Finite Precision Analysis of Support Vector Machine Classification in Logarithmic Number Systems

Faisal M. Khan, Mark G. Arnold and William M. Pottenger
Computer Science and Engineering, Lehigh University, Bethlehem, Pennsylvania
{fmk2,maab,billp}@lehigh.edu

Abstract

In this paper we present an analysis of the minimal hardware precision required to implement Support Vector Machine (SVM) classification within a Logarithmic Number System architecture. Support Vector Machines are fast emerging as a powerful machine-learning tool for pattern recognition, decision-making and classification. Logarithmic Number Systems (LNS) utilize the property of logarithmic compression for numerical operations. Within the logarithmic domain, multiplication and division can be treated simply as addition or subtraction. Hardware computation of these operations is significantly faster with reduced complexity. Leveraging the inherent properties of LNS, we are able to achieve significant savings over double-precision floating point in an implementation of a SVM classification algorithm.

1. Introduction

Cognitive systems capable of gathering information, detecting significant events, making decisions and/or coordinating operations are of immense value to a wide variety of application domains, from biomedical devices to automated military units. The core functionality of such machine learning and classification involves mathematical kernels employing commonly used operators [13].

Thus far, the driving thrust of progress has been in software-based solutions executing on general-purpose single or multi-processor machines. Aside from a plethora of work in neural-network hardware implementations [7], there exists a noteworthy absence of hardware-based machine-learning technologies.

This paper describes preliminary research towards the development of robust, hardware-based kernel solutions beyond neural networks for application-specific deployment. Specifically, we are employing Support Vector Machines (SVMs), a representative kernel-based machine-learning technique especially suited to high-dimensional data [13], [19], [20], [24].

As noted, significant progress has been made in the software domain for modeling and replicating the natural processes of learning, adapting and decision making for intelligent data analysis. Unfortunately, such solutions require significant resources for execution and may consequently be unsuitable for portable applications. Efficient hardware implementations of machine-learning techniques yield a variety of advantages over software solutions. Equipment cost and complexity are reduced. Processing speed, reliability and battery life are increased. The availability of application-specific hardware components for detecting events, decision-making, etc further enhance efficiency.

For these reasons we leverage logarithmic arithmetic for its energy-efficient properties [5], [4], [21]. Successful deployment of logarithmic functionality in neural networks has been shown to increase reliability and reduce power usage [3], [2]. We anticipate further progress in kernel-based SVMs since the majority of machine-learning kernels employ multiplication and/or exponentiation operators, the performance of which logarithmic computation significantly improves.

The primary task in this endeavor is to analyze the precision requirements for performing SVM classification in LNS hardware and compare them against the cost of using traditional floating-point architectures. Furthermore, comparison with neural-network precision demands and existing hardware SVMs also provides an excellent framework for analysis.

In the following sections we review SVM and LNS backgrounds along with related work in hardware-based machine-learning/decision making. We present our approach for analyzing LNS SVM classification and the results of the study. We follow with a conclusion and a discussion of the future work currently underway.

2. Support Vector Machines

The Support Vector Machine (SVM) algorithm is well grounded in statistical learning theory [23] but is abstractly a simple, intuitively clear algorithm [12]. It performs excellently for complex real-world problems that may be difficult to analyze theoretically.

SVMs are an extension of linear models that are capable of nonlinear classification. Linear models are incapable of representing a concept with nonlinear boundaries between classes. SVMs employ linear models to represent nonlinear class boundaries by transforming the input, or *instance space*, into a new space using a nonlinear mapping.

This transformation is facilitated through the use of kernels. The SVM algorithm can be treated linearly within the instance space, whereas the choice of various kernels may map the core operations transparently to a higher dimensional space. Consequently, complex pattern recognition and classification approaches can abstractly be represented linearly.

Following this transformation, a Maximum Margin Hyperplane (MMH) that separates the instances by class is learned, thereby forming a decision boundary. The MMH comes no closer to a given instance than it must; in the ideal case it optimally separates classes. *Support vectors* are the instances closest to the MMH. A set of support vectors thus defines the decision boundary for a given set of instances. This simplifies the representation of the decision boundary since other training instances can be disregarded.

