On minimizers and convolutional filters: a partial justification for the unreasonable effectiveness of CNNs in categorical sequence analysis

Yun William Yu

1 Department of Mathematics, University of Toronto, Ontario, M5S 1A1, Canada
2 Computer and Mathematical Sciences, University of Toronto at Scarborough, Toronto, Ontario, M1C 1A4, Canada
ywyu@math.toronto.edu

Abstract. Minimizers and convolutional neural networks (CNNs) are two quite distinct popular techniques that have both been employed to analyze biological sequences. At face value, the methods seem entirely dissimilar. Minimizers use min-wise hashing on a rolling window to extract a single important k-mer feature per window. CNNs start with a wide array of randomly initialized convolutional filters, paired with a pooling operation, and then multiple additional neural layers to learn both the filters themselves and how those filters can be used to classify the sequence. In this manuscript, I demonstrate through a careful mathematical analysis of hash function properties that there are deep theoretical connections between minimizers and convolutional filters—in short, for sequences over a categorical alphabet, random Gaussian initialization of convolutional filters with max-pooling is equivalent to choosing minimizers from a random hash function biased towards more distinct k-mers. This provides a partial explanation for the unreasonable effectiveness of CNNs in categorical sequence analysis.

Keywords: Minimizers · CNNs · Hashing

1 Introduction

It is a pithy statement now that the near exponential explosion of biological sequence data we confront requires the construction of more efficient tailored algorithms capable of handling the data deluge [28,4]. Over the last decade, several tools have emerged from the algorithmic literature as instrumental for biological sequence analysis, whether in reducing time- or space-complexity. For example, for read-alignment/assembly, compressive data structures that rely on the inherent redundancy of the underlying genome(s), such as the FM-index [15], entropy-scaling search trees [49], and more recently the r-index [20] have been widely used [43,48,30]. Alternately, probabilistic sketching methods such as MinHash [4], HyperLogLog [17], and HyperMinHash [50] have found great applicability in fast genome comparison [37,3]. Rather than understanding the entropic structure of the underlying data, these methods instead use random hash functions to force the data into a shape with well-characterized approximation error. And finally, local k-mer subsampling schemes such as minimizers [41,39], syniners [14], and minimally-overlapping words [18] reduce the redundancy found in analyzing neighboring k-mers in a sequence which overlap almost completely, and thus allow speeding up tasks like taxonomic classification [47], read mapping, or assembly [32]. These methods have all become necessary because although Moore’s law on transistor density has continued unabated, the same cannot be said about single-threaded processing times or fast memory access.

However, it bears remarking that even as much of more traditional genomics analysis has turned to increasingly efficient algorithmics, the burgeoning subarea of deep learning in biology has exploded [116], and the neural networks being used for the same tasks (such as metagenomic classification) are by comparison seemingly unconstrained by the need for efficiency. Instead, deep learning is able to continue to tap into Moore’s law because their computational primitive of numerical linear algebra...
is easily vectorized onto hardware accelerators such as GPUs (graphics processing units) and TPUs (tensor processing units) [35,27]. On broader more traditional machine learning tasks such as classification and prediction of images those models often face little competition from more traditional methods [11,34]. Furthermore, deep learning methods are often able to achieve comparable accuracy even on core computational biology tasks like variant calling [38], but to do so require extremely large amounts of computation for both training and inference.

Still, although there are theoretical results showing that with unbounded compute, neural networks can learn any function [22], and hardware accelerators are able to leverage much larger amounts of raw processing power than more conventional algorithmic approaches, a lot of work has been done to create more efficient architectures, including convolutional neural nets (CNNs) [19], recurrent neural nets (RNNs) [40], long-short-term memory structures (LSTM) [21], Transformer networks [45], and more. Indeed, much of the business of deep learning is in applying putatively more efficient architectures to novel tasks and datasets. Sometimes, neural networks have a biological justification—and indeed, they were originally inspired by simplified models of biological neurons; for example, CNNs were based on the mammalian visual cortex [23]. However, although some ML models have an intuitive justification, often, they are simply found to perform well empirically, and it is difficult to interpret why it works, or what the learned weights might mean biologically [36].

