Weightless Neural Networks as Memory Segmented Bloom Filters

Leandro Santiago, Leticia Verona, Fabio Rangel,
Fabrício Firmino, Daniel S. Menasché

Universidade Federal do Rio de Janeiro (UFRJ), RJ, Brazil

Wouter Caarls

Pontifcia Universidade Catlica do Rio de Janeiro (PUC), RJ, Brazil

Mauricio Breternitz Jr

Lisbon University Institute- ISCTE-IUL & ISTAR-IUL, Lisbon, Portugal

Sandip Kundu

University of Massachusetts Amherst, Dept. of Electrical and Computer Engineering, USA

Priscila M.V. Lima, Felipe M. G. França

Universidade Federal do Rio de Janeiro (UFRJ), RJ, Brazil

Abstract

Weightless Neural Networks (WNNs) are Artificial Neural Networks based on RAM memory broadly explored as solution for pattern recognition applications. Memory-oriented solutions for pattern recognition are typically very simple, and can be easily implemented in hardware and software. Nonetheless, the straightforward implementation of a WNN requires a large amount of memory resources making its adoption impracticable on memory constrained systems. In this paper, we establish a foundational relationship between WNN and Bloom filters, presenting a novel unified framework which encompasses the two. In particular, we indicate that a WNN can be framed as a memory segmented Bloom filter. Leveraging such finding, we propose a new model of WNNs which utilizes Bloom

☆This is an extended version of the paper presented at ESANN'2019 and invited for the Neurocomputing ESANN 2019 Special Issue (Paper ESANN2019-83).

filters to implement RAM nodes. Bloom filters reduce memory requirements, and allow false positives when determining if a given pattern was already seen in data. We experimentally found that for pattern recognition purposes such false positives can build robustness into the system. The experimental results show that our model using Bloom filters achieves competitive accuracy, training time and testing time, consuming up to 6 orders of magnitude less memory resources when compared against the standard Weightless Neural Network model.

Keywords: Weightless neural network, Bloom filter, Discriminator

2010 MSC: 00-01, 99-00

---

1. Introduction

Weightless Neural Networks (WNNs) [1] are neuron models based on Random Access Memory (RAM) where each neuron is defined through a RAM node. These models have been shown as attractive solutions to solve pattern recognition and artificial consciousness applications achieving competitive performance against other state of the art solutions. WiSARD (Wilkie, Stoneham and Aleksander's Recognition Device) is the pioneering WNN distributed commercially [2] which provides simple and efficient implementation enabling to deploy learning capabilities into real-time and embedded systems.

The straightforward WiSARD implementation needs a considerable amount of memory resources to obtain good learning features. For example, a 1024×1024 binary input with total size of $1,048,576$ bits can be split into $16,384$ tuples of 64 bits each ($64 \times 16,384 = 1,048,576$). Each tuple is then mapped into a RAM. In this configuration, each RAM consumes 2^{64} locations which is impracticable to be implemented in current embedded systems. To deal with those constraints, the RAMs are commonly implemented using dictionary/hash table structures where the tuple values are stored as key-value pairs, with the key representing the memory address and the value being the content of the RAM position (either 0 or 1, under the original WiSARD design [3] or a non-negative integer, in case of WiSARD with bleaching capability [4]).

Our key insight consists of observing that RAMs play the role of *filters* under WNN designs. By allowing additional flexibility in the implementation of RAMs, one can explore a wide range of solutions trading between memory costs, classifier accuracy and computational complexity. Consider an input vector divided into N tuples of length M bits each. Then, the naive implementation of each of the N RAMs using a vector of size 2^M bits is memory expensive, but less computationally costly than a dictionary serving the purpose of indicating whether each bit in the RAM is set to 1 or 0. Alternatively, a Bloom filter may be used to trade between the aforementioned costs, opening up a broad range of opportunities to tune the model accuracy by providing additional degrees of freedom in the design of WiSARD classifiers.

We propose a new WiSARD model that leverages Bloom filters for the implementation of RAMs. Bloom filters [5, 6] are probabilistic data structures which represent a set as small bit array allowing the occurrences of false positives, i.e., in a Bloom filter, an element can be incorrectly classified as member of a set when it is not. *Although false positives detract certain applications, we experimentally discovered that for pattern recognition purposes they can build robustness into the system (as dropout does to deep neural networks). Bloom WiSARD presents similar accuracy when contrasted against WiSARD, but uses significantly less resources and, in this sense, is more robust than WiSARD.* We discuss a unified framework to bridge Bloom filter and WiSARD concepts which might be easily extended to other machine learning tools. Our experiments analyze accuracy, training time, testing time and memory consumption of our model compared against standard WiSARD and WiSARD implemented with hash tables (see Table 1).

The rest of the paper is organized as follows. Background related to this work is presented in Section 2. The unified framework bridging Bloom filters and WiSARD is discussed in Section 3. Section 4 presents the new WiSARD model based on Bloom Filters and describes its implementation. In Section 5, we show the experiments and results related to Bloom WiSARD. Finally, related work is discussed in Section 6 and we conclude this work in Section 7.

Table 1: Comparison of classifiers: simpler models such as Bloom WiSARD typically favor generalization. WiSARD and Dict WiSARD are logically equivalent whereas Bloom WiSARD has more degrees of freedom.

	Space per discriminator	Use of hashes for	Accuracy
WiSARD	$N2^M$ bits	no hashes	reference accuracy
Dict WiSARD	significantly less than WiSARD (worst case equal)	exact set membership (with collision checking)	equal to WiSARD (given mapping from input to RAM)
Bloom WiSARD	typically similar to Dict WiSARD (tunable by design)	approximate set membership (no collision checking)	potentially greater than WiSARD (hashes are tunable)

2. Background

In this Section, we briefly give the relevant background on the WiSARD discriminator followed by Bloom filter concepts.

2.1. WiSARD

WiSARD (Wilkie, Stoneham and Aleksander's Recognition Device) is a multi-discriminator WNN model proposed in the early 80's [2] that recognizes patterns from binary data. Each class is represented by a discriminator which contains a set of RAMs. A binary input with $N \times M$ bits is split into N tuples of M bits. Each tuple n, $n = 1, \ldots, N$, is a memory address to an entry of the n-th RAM. Each RAM contains 2^M locations.

A *pseudo-random mapping* is a deterministic function that maps each binary input matrix to a set of N tuples of M bits each. The function is typically a pseudo-random shuffling of the binary input matrix, hence the name pseudo-random mapping. Each discriminator may be associated to a different pseudo-random mapping, that must remain the same across training and classification phases.

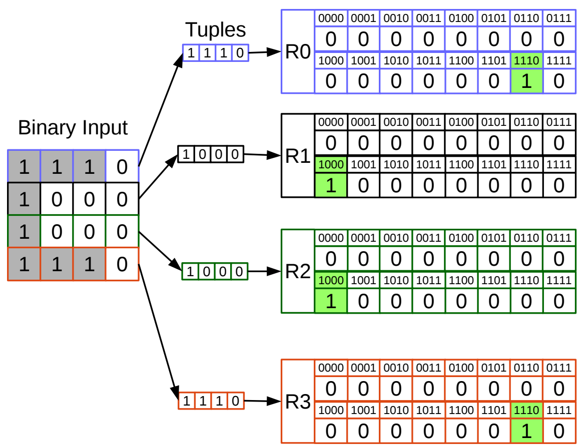


Figure 1: Example of training in WiSARD.

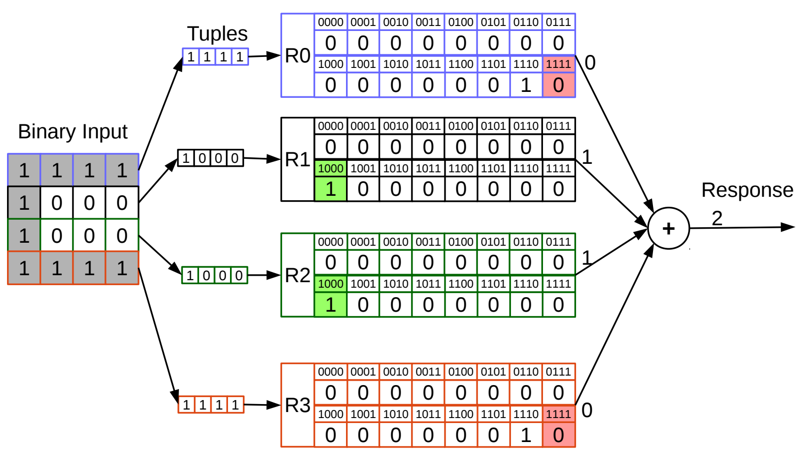


Figure 2: Example of testing operation in one WiSARD discriminator.