SVM training involves minimizing a combination of training error (empirical risk) and the probability of incorrectly classifying unknown data (structural risk), controlled by a single regularization parameter *C* [11]. In the dual form (often preferred for training) this translates to obtaining the coefficients α_i through a quadratic programming problem. Given a set of input instance vectors $\vec{X}$ with class values Y, the objective is to minimize and maximize the following objective function given certain constraints:

$$\max_{b} \min_{0 \le a_i \le C} \vdots H =$$

$$\frac{1}{2}\sum_{i,j} a_i a_j Y_i Y_j K(\vec{X}_i, \vec{X}_j) - \sum_i a_i + b\sum_i Y_i a_i$$

Instances with an $\alpha > 0$ are considered support vectors. The variable *b* is a threshold value which is also computed.

Support Vector classification (in a simple two class problem) simply looks at the sign of a decision function. A test instance $\vec{T}$ is classified by the following decision function [19], [20], [24], [6], [11]:

$$f(\vec{T}) = sign(\sum_i a_i Y_i K(\vec{T}, \vec{X}_i) + b).$$

The choice of the kernel function $K(\vec{X}_i, \vec{X}_j)$ and the resultant feature space is crucially interesting in theoretical and practical terms. It determines the functional form of the support vectors given the regularization parameter *C*; thus, different kernels behave differently. Some common kernels are:

Linear: $K(\vec{X}, \vec{Y}) = (\vec{X} \bullet \vec{Y})$

Polynomial: $K(\vec{X}, \vec{Y}) = (\vec{X} \bullet \vec{Y})^d$

Radial Basis Function (RBF):

$$K(\vec{X}, \vec{Y}) = \exp(-\| \vec{X} - \vec{Y} \|^2 / (2s^2))$$

Sigmoid: $K(\vec{X}, \vec{Y}) = \tanh(\mathrm{K}(\vec{X} \bullet \vec{Y}) + \Theta)$

Interestingly a SVM with an RBF kernel is a simple type of neural network called a radial basis function network, and a sigmoid kernel implements a multilayer perceptron with no hidden layers [24].

Other machine-learning techniques, such as instance-based learning, distance-function learning, etc., leverage similar mathematical kernels using dot products, inner products (employed in image processing) [9], and other formulas. The fundamental operators employed in such kernels are multiplication, division, addition, subtraction, exponentiation, various roots and integration [19], [20], [24], [6], [11].

3. Hardware-based Machine Learning/Data Processing

There exists a significant lack of hardware-based machine-learning systems. With the aforementioned exception of neural networks (e.g., [3], [2], [14], [7], [18], [22]), the advantages of portable, dedicated machine-learning ASICs still remain a viable field to be explored.

Mak et al. [17] present an early attempt in hardware-based pattern matching for information retrieval. Their system is composed of two elements: Data Parallel Pattern Matching Engines (DPPMEs) that are slaves to a unique, master Processing Element (PE). Each DPPME is responsible for locating one pattern within a body of data. When a (complex) query is proposed, the PE decomposes it into basic match primitives, and distributes them among the various DPPMEs, each of which search for one specific pattern from the query, in parallel. Upon conclusion, the PE correlates the generated distributed results in order to actually resolve the query.

Leong and Jabri [16] present a low-power chip for classifying cardiac arrhythmia. The system employs a hybrid decision-tree/neural-network solution in order to classify a large database of arrhythmias with an accuracy of 98.4%. A neural network is employed in order to identify the abnormal heartbeat morphologies associated with arrhythmia, and a decision tree is utilized for analyzing heartbeat timing. The classifier system was designed for use in Implantable Cardioverter Defibrillators (ICDs)—devices that "monitor the heart and deliver electrical shock therapy in the event of a life threatening arrhythmia" [16]. Due to the standard five-year battery life in an ICD, it is imperative for the classifier to operate with extremely low-power consumption; their solution consumes less than 25nWatts.

