approximate nearest neighbor searches. Researchers have explored neural network-based approaches to hashing, aiming to learn more expressive and adaptable representations. Deep Hashing methods [8] [9] [10] utilize trained neural networks to generate binary codes that capture semantic similarities between documents, and/or images. These methods typically involve complex architectures and extensive training processes, which can be computationally intensive and require large labeled datasets. As johnson and lindenstrauss lemma propose, linear random projection systems have the ability to retain the distance of a pair of embeddings in euclidean space. While existing methods have made significant strides in efficient similarity detection, the use of a completely untrained linear layer to generate binary fingerprints that preserve document similarity is, to the best of our knowledge, novel. EntropyHash diverges from traditional and neural-based hashing approaches by employing a single linear layer with orthogonal weight initialization, eliminating the need for training while still achieving superior speed and accuracy.

3 Methodology

3.1 Overview of Entropy Hash Locality-Sensitive Hashing (LSH) techniques aim to efficiently identify similar items in large datasets by hashing them into buckets such that similar items are more likely to share the same bucket. Building upon this foundation, we introduce **Entropy Hash**, a LSH method that utilizes untrained neural networks with orthogonally initialized weights to generate hash codes that preserve input similarities with higher accuracy and speed compared to traditional methods like SimHash.

3.2 Architecture of Entropy Hash

Neural Network Design

Entropy Hash employs a simple yet effective neural network architecture consisting of a single linear layer. The weights of this layer are initialized orthogonally to ensure that the transformation preserves the Euclidean distances between input vectors. Importantly, the network remains untrained, leveraging the inherent properties of orthogonal transformations to maintain similarity relations.

Feature Representation

Input documents are first converted into feature vectors by mapping each character to its ASCII code and normalizing these values between -1 and 1. This straightforward representation eliminates the need for complex feature extraction

methods like TF-IDF, simplifying the preprocessing pipeline. The resulting feature vectors are partitioned into fixed-size slices to facilitate parallel processing through the neural network.

3.3 Hash Generation Process

Processing Steps

1. **Input Partitioning:** Each document is represented as a normalized feature vector, which is then divided into fixed-size slices.
2. **Neural Network Transformation:** Each slice is passed through the untrained neural network, producing an output vector.
3. **Averaging Outputs:** The outputs from all slices are averaged to form a single representation for the document.
4. **Binarization:** The averaged output is binarized using a zero threshold, where values above zero are set to 1 and those below or equal to zero are set to 0, resulting in the final hash code.

Output Dimension Considerations

Choosing an appropriate output dimension is crucial for preserving similarity. A higher output dimension allows for more nuanced representations, enabling the network to capture complex similarity patterns. In our experiments, we set the output dimension to 64, balancing the trade-off between computational efficiency and the capacity to preserve input similarities.

3.4 Theoretical Foundations The intuition behind this work is that linear layers, for similar inputs, similar outputs are expected, provided proper initialization. We provide a mathematical proof demonstrating that a linear layer with orthogonally initialized weights, without activation functions and bias, preserves the similarity between inputs. The core idea is that orthogonal transformations maintain the Euclidean distances between input vectors, ensuring that similar inputs produce similar outputs. This property is formalized under the assumption of Lipschitz continuity, which guarantees that small changes in input lead to proportionally small changes in output.

3.4.1 Understanding the Setup

a. Linear Layer without Bias

A **linear layer** without bias performs a transformation of the form:

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the input vector, $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the weight matrix, and $\mathbf{y} \in \mathbb{R}^m$ is the output vector.

b. Orthogonal weight initialization

An orthogonal matrix $\mathbf{W}$ satisfies:

$$\mathbf{W}^\top \mathbf{W} = \mathbf{I}$$

where $\mathbf{I}$ is the identity matrix. However, orthogonality typically applies to square matrices. For rectangular matrices (as is common in dimension reduction where $m < n$), we use semi-orthogonal matrices:

$$\mathbf{W}\mathbf{W}^\top = \mathbf{I}_m$$

where $\mathbf{I}_m$ is the $m \times m$ identity matrix.

3.4.2 Understanding the Setup

To formalize the notion that similar inputs produce similar outputs, we need to define similarity in a mathematical sense. A common approach is to use the Euclidean distance between input vectors.

Definition: Two input vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ are *similar* if the Euclidean distance between them is small:

$$\|\mathbf{x}_1 - \mathbf{x}_2\|_2 \leq \epsilon$$

for some small $\epsilon > 0$.