At the training phase, initially all RAMs have their locations set to zero (0). Each training sample is treated by the corresponding discriminator which sets to one (1) all accessed RAM positions as illustrated in Figure 1. At the

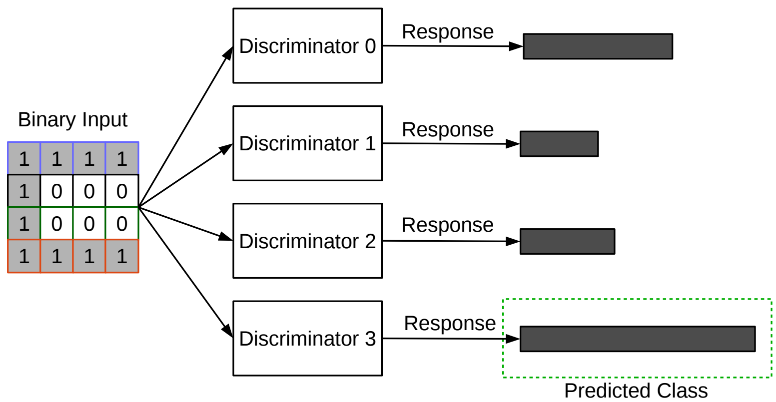


Figure 3: Example of testing operation to WiSARD select predicted class.

classification phase, the input is sent to all discriminators generating responses per discriminators by summing all accessed RAM values as shown in Figure 2. The discriminator with the highest response is chosen as the representative class of the input as exemplified in Figure 3.

2.2. WiSARD based on Dictionary

For certain applications, the standard WiSARD implementation requires a considerable amount of memory resources in order to achieve the required learning results. To deal with this constraints, the RAMs are commonly implemented using dictionary/hash table structures (see Section 1). The tuple values are stored as key-value pairs, with the key representing the memory address and the value being the content of the RAM position (either 0 or 1, under the original WiSARD design [3] or a non-negative integer, in case of WiSARD with bleaching capability [4]).

Similar to standard WiSARD, each tuple from the binary input is stored to the corresponding hash table at the training phase as illustrated in Figure 4. In the classification phase, the responses are generated by adding up the results collected from the hash tables as shown in Figure 5.

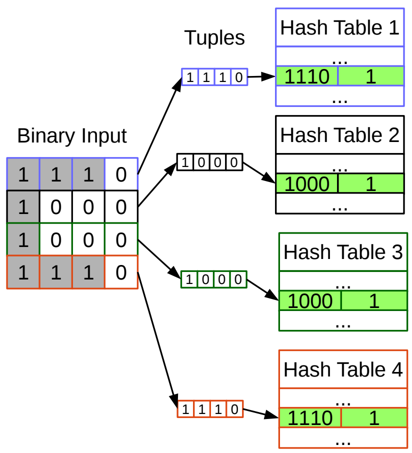


Figure 4: Example of training of a dictionary-based WiSARD.

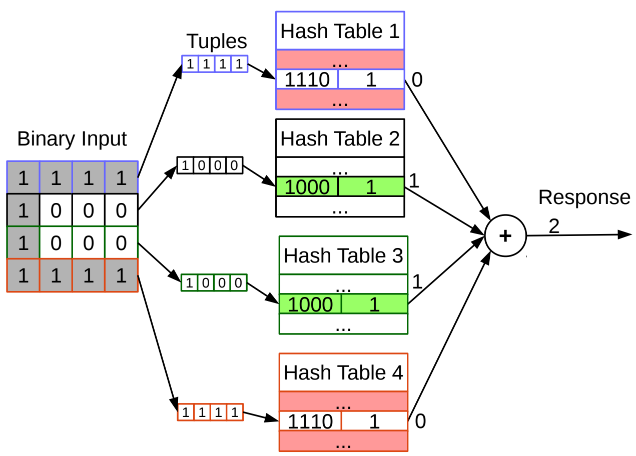


Figure 5: Obtaining the response from a dictionary-based WiSARD discriminator.

2.3. Bloom filter

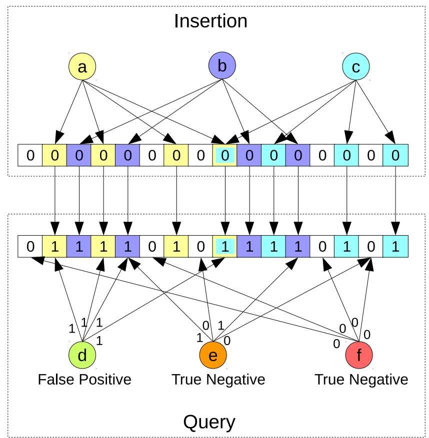


Figure 6: Bloom filter operations example with 16-bit array and 4 hash functions.

Bloom filters [5] are space-efficient data structures for Approximate Membership Query (AMQ) which test whether an element belongs to a given set or not with a certain false positive probability. In other words, sometimes the membership query will respond that an element is stored in the considered set even if it is not. A Bloom filter is composed of an m-bit array and k independent hash functions that map an element into k bit array positions. The algorithm is easily extended for application in WISARD. Bloom filters are commonly used in the network and database domains to provide approximately correct answers to set membership queries, and a number of efficient implementations of Bloom filters have been proposed [7].

For the purposes of set-membership queries, a single-index hash table is at greater risk of returning many false positives. Consider an element A that belongs to a particular set S. A hash of A provides an index to a particular bit in the table, and one sets this bit to 1 to indicate membership in S. However, another element B not belonging to S may hash to the same entry as A, which results in the reporting of a false positive. A Bloom filter uses multiple hashes for each element, potentially setting several bits in the table for each element that belongs to class S. Consider the same element A hashing into $k = 3$ different locations. In the classical Bloom filter, an element is considered to be in S when the bits at all hashed locations are set. Figure 6 also shows element B and the three entries it hashes to. One of the entries collides with one of As entries; however, there exists at least one other entry that is not set, and so the Bloom filter correctly classifies B as not belonging to the set. For a Bloom filter to report a false positive a hash collision must occur for each and every one of the k hash functions.

The standard Bloom filter supports insertion and query operations as exemplified in Figure 6. Initially, all bit array positions are zeroed. In the insertion operation, an element is mapped into k positions of the bit array designated by the k hash functions and the corresponding k bits are set to 1. In the example, a, b and c are inserted using 4 hash functions. The query operation looks up the k positions mapped from the input element, indicating it as either a member of the set, considering a false positive rate if all values are $1's$, or a non-member when any value is 0. In Figure 6, d is a false positive since it was suggested as member of the set (only a, b and c were inserted), while e and f do not belong to the set. Note that a Bloom filter always reports a true negative whenever an element is not a member.

The false positive probability p is affected by the parameters m, n and k, corresponding to bit array size, number of elements to store and number of hash functions, respectively [8]. Given the target false positive probability p and capacity n, parameters m and k can be set as follows: $m = -n\ln(p)/\ln(2)^2$ [9] and $k = m\ln(2)/n$ [8].

3. A unified framework bridging Bloom filters and WiSARD

3.1. Machine learning and Bloom filters

Machine learning can be leveraged to improve the design of Bloom filters [10, 11] and, reciprocally, Bloom filters can be used in the design of general-purpose machine learning tools [12, 13, 14, 15, 16]. In the second direction, learning useful features in an effective way is one of the key machine learning challenges. The success of convolutional neural networks (CNN) stands for its ability to efficiently derive useful features, directly from data, with few parameters [13]. Alternatively, Bloom filters can be instrumental in the derivation of such features.