While interpretations of a CNNs weights and neurons are often made by way of comparison to the receptive fields of visual neurons, that interpretation of CNNs falls apart when applied to biological sequence data. Unlike the brightness of a pixel in an image, biological sequences are not ordinal, but rather instead categorical vectors—a nucleotide is either present or not in a particular position, without any sense of relative magnitude. In images, for example, a filter may detect a horizontal edge, which is characterized by bright pixels on one side and dark on the other; if you slightly brighten the pixels on one side, or add some noise to the image, the edge is still characterized by that pattern, which filters are trained to pick up. However, while you could imagine patterns of nucleotides that correspond to an edge, e.g. AAAATTTT, there are no intermediate ‘levels’ for a CNN filter to pick up. Still, CNNs have proved effective on genomic data, such as in the classification of metagenomic sequences of bacteria by taxonomy [16]. It thus seems that CNNs are not intuitively suited for categorical sequence classification, but empirically, they irrespectively have excellent performance. Why?

In this manuscript, I prove the technical Theorem showing that the hash family of dot products with spherical Gaussian multivariates preferentially selects more extremal elements as maximums. A immediate corollary is then our Main Theorem showing that on categorical sequences, minimizers and convolutional filters with max-pooling are nearly equivalent, resolving our central question by connecting together two seemingly unrelated methods that have both achieved powerful results in computational biology.

**Theorem 1.** Consider a CNN acting on one-hot encoded categorical vectors with a single convolutional filter following by max-pooling, both of stride-1. Then

1. the output of the max-pooling layer is precisely equivalent to choosing a minimizer over the windows/patches that are max-pooled;
2. And if the weights of the filter are initialized as a spherical Gaussian multivariate, then the implied minimizer hash function is at least as likely to choose the the most distinct dense categorical feature in a window as any other k-mer, where distinctness is measured in absolute Hamming deviation from the other features.

Note that in the 1D case prevalent in genomic sequences, dense categorical features correspond precisely to k-mers. By using the descriptor ‘dense,’ we are restricting our universe to features where every position is specified, rather than ones with wildcards, gaps, or ignored regions; however, it is worth mentioning that selecting gapped k-mers can be done by zero-ing out some of the entries of the Gaussian multivariate.
2 Building blocks

We begin with a review of hashing, permutations, MinHash, minimizers, and CNNs. Although all of our analyses generalize naturally to higher dimensions, the notation becomes quite confusing, so without loss of generality we will focus on the 1D case. Three key intuitions will be needed:

1. For minimizers and MinHash, the hash functions need only impose an ordering on the domain, and as such need not follow exactly classical hash function constructions.
2. In contrast to MinHash, minimizers do not require min-wise independence of their hash functions.
3. A random convolutional filter followed by max-pooling precisely orders the space of k-mers and chooses an extreme element.

2.1 Hashing and permutations

The key to our analysis will be in understanding the role of hashing [8] and permutations. We recall some basic concepts here. For reference, see any standard text or this gentle introduction to fast modern hashing methods by Thorup [44].

**Definition 1.** A hash function \( h : U \to M \) maps elements from some universe \( U \) to some range \( M \).

**Definition 2.** A random hash function \( h : U \to M \) is a function that is uniformly drawn from a family \( H \) of hash functions.

To illustrate, let’s start with a classical example. Let \( F_p \) be the finite field with \( p \) elements, for some prime \( p \). Then we can define a hash function \( h_{a,b} : F_p \to F_p \) given by \( h_{a,b}(x) = ax + b \mod p \). The hash family here is parameterized by \( a,b \in F_p \), and so our analysis can operate on probabilities and expectations that arise from treating \( a,b \), and therefore \( h_{a,b} \), as random variables.

Another more sophisticated modern example is the vector multiply-shift family of hash functions, introduced by Dietzfelbinger [2]. We introduce it here because the construction is mathematically equivalent to a dot product with uniform random integers in the integer ring, followed by an integer division/bit-shift operation. Let \( U = [2^w]^d \) and \( M = [2^l] \), where \([a] \) is the integer ring \([0,1,\ldots,o-1]\). We will need \( w \geq l + t - 1 \), and then we pick a uniform random vector \( a \in [2^w]^d \), as well as a uniform random element \( b \in [2^w] \). We define a hash function \( h_{a,b} : U \to M \) given by \( h_{a,b}(x) = (x \cdot a + b) \mod 2^{w-l} \), where the dot product and addition are in the ring \([2^w]\), and \( \mod \) is ordinary integer division without remainder (i.e. a bit-shift to the right by \( w-l \) bits), leaving an answer that is just \( l \) bits. In practice, using fast 64-bit integer arithmetic, we can let \( l = 32 \), \( w = 32 \), and \( w = 64 \), allowing us to hash vectors of unsigned 32-bit integers by doing a dot product with a random 64-bit integer vector, adding another random 64-bit integer, and then taking the higher 32-bits.