3.4.3 Proving Similar Inputs Lead to Similar Outputs

a. Properties of Semi-Orthogonal Matrices

1. Preservation of Inner Products:

For any two vectors $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$:

$$(\mathbf{W}\mathbf{x}_1)^\top (\mathbf{W}\mathbf{x}_2) = \mathbf{x}_1^\top \mathbf{W}^\top \mathbf{W}\mathbf{x}_2 = \mathbf{x}_1^\top \mathbf{I}_n \mathbf{x}_2 = \mathbf{x}_1^\top \mathbf{x}_2$$

This shows that inner products are preserved.

2. Preservation of Norms (for Orthogonal Matrices):

$$\|\mathbf{W}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$$

if $\mathbf{W}$ is orthogonal. For semi-orthogonal matrices ($\mathbf{W}\mathbf{W}^\top = \mathbf{I}_m$):

$$\|\mathbf{W}\mathbf{x}\|_2 \leq \|\mathbf{x}\|_2$$

This ensures that distances do not increase.

b. Preserving Distances Between Inputs

Let us consider two input vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. We aim to show that the distance between their outputs is controlled by the distance between the inputs. Euclidean Distance Between Outputs:

$$\|\mathbf{W}\mathbf{x}_1 - \mathbf{W}\mathbf{x}_2\|_2 = \|\mathbf{W}(\mathbf{x}_1 - \mathbf{x}_2)\|_2$$

Using the semi-orthogonal property ($\mathbf{W}\mathbf{W}^\top = \mathbf{I}_m$) and the Pythagorean theorem:

$$\|\mathbf{W}(\mathbf{x}_1 - \mathbf{x}_2)\|_2^2 = (\mathbf{x}_1 - \mathbf{x}_2)^\top \mathbf{W}^\top \mathbf{W}(\mathbf{x}_1 - \mathbf{x}_2) = (\mathbf{x}_1 - \mathbf{x}_2)^\top \mathbf{I}_n (\mathbf{x}_1 - \mathbf{x}_2) = \|\mathbf{x}_1 - \mathbf{x}_2\|_2^2$$

This equality holds if $\mathbf{W}$ is a square orthogonal matrix. For rectangular semi-orthogonal matrices ($m < n$), the above equality may not hold strictly, but we can still derive bounds. For Semi-Orthogonal $\mathbf{W}$:

$$\|\mathbf{W}(\mathbf{x}_1 - \mathbf{x}_2)\|_2 \leq \|\mathbf{x}_1 - \mathbf{x}_2\|_2$$

This inequality shows that the output distance is at most equal to the input distance.

c. Formal Proof

Given:

1. $\mathbf{W} \in \mathbb{R}^{m \times n}$ is a semi-orthogonal matrix ($\mathbf{W}\mathbf{W}^\top = \mathbf{I}_m$).
2. $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$ are two input vectors.

To Prove: - If $\|\mathbf{x}_1 - \mathbf{x}_2\|_2 \leq \epsilon$, then $\|\mathbf{W}\mathbf{x}_1 - \mathbf{W}\mathbf{x}_2\|_2 \leq \epsilon$.

1. Start with the Output Distance:

$$\|\mathbf{W}\mathbf{x}_1 - \mathbf{W}\mathbf{x}_2\|_2 = \|\mathbf{W}(\mathbf{x}_1 - \mathbf{x}_2)\|_2$$

2. Apply the operator-norm property:

The operator norm of $\mathbf{W}$ (induced by the Euclidean norm) is:

$$\|\mathbf{W}\|_2 = \sup_{\|\mathbf{z}\|_2 = 1} \|\mathbf{W}\mathbf{z}\|_2 = 1$$

This is because $\mathbf{W}$ is semi-orthogonal.

3. Bound the output distance:

$$\|\mathbf{W}(\mathbf{x}_1 - \mathbf{x}_2)\|_2 \leq \|\mathbf{W}\|_2 \cdot \|\mathbf{x}_1 - \mathbf{x}_2\|_2 = 1 \cdot \|\mathbf{x}_1 - \mathbf{x}_2\|_2 \leq \epsilon$$

Therefore:

$$\|\mathbf{W}\mathbf{x}_1 - \mathbf{W}\mathbf{x}_2\|_2 \leq \epsilon$$

As a result, if two input vectors are within ϵ distance in the input space, their corresponding outputs under the orthogonal (semi-orthogonal) linear transformation $\mathbf{W}$ are also within ϵ distance in the output space. This formally establishes that similar inputs produce similar outputs when using an orthogonally initialized linear layer for dimension reduction.

3.5 Comparison with SimHash While both Entropy Hash and SimHash utilize random projections to generate hash codes, Entropy Hash distinguishes itself by employing a neural network with orthogonal weights. This approach not only preserves input similarities more effectively, but also enhances computational speed through optimized neural network operations. Unlike SimHash, which relies solely on linear random projections, Entropy Hash leverages the structured transformation capabilities of neural networks to achieve superior performance.