Machine learning discriminators are *filters*, as they filter the elements that should be discriminated from the remainder of the population. Such observation suggests that foundational results on Bloom filters can be applied to improve the design of discriminators, and has grounds in biological models relating familiarity mechanisms in the brain to filters [17]. In this direction of research, we encompass the search for a unified classification framework wherein Bloom filters and other machine learning tools, such as WiSARD, are special instances.

3.2. Similarities and differences between Bloom filters and WiSARD

WiSARD and Bloom filters are closely related data structures. Both store data in a binary RAM indexed by a function computed over the input. In the case of WiSARD, the index is determined by the pseudo-random mapping, interpreting a certain pattern of bits from the input as a binary number. For Bloom filters, it is a hash function.

An important distinction is the fact that WiSARD keeps a separate memory for every tuple of the pseudo-random mapping, whereas all hash functions in a Bloom filter index into the same hash table. Note that WiSARD accounts for a single hash function which maps each tuple instantiation, i.e., each string of bits comprising a tuple, into a position of the corresponding memory. Bloom filters, in contrast, account for multiple hash functions, wherein each hash function maps the whole input into a memory position, always assuming a single memory.

3.3. A unified framework bridging Bloom filters and WiSARD

3.3.1. Terminology

Let T be a set of vectors corresponding to the tuples comprising each input instance. The input instances are assumed to be binarized, i.e., each input instance is a binary vector of length $l = |x|$. For example, if $T = \{(1, 2), (3, 4), (5)\}$ then inputs of size $l = 5$ are divided into three tuples, $|T| = 3$, of sizes 2, 2 and 1. The first tuple corresponds to the first two entries of the input, the second tuple corresponds to the subsequent two entries, and the last tuple corresponds to the last entry.

Given an input x, the t-th tuple of the input is denoted by x_t, and its size is denote by M_t. Let F_t be the hash functions applied over the t-th tuple, $t = 1, \ldots, |T|$. Each $f \in F_t$ receives as input the t-th tuple of the input, and generates as output a memory position, $f \in F_t : x_t \in \{0, 1\}^{M_t} \to \mathbb{N}$. Whenever the hash functions applied over the tuples are all the same, we drop subscript t and denote the set of hash functions simply as F. Similarly, whenever all tuples have the same size, the latter is simply referred to as M.

Let $\mathcal{M}_t$ be a vector characterizing the memory corresponding to the t-th tuple, $\mathcal{M}_t[i] \in \{0, 1\}$ for $i = 1, \ldots, |\mathcal{M}_t|$. Under the Bloom filter framework, there is a single memory as the input is typically assumed to correspond to a single tuple. In that case, we drop subscript t and denote the memory simply as $\mathcal{M}$. Table 2 summarizes the notation.

3.3.2. Combining Bloom filters and WiSARD

We may combine Bloom filters and WiSARD into a general framework by explicitly accounting for the decisions about the number of tuples comprising the input, which translates into the number of memories, and the number of hash functions per tuple. Consider the training stage described in Algorithm 1.

If there is only a single tuple, i.e., the entire input is used at once ($|T| = 1$) and many functions ($|F| > 1$), we have a classical Bloom filter. Each function, in this case, is typically a special hash function which maps data from a large state space (large size) onto another state space of small size. Otherwise, the required memory would be prohibitive. On the other hand, if there are many tuples ($|T| > 1$) and only a single function ($|F| = 1$) we have WiSARD. In this case we may use a hash function but need a sufficiently large memory if we wish to avoid collision. WiSARD typically uses a collision-free function that simply

Table 2: Table of notation

variable	description
x	input (binary vector of length $\|x\|$)
T	set of tuples
$N = \|T\|$	number of tuples per input
M	size of each tuple, $M = \|x\|/N$ (when tuple sizes are heterogeneous we denote by M_t the t-th tuple size)
x_t	t-th tuple of input x, $t = 1, \ldots, N$ (binary vector of length M)
F	set of (hash) functions (when sets are heterogeneous across tuples we denote by F_t the t-th set of hashes)
$k = \|F\|$	number of (hash) functions (see Section 2.3)
$\mathcal{M}_t$	state of memory corresponding to t-th tuple (binary vector of length $\|\mathcal{M}_t\|$, $\|\mathcal{M}_t\| = 2^M$ in a classical WiSARD)
$\{\mathcal{M}_1, \ldots, \mathcal{M}_N\}$	discriminator state
$f(x_t)$	function that maps x_t into a position of $\mathcal{M}_t$

interprets the tuple as an address.

Given this generic framework, we can observe the possibility of using multiple hash functions per tuple, essentially creating multiple parallel Bloom filters. This greatly expands the range of usable tuple sizes, because memory size is no longer dictated by address size and can be tuned, in combination with the number of hash functions, to a desired collision rate.

The classification phase generalization is similar to that of the training, with the caveat that multiple tuples are needed in order to distinguish the most likely class. In the case of the classic Bloom filter (single tuple), the discriminator response R is a binary value, and can therefore only be used as a one-class classifier. The granularity of the discriminator responses increases when using more tuples (see Algorithm 2).

Algorithm 1 Unified framework for training Bloom filters and WiSARD discriminators

for all training examples x **do**
    for all tuples $t \in T$ **do**
        for all functions $f \in F_t$ **do**
            $\mathcal{M}_t[f(x_t)] = 1$
        end for
    end for
end for

Algorithm 2 Unified framework for determining Bloom filters and WiSARD discriminator responses

$R \leftarrow 0$
for all tuples $t \in T$ **do**
    for all functions $f \in F_t$ **do**
        $R_t \leftarrow R_t + \mathcal{M}_t[f(x_t)]$
    end for
    if $R_t = |F_t|$ **then**
        $R \leftarrow R + 1$
    end if
end for
return R

Note that in line 5 of the algorithm the response from tuple t, R_t, is compared against the maximum response, $|F_t|$. If they are equal, this means that the given tuple is stored in memory, and the final response R is incremented by one unit. In particular, note that by restricting the increment to the scenario wherein $R_t = |F_t|$ naturally prevents false negatives. Alternatively, requiring less than the full amount of hash hits in line 6 consists of an additional generalization of Bloom filters, allowing for both false positives and false negatives when assessing the pertinence of a given element to a given class.

Recall that for each target class there is a corresponding discriminator. Given an input, for each discriminator the algorithm above returns a value R. Then, the discriminators are compared against each other through the corresponding returned values. The discriminator that yields maximum return is typically chosen as the class corresponding to the given input. There are multiple variations with respect to how the returned value R is computed given the input (e.g., depending on whether one accounts for bleaching [4]), but the algorithm above serves to capture the essence behind all variants.

3.4. Collision rates

There is a vast literature on dimensioning and tuning Bloom filters to achieve a given target collision rate [18]. We envision that by establishing connections between Bloom filters and WiSARD we can leverage such results for the dimensioning of WiSARD. The dimensioning of WiSARD, based on first principles, in turn, is a vastly unexplored field. The same holds for many other discriminators, including CNNs [13, 19, 20], for which most of the tuning is executed in an ad-hoc experimental fashion. The perspective of doing such tuning of Bloom WiSARD based on first principles may shed light into other discriminators.

Note that collisions are a negative side effect in the realm of Bloom filters. In the realm of WiSARD, in contrast, collisions may actually increase the accuracy of the discriminator. This is because a higher collision rate may correspond to an increased capacity of generalization, which may, in turn, benefit the system. Therefore, the system should first be tuned to attain a desired target collision rate, greater than zero. Then, as a second step that target may be varied to find the best spot accounting for the training and test sets.

3.4.1. Perspectives towards collision rate tuning

Next, we briefly overview two different approaches for collision rate tuning. The first is inspired by the architectural design of Bloom filters in hardware. The second is based on an extension of Bloom filters to answer queries such as "is x close to an element in S?" rather than "is x an element in S?" [21, 22]

Recent advances in the design of Bloom filters [7] accounting for its hardware implementation suggest that dividing the memory used to implement a Bloom filter into separate memory banks is beneficial. The efficient implementation of Bloom filters in hardware involves the manipulation of hash functions to avoid collisions and to make simultaneous access to multiple memory banks. Such manipulation of hash functions proposed in [7] accounting for multiple memory banks is similar in spirit to the manipulation of hash functions and of sizes of the memories of a WiSARD discriminator, as advocated in this work, albeit for very different purposes. Whereas in [7] the goal is to improve the efficiency of the Bloom filter implementation in hardware, our goal is to increase the accuracy of WiSARD discriminators. In both cases, the collision rates must be controlled to achieve the desired goals.