Depending on the application at hand, different properties of a hash function may be desirable, such as universality [8], strong universality [40], or min-wise independence [6]. Both of the examples given above exhibit strong universality (also known as 2-independence).

**Definition 3.** \( \mathcal{H} \) is a strongly universal (or 2-wise independent) hash family if \( \forall i_1 \neq i_2 \in U \), and \( \forall j_1, j_2 \in M \),

\[
\Pr_{h \sim \mathcal{H}}(h(i_1) = j_1 \land h(i_2) = j_2) = \frac{1}{|M|^2}
\]

Roughly speaking, any two hash values can be construed as independent uniform random variables on the range.

However, in this work, we are primarily interested in permutations. Nontrivial permutations are expensive to encode, but they can be approximated via random hashing. This is the basis of the celebrated MinHash algorithm for computing set similarity [3]. Note that for any set \( S \subset U \) with no collisions under a hash function \( h \)—i.e. \( h(s_i) \neq h(s_j) \) for any \( s_i \neq s_j \in S \)—the hash function defines an ordering/ permutation on \( S \). One property of random permutations people have sought to capture is that every item in \( S \) has an equal chance of being the smallest, which we can formalize.
Definition 4. A family of hash functions $\mathcal{H}$ is min-wise independent if for any set $S \subseteq U$, and any $s \in S$,

$$\Pr_{h \sim H}(\min_{S} \{h(S)\} = h(s)) = \frac{1}{|S|} \quad (2)$$

Min-wise independence unfortunately does not follow as a consequence of strong universality, though you can approximate it with sufficiently high degrees of k-independence [24] or by using a twisted variant of tabulation hashing [10].

Alternately, many real-world implementations eschew the formal guarantees of random hashing, and instead just use a deterministic hash function—such as MurmurHash3 [2], the SHA family of cryptographic hash functions [13], or efficient canonical choices of group generators in prime fields. This is very common in bioinformatics software making use of minimizers. However, such constructions obscure the connections we will be drawing in this paper.

2.2 Min-wise hashing

Let’s begin with min-wise hashing (MinHash), which can be thought of as a randomized global feature selection method over the space of k-mers in a sequence, and which is closely related to the local feature selection method of minimizers. MinHash was invented for computing the resemblance of documents as given by the Jaccard index of k-mers (or n-grams or shingles) [5], so it gives a quite strong promise about the features it selects.

Definition 5. Given two sets $A$ and $B$, the Jaccard index [25] is

$$t(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (3)$$

MinHash relies on a random permutation of the set $A \cup B$.

Lemma 1. Given a min-wise independent hash function $h$ and two sets $A, B$,

$$\Pr_{x \in A, y \in B}(\min_{x \in A} h(x) = \min_{y \in B} h(y)) = \Pr_{x \in A \cup B}(\arg \min_{x \in A \cup B} h(x) \in A \cap B) = t(A, B). \quad (4)$$

Proof. The lemma follows directly from Definition 4 because every item has in the union has equal probability of being the minimum.

A set $A$ can then be represented for computing Jaccard index by storing just the minimum item (or its hash value). The error in Jaccard index estimation can then be driven down by a factor of $\frac{1}{\sqrt{m}}$ by repeating with $m$ independent permutations and averaging. Alternately, the error can also be driven down by using a single permutation and keeping the smallest $m$ items [9]. MinHash is great for comparing entire sequences globally against each other [37], and one hope would be that we can modify MinHash to being a local feature selection method. That goal is effectively achieved through minimizers, though as a historical note, minimizers were invented independently of MinHash.

2.3 Minimizers

Minimizers [41,39] are a local k-mer selection scheme, in some ways, the classic k-mer selection scheme. The most important feature of a local k-mer selection scheme is translation invariance; we want to subsample the set of k-mers in a sequence in such a way that even if we insert or delete a letter at the beginning of the sequence, the set of minimizers does not change very much. There
are a number of more modern k-mer selection schemes\cite{14,18,42}, with slightly different properties, but minimizers were among the first used in computational biology. They are related to min-wise hashing\cite{5}, but instead of getting features for an entire sequence at once, they break up a sequence into smaller windows and get the minimum hash within each window. That minimum k-mer, a \textit{minimizer}, is used to match the sequence to some reference, e.g. for classification\cite{47} or for sequence assembly/mapping\cite{32}. Often, for longer sequences, we match the sequence to a reference only if multiple minimizers from different windows match. The compressive nature of minimizers appears because most of the time, the minimizer remains constant as the window rolls, so the total number of minimizers for a sequence is much smaller than the total number of windows.