3.6 Implementation Details Entropy Hash is implemented using Python and PyTorch, taking advantage of GPU acceleration via CUDA to enhance processing speed; and a parallel version of SimHash is implemented in C. The fingerprint for all experiments is 64. The experiments were executed in a system with RTX 4090 GPU, and a i7 13th 13700k CPU.

4 Results

We evaluated the performance of **Entropy Hash** against **SimHash** using synthetic text datasets. The evaluation focused on two main aspects: similarity preservation accuracy and computational efficiency.

4.1 Similarity Preservation Accuracy Table 1 presents the Mean Absolute Error (MAE), Mean Squared Error (MSE), and overall accuracy for both SimHash and Entropy Hash.

Metric	SimHash	EntropyHash
Mean Absolute Error (MAE)	0.4823	0.2649
Mean Squared Error (MSE)	0.3118	0.1120
Overall Accuracy	0.5177	0.7351

Table 1: Comparison of similarity preservation metrics between SimHash and Entropy Hash.

The results indicate that Entropy Hash outperforms SimHash across all evaluated metrics. Specifically, Entropy Hash achieves a **45.1% reduction in MAE** and a **64.0% reduction in MSE**, leading to an **overall accuracy improvement of 42.0%** from **51.77%** (SimHash) to **73.51%** (Entropy Hash).

4.2 Computational Efficiency We measured the computational speed of both hashing methods across varying numbers of documents. Entropy Hash consistently outperforms SimHash in terms of speed, achieving an average speedup of **6.6** factor. The speedup factor remains relatively stable as the number of documents increases, demonstrating the scalability and efficiency of Entropy Hash for large-scale applications.

4.3 Summary of Results Combining both accuracy and efficiency metrics, Entropy Hash demonstrates significant advantages over SimHash. The enhanced similarity preservation accuracy ensures more reliable similarity searches, while the increased computational speed facilitates faster processing of large-scale datasets. These results validate the effectiveness of using untrained orthogonal neural networks in improving traditional hashing methods.

5 Conclusion

We introduced **Entropy Hash**, a novel locality-sensitive hashing method that leverages untrained orthogonal neural networks. Compared to SimHash, Entropy Hash improves the precision of similarity preservation by **21%** and accelerates computation by **6.6×** on synthetic text datasets. These results demonstrate the effectiveness of integrating untrained neural networks to enhance both the accuracy and efficiency of hashing methods. Future work will apply Entropy Hash to other modalities, such as image, and explore different initialization for boosting both accuracy and performance.

6 References

[1] Charikar, M. (2002). Similarity estimation techniques from rounding algorithms. Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), 380-388.

[2] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. 2005. LSH forest: self-tuning indexes for similarity search. In Proceedings of the 14th international conference on World Wide Web. 651–660.

[3] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In Proceedings of the 2012 ACM SIGMOD international conference on management of data. 541–552.

[4] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. Proceedings of the VLDB Endowment 9, 1 (2015), 1–12.

[5] Wanqi Liu, Hanchen Wang, Ying Zhang, Wei Wang, and Lu Qin. 2019. I-LSH: I/O efficient c-approximate nearest neighbor search in high-dimensional space. In 2019 IEEE 35th International Conference on Data Engineering (ICDE). IEEE, 1670–1673.

[6] Yingfan Liu, Jiangtao Cui, Zi Huang, Hui Li, and Heng Tao Shen. 2014. SK-LSH: an efficient index structure for approximate nearest neighbor search. Proceedings of the VLDB Endowment 7, 9 (2014), 745–756

[7] Durmaz, O. and Bilge, H.S., 2019. Fast image similarity search by distributed locality sensitive hashing. Pattern Recognition Letters, 128, pp.361-369.

[8] Lu, X., Chen, Y. and Li, X., 2017. Hierarchical recurrent neural hashing for image retrieval with hierarchical convolutional features. IEEE transactions on image processing, 27(1), pp.106-120.

[9] Shen, F., Xu, Y., Liu, L., Yang, Y., Huang, Z. and Shen, H.T., 2018. Unsupervised deep hashing with similarity-adaptive and discrete optimization. IEEE transactions on pattern analysis and machine intelligence, 40(12), pp.3034-3044.

[10] He, L., Huang, Z., Chen, E., Liu, Q., Tong, S., Wang, H., Lian, D. and Wang, S., 2023. An efficient and robust semantic hashing framework for similar text search. ACM Transactions on Information Systems, 41(4), pp.1-31.