A structured way of dealing with collisions under the Bloom WiSARD framework to improve classification accuracy may involve "distance sensitive Bloom filters" [23]. Under distance sensitive Bloom filters, similar inputs are mapped into similar memory positions. We envision that such similarity may be explored in the design of Bloom WiSARD, and leave that as subject for future work [21].

4. WiSARD based on Bloom Filters

When adopting a WiSARD architecture, the binary transformation impacts the accuracy and the learning capacity of the model affecting its input size, which determines the number of RAMs and the tuple size for each discriminator. Thus, huge RAMs might be required to achieve a good accuracy.

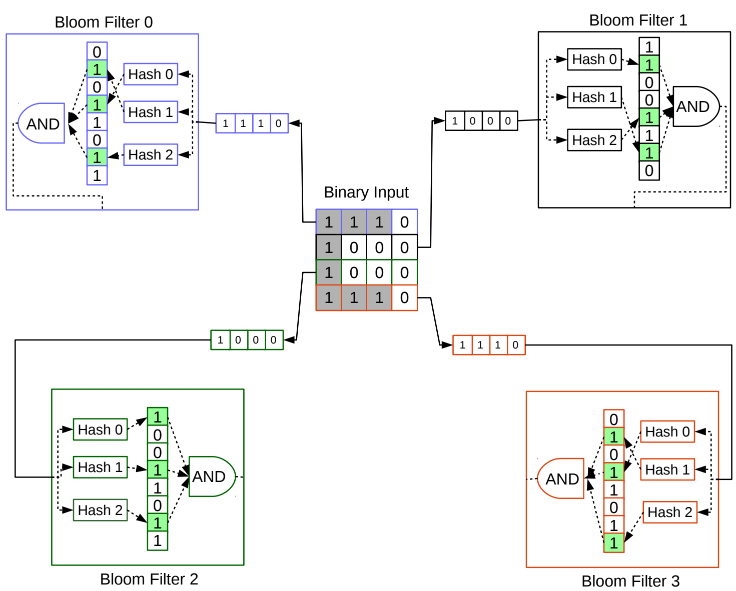


Figure 7: Example of training in Bloom WiSARD with 16-bit input, 4-bit tuples and 4 Bloom filters.

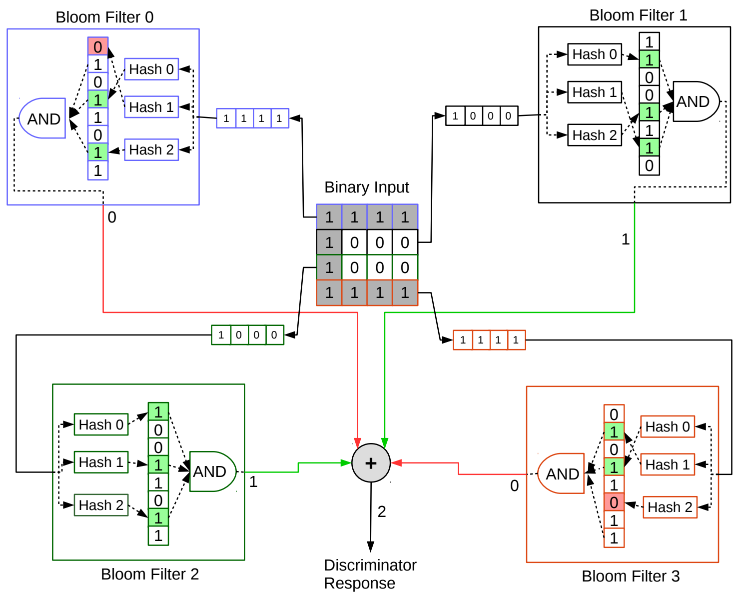


Figure 8: Example of classification in Bloom WiSARD with 16-bit input, 4-bit tuples and 4 Bloom filters.

The memory structures subsumed by a WiSARD WNN are typically sparse. We extend WiSARD by replacing RAMs with Bloom filters to reduce its memory footprint by avoiding storage of irrelevant zero positions. The new model is termed Bloom WiSARD.

On the training phase, the tuples are inserted into Bloom filters by updating the k bit array positions as depicted in Figure 7. On the classification phase, the tuples are queried into their associated Bloom filters returning whether each tuple is a member or not by ANDing all k bit values as presented in Figure 8. Similar to WiSARD, the discriminator responses are calculated by summing the N Bloom filter membership results. The responses of the discriminators are then compared, and the class corresponding to the discriminator with the highest response is selected.

Our Bloom WiSARD implementation utilizes a double hashing technique [24] to generate k hash functions in the form: $h(i, k) = (h_1(k) + i \times h_2(k)) \pmod{n}$, where h_1 and h_2 are universal hash functions. We adopt MurmurHash for h_1 and h_2 [25].

5. Experiments and Results

Table 3: Specification of binary classification data sets.

Dataset	**# Train**	**# Test**	**# Features**
Adult	$32,561$	$16,281$	14
Australian	460	230	14
Banana	$3,532$	$1,768$	2
Diabetes	512	256	8
Liver	230	115	6
Mushroom	$5,416$	$2,708$	22

To evaluate the proposed model, we compare Bloom WiSARD against two different WiSARD versions: standard WiSARD introduced in Section 2.1 and

Table 4: Specification of multiclass classification data sets.

Dataset	# Train	# Test	# Features	# Classes
Ecoli	224	112	7	8
Glass	142	72	9	7
Iris	100	50	4	3
Letter	$13,332$	$6,668$	16	26
MNIST	$60,000$	$10,000$	784	10
Satimage	$4,435$	$2,000$	36	6
Segment	$1,540$	770	19	7
Shuttle	$43,500$	$14,500$	9	7
Vehicle	564	282	18	4
Vowel	660	330	10	11
Wine	118	60	13	3

Table 5: Hyper-parameters of binary classification data sets. All the parameters are the same for the three WiSARD versions: thermometer bit (Therm.) is the length of numerical attributes in binary format, nominal bit is the number of 1's used to represent each value of the categorical attribute using one hot encoding and capacity is the Bloom filter configuration. The value $-$ indicates that there is no numerical (Therm.) or categorical attributes in the dataset.

Dataset	Tuple size	Therm. (Bits)	Nominal (Bits)	Capacity
Adult	28	128	30	500
Australian	20	20	5	460
Banana	20	512	$-$	50
Diabetes	20	20	$-$	512
Liver	20	64	$-$	100
Mushroom	20	$-$	5	100

Table 6: Hyper-parameters of multiclass classification data sets. All the parameters are the same for the three WiSARD versions: thermometer bit (Therm.) is the length of numerical attributes in binary format, nominal bit is the number of 1's used to represent each value of the categorical attribute using one hot encoding and capacity is the Bloom filter configuration. The value − indicates that there is no numerical (Therm.) or categorical attributes in the dataset.

Dataset	**Tuple size**	**Therm. (Bits)**	**Nominal (Bits)**	**Capacity**
Ecoli	20	20	–	100
Glass	20	128	–	100
Iris	20	20	–	100
Letter	28	20	–	500
MNIST	28	–	–	5000
Satimage	20	20	–	100
Segment	20	20	–	100
Shuttle	20	20	–	100
Vehicle	20	20	–	100
Vowel	20	20	–	100
Wine	20	20	–	100

Table 7: Accuracy and memory results of classifiers in binary classification problems.