More precisely, consider an alphabet $\Sigma$. For genomics, $\Sigma = \{A, C, G, T\}$, whereas for protein sequences $|\Sigma| = 20$ in the standard amino acid alphabet. We are interested in analyzing the set of variable-length strings $\Sigma^* = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$. Given a length-$l$ string $x = x_0 \cdots x_{l-1} \in \Sigma^l$, where each $x_i \in \Sigma$, one common analytical technique is to consider all length-$k$ substrings $\{k_0, \ldots, k_{l-k}\}$, where $k_i = x_i \cdots x_{i+k-1} \in \Sigma^k$. Given a hash function $h$, instead of constructing a MinHash sketch by taking the smallest $k$-mers overall, we instead define the minimizers of the sequence as $\{w_0, \ldots, w_{l-k-w}\}$, where

$$w_i = \arg \min_{k_j \in \{k_i, \ldots, k_{i+w-1}\}} h(k_j).$$

\textbf{Lemma 2.} Let $l > w$, and let $x \in \Sigma^l$ be any string of length $l$ without duplicate $k$-mers. Then using the notation given above, if $h$ is a min-wise independent hash function, then adjacent windows share a minimizer with probability $Pr_h(w_i = w_{i+1}) \geq \frac{w-1}{w+1}$.

\textbf{Proof.} Adjacent windows share $w - 1$ k-mers, and there are $w + 1$ unique k-mers between the two windows. Thus, the Jaccard index is precisely $\frac{w-1}{w+1}$, and the proof follows from Lemma 1.

Thus, the deduplicated set of minimizers of a sequence is much fewer than $l - k - w$. However, a key difference between minimizers and MinHash is that the hash function need not actually be min-wise independent in general. This is because we are not using minimizer-collision probabilities as an estimator for Jaccard index, but rather simply using them as a sparse sampling of the k-mer space. Minimizers only require translation invariance, which is satisfied by any arbitrary permutation, rather than needing a random permutation. Indeed, this fact has been exploited to construct non-random minimizers that are more evenly distributed, or that are likely to be rarer in a genome\cite{61,26}. Still, some amount of randomness is important; in the worst case, an adversarially chosen permutation that orders k-mers in the same order as appears in the sequence gives no amount of subsampling. Or, more realistically, simply taking a lexicographic ordering on the space of k-mers works quite badly, for example, on poly-A strings where the repetitiveness of the poly-A prefix causes many distinct k-mers to be minimizers of adjacent windows.

\subsection{CNNs}

CNNs are inspired by the visual cortex of mammals, and were notably demonstrated to be effective for image processing\cite{23,19}. CNNs are characterized by applying a set of filters in a translation-invariant noise-robust way to different parts of an image to generate a set of features, adding a pooling layer to reduce the information passed downstream, and then following up with a feed-forward neural network for analyzing the features and connecting them to a classification or prediction\cite{31}. Of course, this is a gross oversimplification, and modern architectures are much more complex, but this captures the basic idea.

For our analysis, we consider a simple 1D-CNN on a 4-letter alphabet $\Sigma$ with an initial convolutional layer with a single filter of size $k$ and stride-1, a max-pooling layer with patch-size $w$ and stride-1, followed up by an arbitrary feed-forward neural network—consider a multi-layer perceptron
for simplicity, but it is irrelevant for our analysis—and initialized with Gaussian random weights with mean 0 and variance 1. The results that follow generalize naturally to other finite alphabets, multiple filters, and multiple dimensions, but the analysis will be easier in this setup.

More formally, let \( s = [s_0, \ldots, s_{l-1}] \in \Sigma^l \) be a length-\( l \) string. Encode the string \( s \) in a one-hot encoding \( e = e(s) = [e_0, \ldots, e_{4l-1}] \in \{0,1\}^d \). Our convolutional filter \( g \) is initialized as a spherical Gaussian multivariate vector with i.i.d. components of length \( 4k \), \( g = [g_0, \ldots, g_{4k-1}] \). Because our one-hot encoding expanded the length of the vector by a factor of 4, we will be using a stride of 4 instead of 1—we choose to eschew tensor notation for the sake of simplicity. Then, the first convolutional layer is the function \( L_1 : \{0,1\}^d \to \mathbb{R}^{l-k} \) given by

\[
L_1(e)_i = (e * g)_{4i} = \sum_{j=1}^{4k} e_{4i+j}g_j \quad \text{for } i \in \{0, \ldots, l-k-1\} \quad (6)
\]

The second layer of the CNN is the max-pooling function, which will be given by \( L_2 : \mathbb{R}^{l-k} \to \mathbb{R}^{l-k-w} \) as

\[
L_2(y)_i = \max_{j \in \{0, \ldots, w-1\}} y_{i+j} \quad \text{for } i \in \{0, \ldots, l-k-w-1\}. \quad (7)
\]

After the max-pooling layer, the remainder of the neural network is trained to use the outputted features for the designated classification or prediction task.