Dataset	**WNN**	**Acc**	**Acc (Std)**	**Memory (KB)**	**Stats. signif.**
Adult	WiSARD	0.722	0.0069129626	8978432	
	Dict WiSARD	0.721	0.0055560122	383.535	✗
	Bloom WiSARD	0.718	0.0061495748	80.173	✓
Australian	WiSARD	0.843	0.0166130202	4096	
	Dict WiSARD	0.841	0.0141203978	11.299	✗
	Bloom WiSARD	0.834	0.0223775813	8.613	✗
Banana	WiSARD	0.87	0.0054630514	13312	
	Dict WiSARD	0.871	0.0061359655	23.428	✗
	Bloom WiSARD	0.864	0.0057860498	3.047	✓
Diabetes	WiSARD	0.698	0.0202749051	2048	
	Dict WiSARD	0.689	0.0195351559	6.553	✗
	Bloom WiSARD	0.69	0.0262359291	4.793	✗
Liver	WiSARD	0.593	0.0406562425	5120	
	Dict WiSARD	0.587	0.0271486839	6.387	✗
	Bloom WiSARD	0.591	0.0483371899	2.344	✗
Mushroom	WiSARD	1.0	0	8192	
	Dict WiSARD	1.0	0	19.209	✗
	Bloom WiSARD	1.0	0	3.75	✗

dictionary WiSARD discussed in Section 2.2 (see Table 1). The implementations are made available on a GitHub repository [26].

5.1. Dataset

We select the MNIST database [27] and a subset of binary classification and multiclass classification datasets used in [28]. Most of the problems were taken from UCI public repository [29] and they have different characteristics in terms of number of samples, number of classes and number of features. Some datasets do not provide the training set and testing set in separated files. For these

Table 8: Accuracy and memory results of classifiers in multiclass classification problems.

Dataset	**WNN**	**Acc**	**Acc (Std)**	**Memory (KB)**	**Stats. signif.**
Ecoli	WiSARD	0.793	0.0284596026	7168	
Ecoli	Dict WiSARD	0.799	0.0233683077	5.664	✗
Ecoli	Bloom WiSARD	0.799	0.0202621531	3.281	✗
Glass	WiSARD	0.72	0.030516073	51968	
Glass	Dict WiSARD	0.73	0.0286242553	20.884	✗
Glass	Bloom WiSARD	0.726	0.0367137219	23.789	✗
Iris	WiSARD	0.985	0.01396424	1536	
Iris	Dict WiSARD	0.977	0.0284780617	0.747	✗
Iris	Bloom WiSARD	0.976	0.0215406592	0.703	✗
Letter	WiSARD	0.845	0.006130619	10223616	
Letter	Dict WiSARD	0.846	0.0044277676	121.748	✗
Letter	Bloom WiSARD	0.848	0.0045028728	91.292	✓
MNIST	WiSARD	0.917	0.0043519651	9175040	
MNIST	Dict WiSARD	0.916	0.0042990086	1368.457	✗
MNIST	Bloom WiSARD	0.915	0.0056781577	819.049	✗
Satimage	WiSARD	0.851	0.0080425043	27648	
Satimage	Dict WiSARD	0.853	0.0083887946	69.141	✗
Satimage	Bloom WiSARD	0.851	0.0057708318	12.656	✗
Segment	WiSARD	0.935	0.0079103597	17024	
Segment	Dict WiSARD	0.934	0.0077444423	7.724	✗
Segment	Bloom WiSARD	0.933	0.0080506388	7.793	✗
Shuttle	WiSARD	0.87	0.0107019751	8064	
Shuttle	Dict WiSARD	0.869	0.0112712713	4.956	✗
Shuttle	Bloom WiSARD	0.868	0.012279044	3.691	✗
Vehicle	WiSARD	0.67	0.021343718	9216	
Vehicle	Dict WiSARD	0.672	0.017094994	17.617	✗
Vehicle	Bloom WiSARD	0.662	0.0238480121	4.219	✗
Vowel	WiSARD	0.876	0.0161340516	14080	
Vowel	Dict WiSARD	0.876	0.0135044121	16.221	✗
Vowel	Bloom WiSARD	0.876	0.0262235043	6.445	✗
Wine	WiSARD	0.932	0.0260741464	4992	
Wine	Dict WiSARD	0.924	0.030945741	4.248	✗
Wine	Bloom WiSARD	0.926	0.0260741464	2.285	✗

Table 9: Training and testing time of classifiers in binary classification problems.

Dataset	**WNN**	**Training (s)**	**Training (Std)**	**Testing (s)**	**Testing (Std)**
Adult	WiSARD	4.414	0.8035190	1.05	0.0036088
	Dict WiSARD	1.947	0.0023436	1.188	0.0021732
	Bloom WiSARD	1.932	0.0024696	1.166	0.000236
Australian	WiSARD	0.002	$1.5453E-05$	0.001	$1.5212E-05$
	Dict WiSARD	0.002	$8.1612E-06$	0.001	$7.6577E-06$
	Bloom WiSARD	0.002	$1.0162E-05$	0.001	$1.5519E-05$
Banana	WiSARD	0.052	$9.7794E-05$	0.028	0.0003726
	Dict WiSARD	0.054	0.0001169	0.033	$9.0079E-05$
	Bloom WiSARD	0.058	$3.9803E-05$	0.036	0.0001168
Diabetes	WiSARD	0.001	$9.269E-06$	0.0007	$7.0736E-06$
	Dict WiSARD	0.001	0.000004	0.0008	$4.5087E-06$
	Bloom WiSARD	0.001	$4.0986E-06$	0.0008	$1.6984E-05$
Liver	WiSARD	0.001	$1.47E-05$	0.0007	$1.3299E-05$
	Dict WiSARD	0.001	$4.2355E-06$	0.0008	$3.3714E-06$
	Bloom WiSARD	0.001	$1.6464E-06$	0.0009	$1.1822E-06$
Mushroom	WiSARD	0.0509	$8.7859E-05$	0.0278	0.0003098
	Dict WiSARD	0.054	0.0001117	0.0335	$8.932E-05$
	Bloom WiSARD	0.057	0.0002169	0.0348	0.0001105

datasets, we adopt the same methodology applied in [28]: we randomly shuffle the data and partition it in 3 parts, such that 2/3 and 1/3 are used for training and testing sets, respectively. Table 3 and Table 4 show the parameters of the binary and multiclass classification data sets, respectively.

5.2. Experimental Setup

The experiments were performed on an Intel Core i7-6700(3.40GHz) processor with 32GB of RAM running Ubuntu Linux 16.04. The core of all WiSARD experiments was implemented in a single-thread C++11 library accessed

Table 10: Training and testing time of classifiers in multiclass classification problems.

Dataset	WNN	Training (s)	Training (Std)	Testing (s)	Testing (Std)
Ecoli	WiSARD	0.0005	$8.0849E-06$	0.0005	0.000020714
	Dict WiSARD	0.0005	$2.3084E-06$	0.0005	$4.390E-06$
	Bloom WiSARD	0.0005	0.000001152	0.0007	$2.7085E-06$
Glass	WiSARD	0.003	$2.6116E-05$	0.003	$4.6433E-05$
	Dict WiSARD	0.003	$2.0587E-05$	0.003	0.000032357
	Bloom WiSARD	0.003	$3.5548E-06$	0.003	$1.1031E-05$
Iris	WiSARD	0.0001	$8.5413E-06$	0.000009	0.000002083
	Dict WiSARD	0.0001	$6.9818E-06$	0.000008	$9.8382E-07$
	Bloom WiSARD	0.0001	0.00000131	0.0001	$6.6729E-06$
Letter	WiSARD	1.483	0.9651815243	0.16	0.0082967305
	Dict WiSARD	0.0717	0.0001674798	0.22	0.0006118143
	Bloom WiSARD	0.07	0.000032443	0.208	0.0003070775
MNIST	WiSARD	4.317	2.1310536808	0.33	0.0095871641
	Dict WiSARD	0.811	0.0025265669	0.475	0.0036404496
	Bloom WiSARD	0.775	0.0037293041	0.369	0.0007896101
Satimage	WiSARD	0.048	$6.9346E-05$	0.034	0.0006351119
	Dict WiSARD	0.05	$8.8147E-05$	0.049	0.0002178829
	Bloom WiSARD	0.053	$7.6182E-05$	0.05	0.0001132872
Segment	WiSARD	0.009	$3.3891E-05$	0.007	$4.7123E-05$
	Dict WiSARD	0.009	$2.0399E-05$	0.01	$6.513E-05$
	Bloom WiSARD	0.01	$2.3205E-05$	0.011	$2.9526E-05$
Shuttle	WiSARD	0.119	0.0001553173	0.064	0.0013783008
	Dict WiSARD	0.12	0.0002099746	0.078	0.0006038582
	Bloom WiSARD	0.132	$8.0745E-05$	0.103	0.0003318945
Vehicle	WiSARD	0.003	$1.9918E-05$	0.0021	$3.9435E-05$
	Dict WiSARD	0.003	$1.0621E-05$	0.0026	$1.4197E-05$
	Bloom WiSARD	0.003	$3.2052E-06$	0.0028	$8.6081E-06$
Vowel	WiSARD	0.0023	$1.6889E-05$	0.0025	$4.5306E-05$
	Dict WiSARD	0.0023	$1.083E-05$	0.0032	$5.1637E-05$
	Bloom WiSARD	0.0022	$7.5554E-06$	0.0036	0.000012788
Wine	WiSARD	0.0006	$1.0351E-05$	0.0003	$1.54E-05$
	Dict WiSARD	0.0005	$6.0381E-06$	0.0003	$2.096E-06$
	Bloom WiSARD	0.0005	$6.4565E-06$	0.0004	$1.2886E-06$

through a Python interface. To convert the input attributes to binary format, we concatenate all binary attributes using thermometer (resp., hot encoding) to transform the continuous (resp., categorical) attributes. The input size, number of RAMs and tuple size varied according to the dataset, but were kept constant across all considered WiSARD architectures. Bloom filters are setup with 10% of false positive probability. The capacities were empirically selected for each dataset and m and k were obtained through the formulas presented in Section 2.3. Table 5 and Table 6 show the hyper-parameters of the binary and multiclass classification data sets, respectively.

5.3. Accuracy, Performance and Memory Consumption Results

All results are obtained through the mean of 20 runs with negligible standard deviation. Table 7 and Table 8 show the results for binary classification and multiclass classification datasets, respectively. Note that the accuracy of Dict WiSARD and WiSARD slightly differ as we used different pseudo-random mappings at each training epoch (see Table 1). We ran statistical hypothesis tests to check if the gain or loss in accuracy of Bloom WiSARD against WiSARD is statistically significant. We used a two tail test, with significance value of 5%. The results are reported in the last column of Table 7 and Table 8. A check mark indicates that the gains or losses of accuracy of Bloom WiSARD and Dict WiSARD are statistically significant when compared against WiSARD.

The training and testing time results are shown in Table 9 and Table 10 for binary classification and multiclass classification datasets, respectively. Overall, Bloom WiSARD achieved comparable accuracy, training time and testing time when compared against WiSARD and Dict WiSARD, while consuming a smaller amount of memory. Bloom WiSARD's memory consumption is reduced up to 6 orders of magnitude (Adult and Letter) compared against standard WiSARD and approximatelly 7.7 times (Banana) when compared against dictionary WiSARD. The memory resources can be further reduced by increasing the false positive rate and the accuracy can be increased by tuning the hash functions to capture essential aspects of the data, which we leave as subject for future work.

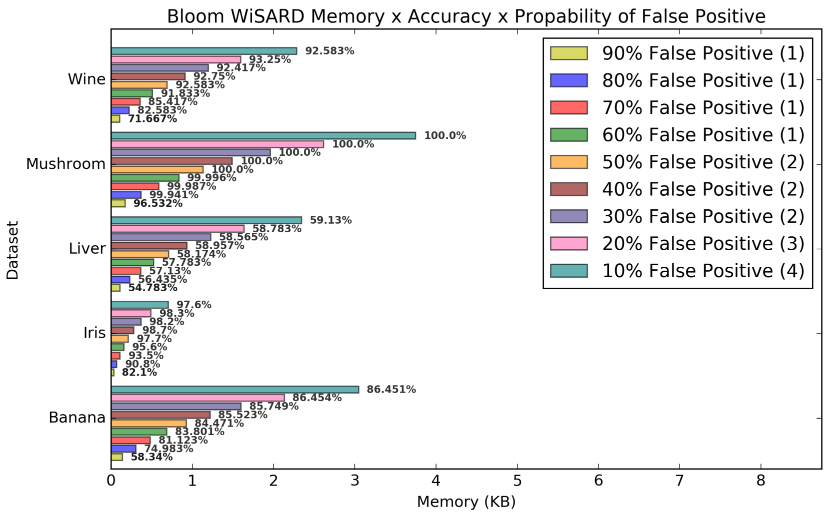
5.4. False Positive Rate vs. Accuracy vs. Memory Analysis

Table 11: Standard deviation of accuracy when varying the false positive rate (FPP) of Bloom WiSARD for binary classification problems.

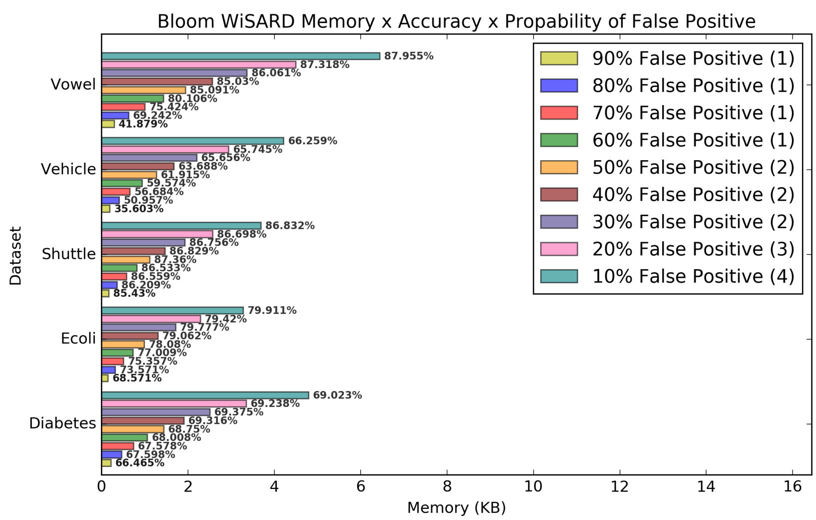
FPP	**Adult**	**Australian**	**Banana**	**Diabetes**	**Liver**	**Mushroom**
10%	0.0061496	0.0223776	0.0057860	0.0262359	0.0483372	0
20%	0.0058269	0.0155309	0.0058466	0.0284882	0.0366251	0
30%	0.0044720	0.018285	0.0078729	0.0250549	0.0299053	0
40%	0.0035480	0.021014	0.0061894	0.0166406	0.0413020	0
50%	0.0049795	0.019081	0.009687	0.0183635	0.030360	0
60%	0.0072392	0.0177929	0.0068795	0.0234342	0.035050	0.000161
70%	0.0059162	0.0185115	0.0168765	0.0175129	0.03928	0.0001761
80%	0.0082073	0.0226118	0.0269275	0.0239359	0.028563	0.000787
90%	0.0091519	0.0237056	0.0359376	0.0183749	0.0461771	0.0191718

In Section 5.3, the false positive rate of Bloom filters were fixed to 10%. In contrast to traditional use of Bloom filters where one needs to ensure correct query responses with high probability, Bloom WiSARD does not require low false positive rate because even if a tuple is erroneously returned as member of a Bloom filter, the model is not compromised and false positives can still improve the generalization capability of the system. In order to evaluate the potential of Bloom WiSARD, the accuracy and memory consumption are evaluated for different configurations of the false positive rate. For all data sets, the rate is varied from 10% to 90%.