It should be apparent that the setup for convolutional filters closely resembles that of the setup for minimizers, where the ‘hash function’ is a dot product with the weights vector. In both cases, some function is applied to k-mers in the string on a rolling basis. Then, we keep the extreme value of the output of that function, whether it is a minimum or maximum, and pass that onto further analytical pipelines. The primary question at hand is why convolutional filters work on categorical sequences, when the usual story told is about robustness to non-categorical (ordinal/interval/ratio) noise in feature selection.

### 3 Proofs

Recall that the vector-multiply-shift family of hash functions defined by dot product with a uniform random ring element, following by division without remainder, is 2-independent. Our main claim is that the family of hash functions defined by dot product with a multivariate spherical Gaussian has the property that more unique k-mers with a string are more likely to be extremal elements and thus selected as minimizers/maximizers.

**Theorem 2.** Consider a set \( S \) of \( n \) binary vectors in \( \{0,1\}^d \), all with the same number \( m \) of set bits. Define the degree \( \Delta(x) \) of \( x \in S \) by

\[
\Delta(x) = \sum_{s \in S} ||x - s||_1 \quad (8)
\]

Let \( g \) be a spherical multivariate Gaussian random variable of dimension \( d \) (each entry \( g_i \sim \mathcal{N}(0,1) \) i.i.d.), defining a hash function \( h : S \to \mathbb{R} \) by \( h(x) = x \cdot g \).

Then for all \( \Delta(x) > \Delta(y) \)

\[
\Pr \left( \left( x \cdot g = \max_{s \in S} s \cdot g \right) \geq \left( y \cdot g = \max_{s \in S} s \cdot g \right) \right). \quad (9)
\]

\footnote{Note that we follow the common machine learning convention whereby convolution indices are not reversed; apologies for any confusion to those from outside of ML, but it makes no difference because the indices of \( g \) are randomly initialized.}
Proof. Without loss of generality, let $S = \{x = s_1, y = s_2, s_3, \ldots, s_n\}$. Let the random variable $Y_i = h(s_i)$, and let $\Delta_i = \Delta(s_i)$.

Lemma 3. $\Delta_i = \mathbb{E}\sum_{j=1}^{n}(Y_i - Y_j)^2$

Proof. Each $Y_i$ is a sum of 1D Gaussians from $g$, so $Y_i - Y_j$ precisely cancels out any shared terms, leaving us with positive copies of the Gaussians only in $Y_i$, and negative copies of the Gaussians only in $Y_j$—and of course the Gaussian distribution is invariant under reflection. Since all our binary vectors had the same number of shared bits, the total number of both positive and negative Gaussians that are summed to form $Y_i - Y_j$ is precisely the Hamming distance $||s_i - s_j||_1$. The sum of $t$ independent $\mathcal{N}(0, 1)$ Gaussians is just another Gaussian with distribution $\mathcal{N}(0, t)$ (i.e. a variance $t$ Gaussian), and the second moment of a Gaussian is just its variance. Thus, $\mathbb{E}(Y_i - Y_j)^2 = ||s_i - s_j||_1$, and the lemma follows by linearity of expectation.

Interpreting Lemma 3, the degree $\Delta_i$ gives the expected squared deviation after the dot product. Arguing solely from expectations, we expect the lowest degree items to have the lowest squared deviation after the dot product, and therefore be closest to the mean. Conversely, high degrees correspond to being further away from the mean, and the set member that has the highest squared deviation from the mean has to be either the min or the max. The joint distribution is symmetric about the origin, so the probability of being the min or the max are equal. Thus, we need only show that with probability at least 0.5, $Y_1$ has a higher squared deviation than $Y_2$.

Thus, define a new random variable $Z = \sum_{j=1}^{n}(Y_i - Y_j)^2 - \sum_{j=1}^{n}(Y_2 - Y_j)^2$. When $Z > 0$, $Y_1$ is a more extreme value than $Y_2$, and vice versa when $Z < 0$. Of course, $\mathbb{E}Z = \Delta_1 - \Delta_2 > 0$, but expectation is insufficient for showing that with probability 0.5, $Z > 0$. That is thus the goal for the remainder of this proof.

As in the proof of Lemma 3, we use the fact that the $Y_i$’s arise as sums of selected Gaussians from $g = \{g_1, \ldots, g_d\}$. Let $t = ||s_i - s_j||_1$, which is even because all the bit vectors have the same number of set bits. Then $(Y_i - Y_j)$ is the sum of $t/2$ entries from $g$ minus the sum of another distinct $t/2$ entries from $g$. Then $(Y_i - Y_j)^2$ is the sum of $t$ squared Gaussians and all of the cross terms, which are either positive or negative copies of products of two independent Gaussians. Thus, we can rewrite

$$Z = \sum_{i=1}^{d}c_i g_i^2 + \sum_{i=1}^{d} \sum_{j=i+1}^{d} c_{i,j}g_i g_j,$$

where the $c_i$’s and $c_{i,j}$’s are unknown coefficients in $Z$.

First, note that $g_i g_j = \frac{1}{4}(g_i + g_j)^2 - \frac{1}{4}(g_i - g_j)^2$, the difference of two independent parameter-1 $\chi^2$ variables. Additionally, using $\mathbb{E}g_i^2 = 1$ and $\mathbb{E}g_i g_j = 0$, that implies that $\mathbb{E}\sum_{i=1}^{d}c_i g_i^2 = \Delta_1 - \Delta_2$.

The summation $\sum_{i=1}^{d}c_i g_i^2$ can be thought of as a sum of a bunch of positive and negative copies of parameter-1 $\chi^2$ variables, with more positive than negative terms. Pairing together mutually independent positive and negative $\chi^2$ variables, we can thus rewrite $Z$ as the sum of positive $\chi^2$ variables with a collection of differences of mutually independent parameter-1 $\chi^2$ variables. The difference of two mutually independent random variables of the same distribution is of course symmetric about the origin—in this case, the difference of two independent $\chi^2$ variables gives a variance-gamma distribution [29]. Thus, $Z$ is the sum of the leftover $\chi^2$ terms, which are positive, and something that is symmetric about the origin, so $\Pr(Z > 0) \geq 0.5$, concluding the proof.

Note that this proof is actually stronger than the claim in the theorem, as we have just shown that if $\Delta(x) > \Delta(y)$, with probability at least 0.5, $h(x)$ is further from the mean than $h(y)$.

We can now directly apply Theorem 2 to get Theorem 1. A one-hot encoding of a categorical vector precisely gives binary vectors with the same number of set bits. The first part of Theorem 1 follows directly from the definitions set up in Sections 2.3 and 2.4 as convolution with a randomly
initialized spherical Gaussian is clearly some kind of hash function, though perhaps not one with the standard properties, and that a max-pooling operation is equivalent to taking a minimum across windows. However, we just proved that convolution with a spherical Gaussian applies a dot-product hash function to each k-mer that preferentially selects more distinct k-mers, proving the second part of Theorem 1.

4 Discussion

Our results can be interpreted in the context of random projection theory. A dot product with a spherical Gaussian is precisely a random projection from points on a high-dimensional hypercube to the real line. Because the dimensionality is reduced all the way down to one, we cannot actually give probabilistic guarantees a la Johnson-Lindenstrauss [33], but it remains true that with higher than uniform probability, extremums in the hypercube remain extremums after projection. Thus, the family of hash functions we get preferentially selects more distinct k-mers in categorical vectors.

Luckily, as discussed earlier, minimizers and CNN features do not depend on more classical notions of uniform hashing. Indeed, this idea has been independently explored for both minimizers and CNNs. On the minimizer front, this primarily consists of modifications to the hash function for better density [51], or by postprocessing the permutation using inverse document/genome frequency [26]. For CNNs, feature construction is largely trained, instead of designed, but there is also work specifically on architectures for promoting better features [7]. Here we have shown that even just initializing with Gaussian weights causes some correlation to inverse set frequency.

Notably, clusters of similar data do still get mapped close together with a Gaussian convolution, as opposed to a random minimizer. This is a known property of CNN filters, though it is not so important when the output is used as a minimizer for exact matching. To put it in biological terms, all the k-mers matched by a spaced seed get mapped close to each other by a Gaussian convolution; irrelevant for exact matching, but it does open up possibilities for other kinds of analysis, as obviously evinced by CNN classification working well.

Of course, this highlights one of the potential ramifications of this paper: it might be possible to design or train convolutional filters that explicitly learn some minimizer hash function of interest. Indeed, this is an already known part of the effectiveness of CNNs, that they are able to learn features of interest. We envision training hash functions that are weighted towards particular biologically significant k-mers. The advantage of these trained hash functions over a lookup table of some kind is in computational efficiency of both time and space.