Results are presented in Figure 9. Memory consumption and accuracy decrease as the false positive probability increases.Overall, the accuracy is kept acceptable until reaching a 50% false positive rate. At that point, accuracy is decreased on average by 1.3% with a worst case of about 4.3% (Vehicle). Ac-

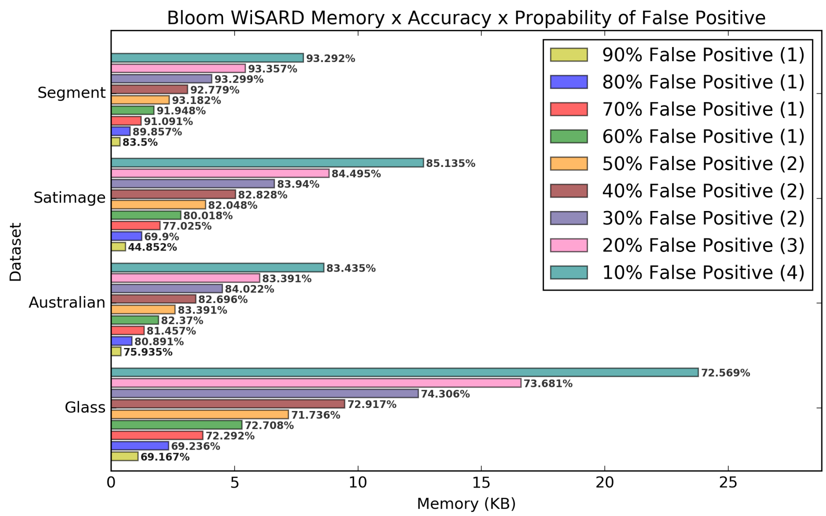


(a) Part 1: Wine, Mushroom, Liver, Iris and Banana

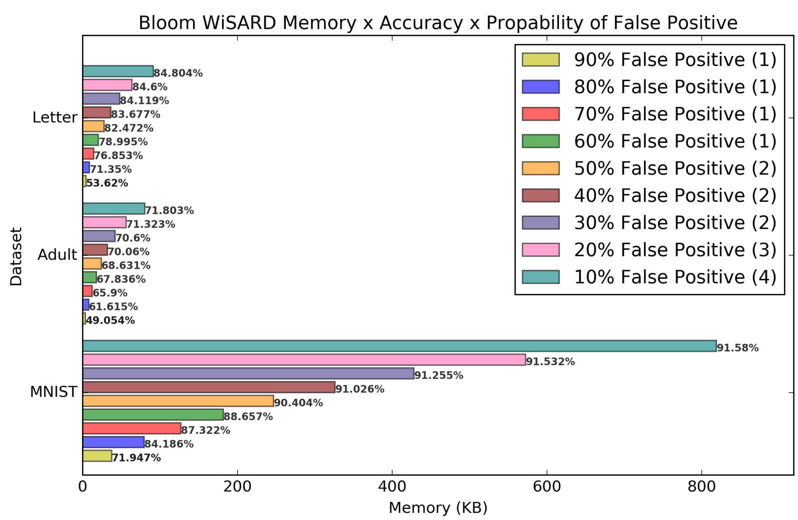


(b) Part 2: Vowel, Vehicle, Shuttle, Ecoli and Diabetes.

Figure 9: Accuracy and memory consumption results when varying the false positive rate of Bloom WiSARD. In the legend, the number of hash functions is shown in parentheses at end of each false positive rate. The accuracy is shown at right side of each bar.



(c) Part 3: Segment, Satimage, Australian and Glass.



(d) Part 4: Letter, Adult and MNIST.

Figure 9: (Cont.) Accuracy and memory consumption results when varying the false positive rate of Bloom WiSARD. In the legend, the number of hash functions is shown in parentheses at end of each false positive rate. The accuracy is shown at right side of each bar.

Table 12: Standard deviation of accuracy when varying the false positive rate (FPP) of Bloom WiSARD for multiclass classification problems.

FPP	**Ecoli**	**Glass**	**Iris**	**Letter**	**MNIST**	**Satimage**
10%	0.0202621	0.0367137	0.0215406	0.0045029	0.0056781	0.0057708
20%	0.0196581	0.0257881	0.0170587	0.005713	0.0042678	0.0076205
30%	0.0150135	0.027252	0.0177764	0.0056912	0.0027261	0.0056868
40%	0.0257965	0.0352734	0.014526	0.0050576	0.004139	0.0130475
50%	0.0179628	0.0225133	0.0247184	0.006447	0.0042574	0.0101778
60%	0.0252341	0.030198	0.02498	0.0084854	0.0046192	0.0139484
70%	0.0311094	0.0371317	0.0339853	0.0065570	0.0054063	0.0140646
80%	0.025939	0.0421900	0.0435431	0.006557	0.0056860	0.0229456
90%	0.0375212	0.0250770	0.048775	0.0082801	0.0086809	0.05209

cordingly, memory consumption is reduced by roughly 3.3 times after an increase in 10% of false positive rate. In addition, as the false positive rate increases the number of hash functions, for each Bloom filter is reduced from 4 (10%) to 2 (50%) hash functions resulting in a slight increase of speed up at the training and classification phases.

Table 11 (binary classification datasets), Table 12 (multiclass classification datasets) and Table 13 (multiclass classification datasets) present the standard deviation of the accuracy related to different false positive probability configurations in Bloom WiSARD related to the accuracy results in Figure 9.

6. Related Work

Weightless neural networks have been studied for more than six decades [30]. A significant body of work has tackled the memory and computational efficiency of weighted [31] and weightless neural networks [32].

The memory required by the Virtual Generalising RAM weightless neural model (VG-RAM) [32, 33], for instance, is bounded by the size of the training

Table 13: Standard deviation of accuracy when varying the false positive rate (FPP) of Bloom WiSARD for multiclass classification problems (cont.).

FPP	**Segment**	**Shuttle**	**Vehicle**	**Vowel**	**Wine**
10%	0.0080506	0.012279	0.023848	0.0262235	0.0260741
20%	0.0079239	0.012454	0.0267349	0.0165499	0.0255359
30%	0.0081353	0.011607	0.024524	0.0160635	0.0335307
40%	0.0092052	0.0107775	0.0237721	0.0173125	0.0308558
50%	0.0077433	0.023087	0.0242694	0.0136733	0.0226538
60%	0.0079103	0.0109617	0.0317173	0.0185388	0.0235112
70%	0.013266	0.0152805	0.0220679	0.0187756	0.0473975
80%	0.0126708	0.0086389	0.0319956	0.0227475	0.0583274
90%	0.0199910	0.0102771	0.0231014	0.0261485	0.0361325

set. Input/output pairs presented during training phase are kept in memory. In the test phase, the memory of VG-RAM neurons is searched associatively by comparing the input presented to the network against all inputs in the learned input/output pairs. The output of each VG-RAM neuron is taken from the pair whose input is nearest to the input presented.

A number of recently proposed methods such as Bitwise Neural Networks [34], XNOR-Net [35], binarized neural networks [36] and ternary neural networks [37] leverage the binarization of the input or of the neural network weights to improve efficiency. The training phase considered in those models is similar in spirit to that of WiSARD (and Bloom WiSARD), as they are all memory-oriented approaches. Nonetheless, whereas WiSARD (and Bloom WiSARD) is intrinsically weightless, which renders it a natural choices to extend Bloom filters, the relationship between [34, 35, 36, 37] and Bloom filters is not as straightforward, and is left as subject for future work.

7. Conclusion

WiSARD is a powerful WNN model based on RAM memory that can be easily implemented in hardware and real-time systems. Nevertheless, certain applications require a considerable amount of memory to achieve good learning capabilities becoming impracticable to implement it in current technology. Alternative structures like dictionaries are required to implement the RAM nodes and turn feasible the use of the model.

In this work we propose the Bloom WiSARD model which extends WiSARD by implementing RAM nodes as Bloom filters. By using Bloom filters, memory resources are significantly reduced and for pattern recognition purposes we experimentally found that Bloom filters can build robustness into the system. Our experiments show that the model provides good accuracy and requires low training and testing times. In addition, it consumes up to 6 orders of magnitude less resources than standard WiSARD and about 7.7 times less resources than WiSARD implemented with dictionaries. In addition, increasing the false positive rate of Bloom WiSARD 50% results in 3.3 times less memory and average of 1.77% decreased accuracy compared against a false positive rate of 10% configuration.