Finally, we discussed earlier how our theory results generalize to higher dimensions of categorical data. And of course, multiple filters corresponds to just taking multiple minimizers within each region. This corresponds roughly to generating a full MinHash-style sketch of each window (though it does not work for Jaccard similarity because Gaussian convolutions are not min-wise independent), capturing more information than a single minimizer does. Another reason why this may be helpful for CNNs is due to the fact that memorizing all the output values of a single neuron is possible for CNNs, but rather hard in training time; thus, breaking up the minimizer information across multiple filters likely makes it easier to train. This topic deserves further study.

5 Conclusion

In this manuscript, through a careful probability computation, we proved that the family of hash functions defined by Gaussian dot products is useful for minimizer style analysis. Furthermore, by recasting CNN convolutional filters in a hash-function based framework, we were able to demonstrate their equivalence to minimizers, one of the workhorses of computational biology. We hope that the connection proves fruitful for both methods, enabling the design of better minimizers, as well as providing some mathematically rigorous explanation for why CNNs work in categorical data.
6 Acknowledgements

I thank Daphne Ippolito, Jim Shaw, and David Rolnick for fruitful discussions. This work was supported by startup funding from the University of Toronto at Scarborough Department of Computer and Mathematical Sciences.

References

1. Angermueller, C., Pärnamaa, T., Parts, L., Stegle, O.: Deep learning for computational biology. Molecular systems biology 12(7), 878 (2016)
2. Appleby, A.: Murmurhash project. 2008 (2008)
3. Baker, D.N., Langmead, B.: Dashing: fast and accurate genomic distances with hyperloglog. Genome biology 20(1), 1–12 (2019)
4. Berger, B., Daniels, N.M., Yu, Y.W.: Computational biology in the 21st century: Scaling with compressive algorithms. Communications of the ACM 59(8), 72–80 (2016)
5. Broder, A.Z.: On the resemblance and containment of documents. In: Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171). pp. 21–29. IEEE (1997)
6. Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise independent permutations. Journal of Computer and System Sciences 60(3), 630–659 (2000)
7. Caron, M., Bojanowski, P., Joulin, A., Douze, M.: Deep clustering for unsupervised learning of visual features. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 132–149 (2018)
8. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. Journal of computer and system sciences 18(2), 143–154 (1979)
9. Cohen, E.: Min-hash sketches. (2016)
10. Dahlgaard, S., Thorup, M.: Approximately minwise independence with twisted tabulation. In: Scandinavian Workshop on Algorithm Theory. pp. 134–145. Springer (2014)
11. Deepak, S., Ameer, P.: Brain tumor classification using deep cnn features via transfer learning. Computers in biology and medicine 111, 103345 (2019)
12. Dietzfelbinger, M.: Universal hashing via integer arithmetic without primes, revisited. In: Adventures Between Lower Bounds and Higher Altitudes, pp. 257–279. Springer (2018)
13. Eastlake, D., Jones, P.: Us secure hash algorithm 1 (sha1) (2001)
14. Edgar, R.: Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. PeerJ 9, e10805 (2021)
15. Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: Proceedings 41st Annual Symposium on Foundations of Computer Science. pp. 390–398. IEEE (2000)
16. Fiannaca, A., La Paglia, L., La Rosa, M., Bosco, L., Renda, G., Rizzo, R., Gaglio, S., Urso, A., et al.: Deep learning models for bacteria taxonomic classification of metagenomic data. BMC bioinformatics 19(7), 61–76 (2018)
17. Flajolet, P., Fusy, É., Gandouet, O., Meunier, F.: Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In: Discrete Mathematics and Theoretical Computer Science. pp. 137–156. Discrete Mathematics and Theoretical Computer Science (2007)
18. Frith, M.C., Noé, L., Kucherov, G.: Minimally overlapping words for sequence similarity search. Bioinformatics 36(22-23), 5344–5350 (2020)
19. Fukushima, K., Miyake, S.: Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In: Competition and cooperation in neural nets, pp. 267–285. Springer (1982)
20. Gagie, T., Navarro, G., Prezza, N.: Optimal-time text indexing in bwt-runs bounded space. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1459–1477. SIAM (2018)
21. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
22. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural networks 2(5), 359–366 (1989)
23. Hubel, D.H., Wiesel, T.N.: Receptive fields of single neurones in the cat’s striate cortex. The Journal of physiology 148(3), 574–591 (1959)
24. Indyk, P.: A small approximately min-wise independent family of hash functions. Journal of Algorithms 38(1), 84–90 (2001)
25. Jaccard, P.: The distribution of the flora in the alpine zone. 1. New phytologist 11(2), 37–50 (1912)
26. Jain, C., Rhie, A., Zhang, H., Chu, C., Walenz, B.P., Koren, S., Phillippy, A.M.: Weighted minimizer sampling improves long read mapping. Bioinformatics 36(Supplement 1), i111–i118 (2020)
27. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al.: In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th annual international symposium on computer architecture. pp. 1–12 (2017)
28. Kahn, S.D.: On the future of genomic data. science 331(6018), 728–729 (2011)
29. Klar, B.: A note on gamma difference distributions. Journal of Statistical Computation and Simulation 85(18), 3708–3715 (2015)
30. Kuhnle, A., Mun, T., Boucher, C., Gagie, T., Langmead, B., Manzini, G.: Efficient construction of a complete index for pan-genomics read alignment. Journal of Computational Biology 27(4), 500–513 (2020)
31. Lawrence, S., Giles, C.L., Tsoi, A.C., Back, A.D.: Face recognition: A convolutional neural-network approach. IEEE transactions on neural networks 8(1), 98–113 (1997)
32. Li, H.: Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. Bioinformatics 32(14), 2103–2110 (2016)
33. Lindenstrauss, W.J.J.: Extensions of lipschitz maps into a hilbert space. Contemp. Math 26(189-206), 2 (1984)
34. Liu, Y., Palmedo, P., Ye, Q., Berger, B., Peng, J.: Enhancing evolutionary couplings with deep convolutional neural networks. Cell systems 6(1), 65–74 (2018)
35. Lu, C.P.: K3 moore’s law in the era of gpu computing. In: Proceedings of 2010 International Symposium on VLSI Design, Automation and Test. pp. 5–5. IEEE (2010)
36. Montavon, G., Samek, W., Müller, K.R.: Methods for interpreting and understanding deep neural networks. Digital Signal Processing 73, 1–15 (2018)
37. Ondov, B.D., Treangen, T.J., Melsted, P., Mallonee, A.B., Bergman, N.H., Koren, S., Phillippy, A.M.: Mash: fast genome and metagenome distance estimation using minhash. Genome biology 17(1), 1–14 (2016)
38. Poplin, R., Chang, P.C., Alexander, D., Schwartz, S., Colthurst, T., Ku, A., Newburger, D., Dijamco, J., Nguyen, N., Afshar, P.T., et al.: A universal snp and small-indel variant caller using deep neural networks. Nature biotechnology 36(10), 983–987 (2018)
39. Roberts, M., Hayes, W., Hunt, B.R., Mount, S.M., Yorke, J.A.: Reducing storage requirements for biological sequence comparison. Bioinformatics 20(18), 3363–3369 (2004)
40. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. nature 323(6088), 533–536 (1986)
41. Schleimer, S., Wilkerson, D.S., Aiken, A.: Winnowing: local algorithms for document fingerprinting. In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data. pp. 76–85 (2003)
42. Shaw, J., Yu, Y.W.: Theory of local k-mer selection with applications to long-read alignment. bioRxiv (2021)
43. Simpson, J.T., Durbin, R.: Efficient construction of an assembly string graph using the fm-index. Bioinformatics 26(12), i367–i373 (2010)
44. Thorup, M.: High speed hashing for integers and strings. arXiv preprint arXiv:1504.06804 (2015)
45. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
46. Wegman, M.N., Carter, J.L.: New classes and applications of hash functions. In: 20th Annual Symposium on Foundations of Computer Science (sfcs 1979). pp. 175–182. IEEE (1979)
47. Wood, D.E., Lu, J., Langmead, B.: Improved metagenomic analysis with kraken 2. Genome biology 20(1), 1–13 (2019)
48. Yorukoglu, D., Yu, Y.W., Peng, J., Berger, B.: Compressive mapping for next-generation sequencing. Nature biotechnology 34(4), 374–376 (2016)
49. Yu, Y.W., Daniels, N.M., Danko, D.C., Berger, B.: Entropy-scaling search of massive biological data. Cell systems 1(2), 130–140 (2015)
50. Yu, Y.W., Weber, G.M.: Hyperminhash: Minhash in loglog space. IEEE Transactions on Knowledge and Data Engineering pp. 1–1 (2020). https://doi.org/10.1109/TKDE.2020.2981311
51. Zheng, H., Kingsford, C., Marcais, G.: Improved design and analysis of practical minimizers. Bioinformatics 36(Supplement 1), i119–i127 (2020)