This work opens up a number of avenues for future research. Future work will focus on leveraging extended Bloom filter operations such as the Bloom filter false free zone [38] or frequency counts of elements stored [39, 40], in order to enable Bloom WiSARD to use improved techniques such as DRASiW [41]. In this work, we focused on the use of Bloom filters for the implementation of discriminators, i.e., two-class classifiers. Bloom filters have been extended to allow for more than two output classes [42], and those extensions may be instrumental in the design of general purpose multi-class classifiers. More broadly, we envision that this work is one step further towards the use of Bloom filters for machine learning purposes [6, 14].

Acknowledgment

The authors would like to thank CAPES, COPPETEC, CNPq and FAPERJ for the financial support to this work.

References

[1] I. Aleksander, M. D. Gregorio, F. M. G. França, P. M. V. Lima, H. Morton, A brief introduction to weightless neural systems, in: ESANN 2009, 17th European Symposium on Artificial Neural Networks, 2009. URL https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2009-6.pdf

[2] I. Aleksander, W. Thomas, P. Bowden, Wisard·a radical step forward in image recognition, Sensor Review 4 (3) (1984) 120–124. arXiv:https://doi.org/10.1108/eb007637, doi:10.1108/eb007637. URL https://doi.org/10.1108/eb007637

[3] R. Mitchell, J. Bishop, P. Minchinton, Optimising memory usage in n-tuple neural networks, Mathematics and Computers in Simulation 40 (5) (1996) 549 – 563. doi:https://doi.org/10.1016/0378-4754(95)00006-2. URL http://www.sciencedirect.com/science/article/pii/0378475495000062

[4] D. S. Carvalho, H. C. Carneiro, F. M. França, P. M. Lima, B-bleaching: Agile overtraining avoidance in the wisard weightless neural classifier., in: ESANN, 2013.

[5] B. H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422–426. doi:10.1145/362686.362692. URL http://doi.acm.org/10.1145/362686.362692

[6] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, X. Luo, Optimizing Bloom filter: Challenges, solutions, and comparisons, IEEE Communications Surveys and Tutorials.

[7] M. Breternitz, G. H. Loh, B. Black, J. Rupley, P. G. Sassone, W. Attrot, Y. Wu, A segmented bloom filter algorithm for efficient predictors, in: 2008 20th International Symposium on Computer Architecture and High Performance Computing, IEEE, 2008, pp. 123–130.

[8] P. C. Dillinger, P. Manolios, Bloom filters in probabilistic verification, in: A. J. Hu, A. K. Martin (Eds.), Formal Methods in Computer-Aided Design, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 367–381.

[9] P. Sterne, Efficient and robust associative memory from a generalized bloom filter, Biological Cybernetics 106 (4) (2012) 271–281. `doi:10.1007/s00422-012-0494-6`.
URL `https://doi.org/10.1007/s00422-012-0494-6`

[10] M. Mitzenmacher, A model for learned Bloom filters and optimizing by sandwiching, in: Annual Conference on Neural Information Processing Systems (NeurIPS), 2018.

[11] M. Mitzenmacher, A model for learned Bloom filters and related structures, arXiv preprint arXiv:1802.00884.

[12] M. Hearn, M. Corallo, Connection bloom filtering, Bitcoin Improvement Proposal 37.

[13] J. W. Rae, S. Bartunov, T. P. Lillicrap, Meta-learning neural bloom filters, in: International Conference on Machine Learning (ICML), 2019, arXiv preprint arXiv:1906.04304.

[14] M. M. Cisse, N. Usunier, T. Artieres, P. Gallinari, Robust Bloom filters for large multilabel classification tasks, in: Advances in Neural Information Processing Systems, 2013, pp. 1851–1859.

[15] A. Odena, I. Goodfellow, Tensorfuzz: Debugging neural networks with coverage-guided fuzzing, arXiv preprint arXiv:1807.10875.

[16] S. Dasgupta, T. C. Sheehan, C. F. Stevens, S. Navlakha, A neural data structure for novelty detection, Proceedings of the National Academy of Sciences 115 (51) (2018) 13093–13098.

[17] R. Bogacz, M. W. Brown, Comparison of computational models of familiarity discrimination in the perirhinal cortex, Hippocampus 13 (4) (2003) 494–524.

[18] M. Mitzenmacher, E. Upfal, Probability and computing: randomization and probabilistic techniques in algorithms and data analysis, Cambridge university press, 2017.

[19] S. Chandar, S. Ahn, H. Larochelle, P. Vincent, G. Tesauro, Y. Bengio, Hierarchical Memory Networks, arXiv e-prints (2016) arXiv:1605.07427`arXiv:1605.07427`.

[20] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al., Hybrid computing using a neural network with dynamic external memory, Nature 538 (7626) (2016) 471.

[21] L. Carter, R. Floyd, J. Gill, G. Markowsky, M. Wegman, Exact and approximate membership testers, in: Proceedings of the tenth annual ACM symposium on Theory of computing, ACM, 1978, pp. 59–65.

[22] X. Wang, Y. Ji, Z. Dang, X. Zheng, B. Zhao, Improved weighted bloom filter and space lower bound analysis of algorithms for approximated membership querying, in: International Conference on Database Systems for Advanced Applications, Springer, 2015, pp. 346–362.

[23] A. Kirsch, M. Mitzenmacher, Distance-sensitive bloom filters, in: 2006 Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX), SIAM, 2006, pp. 41–50.

[24] A. Kirsch, M. Mitzenmacher, Less hashing, same performance: Building a better Bloom filter, in: Y. Azar, T. Erlebach (Eds.), Algorithms – ESA 2006, 2006.

[25] Wikipedia, Murmurhash fuction, `https://en.wikipedia.org/wiki/MurmurHash` (2019).

[26] L. Santiago, Bloomwisard implementation, `https://github.com/leandro-santiago/bloomwisard` (2019).

[27] Y. LeCun, The mnist database of handwritten digits, `http://yann.lecun.com/exdb/mnist/` (1998).

[28] G. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 42 (2) (2012) 513–529. `doi:10.1109/TSMCB.2011.2168604`.

[29] D. Dheeru, E. Karra Taniskidou, UCI machine learning repository (2017). URL `http://archive.ics.uci.edu/ml`

[30] W. W. Bledsoe, I. Browning, Pattern recognition and reading by machine, in: Papers of the Eastern Joint IRE-AIEE-ACM Computer Conference, 1959, pp. 225–232.

[31] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, S. W. Keckler, vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design, in: The 49th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Press, 2016, p. 18.

[32] J. Mrsic-Flogel, Aspects of planning with neural systems., Ph.D. thesis, Imperial College London (University of London) (1992).

[33] I. Aleksander, From wisard to magnus: A family of weightless virtual neural machines, in: RAM-Based Neural Networks, World Scientific, 1998, pp. 18–30.

[34] M. Kim, P. Smaragdis, Bitwise neural networks, arXiv preprint arXiv:1601.06071.

[35] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, Xnor-net: Imagenet classification using binary convolutional neural networks, in: European Conference on Computer Vision, Springer, 2016, pp. 525–542.

[36] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks, in: Advances in neural information processing systems, 2016, pp. 4107–4115.

[37] H. Alemdar, V. Leroy, A. Prost-Boucle, F. Pétrot, Ternary neural networks for resource-efficient ai applications, in: 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 2547–2554.

[38] S. Z. Kiss, É. Hosszu, J. Tapolcai, L. Rónyai, O. Rottenstreich, Bloom filter with a false positive free zone, in: IEEE INFOCOM, 2018.

[39] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, G. Varghese, An improved construction for counting bloom filters, in: European Symposium on Algorithms, Springer, 2006, pp. 684–695.

[40] O. Rottenstreich, Y. Kanizo, I. Keslassy, The variable-increment counting Bloom filter, IEEE/ACM Transactions on Networking (TON) 22 (4) (2014) 1092–1105.

[41] M. De Gregorio, M. Giordano, Cloning DRASiW systems via memory transfer, Neurocomputing 192 (2016) 115–127.

[42] B. Chazelle, J. Kilian, R. Rubinfeld, A. Tal, The bloomier filter: an efficient data structure for static support lookup tables, in: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2004, pp. 30–39.