The Kerneltron [10], [11] developed at John Hopkins is a recent SVM classification module. The internally analog, externally digital computational structure employs a massively parallel kernel computation structure. It implements the linear and RBF kernels. Due to the internal analog computation, the system is able to achieve a system precision resolution of 8 bits.

Anguita et al. [1] present a recent endeavor in the field. They propose the design of a fully digital architecture for SVM training and classification employing the linear and RBF kernels. The result is a highly optimal SVM ideal for hardware synthesis. The minimal word size they are able to achieve is 20 bits.

4. Logarithmic Number Systems

We leverage logarithmic arithmetic due to its high degree of suitability for machine-learning-kernel operations. Based on the once ubiquitous engineer's slide rule [4] Logarithmic Number Systems (LNS) are an alternative to fixed- and floating-point arithmetic. LNS utilize the property of logarithmic compression for numerical operations. Within the logarithmic domain, multiplication and division can be treated simply as addition or subtraction. Hardware computation of these operations is significantly faster with reduced complexity. Employing LNS involves an overhead of conversion to and from the logarithmic domain that is insignificant relative to the reduction in kernel computational complexity [4], [21].

Unlike Floating-Point (FP) systems, the relative error of LNS is constant and LNS can often achieve equivalent signal-to-noise ratio with fewer bits of precision relative to conventional FP architectures [4]. Similar to FP architectures, LNS implementations can represent numbers with relative precision; numbers closer to zero such as those used in SVMs [8], are represented with better precision in LNS than FP systems.

LNS provide other benefits conducive to a low-power, reliable application. The logarithmic conversion is inherently a compression algorithm as well. LNS are particularly cost effective when an application performs acceptably with reduced precision. Given successful analog implementations of SVMs [9], [10], we suspected digital low-precision LNS SVMs would be feasible. Such reduced precision permits a diminished word size. In turn, this offers lower power-consumption, and/or additional bits available for error-correcting codes. Furthermore, in CMOS technology, power is consumed when individual bits switch. Conventional multiplication involves extensive computation and bit switching. In LNS, since multiplication is a simple addition, the number of bits and the frequency of their switching are significantly reduced [5].

A disadvantage of LNS is that more hardware is required for addition and subtraction than for multiplication and division. Addition and subtraction in LNS are handled through lookup tables, through signals such as $s(z)= \log(1+b^z)$ and $d(z) = \log|1-b^z|$, but it has been shown that this lookup often requires minimal hardware for systems that tolerate low precision [5]. Let $x=\log|X|$ and $y=\log|Y|$. LNS use $X+Y = Y(1+X/Y)$, thus $\log(|X|+|Y|) = y+s(x-y)$, and $\log(|X|-|Y|) = y+d(x-y)$. The function $s(z)$ is used for sums, and $d(z)$ is used for differences, depending on the signs of X and Y.

Neural-network implementations using LNS already exist [3], [2] that exploit properties of $s(z)$ and $d(z)$ to approximate a sigmoid related to the RBF- and sigmoid-SVM kernels. The mathematical nature of kernel-based operations, given the emphasis on multiplication and exponentiation operations, make LNS an attractive technology for SVMs.

4.1 LNS SVM Classification

SVM classification lends itself quite naturally to implementation in LNS (Figure 1, following page). Only the decision function mentioned in section 2 needs to be realized. Our proposed architecture would apply kernel operations to a test vector and stored support vectors. The mathematical operations would take place within an LNS-based ALU. The kernel results would be multiplied and summed. Finally, classification would simply depend upon the sign of the result.

5. Precision Analysis

Our approach to analyzing LNS precision demands commenced by implementing two versions of the SVM algorithm: an initial double-precision floating-point version to serve as a benchmark for conventional SVMs, and a second LNS version capable of executing with variable precision. Both implementation results were corroborated with existing software solutions [8], [24] to ensure the accuracy of the results.

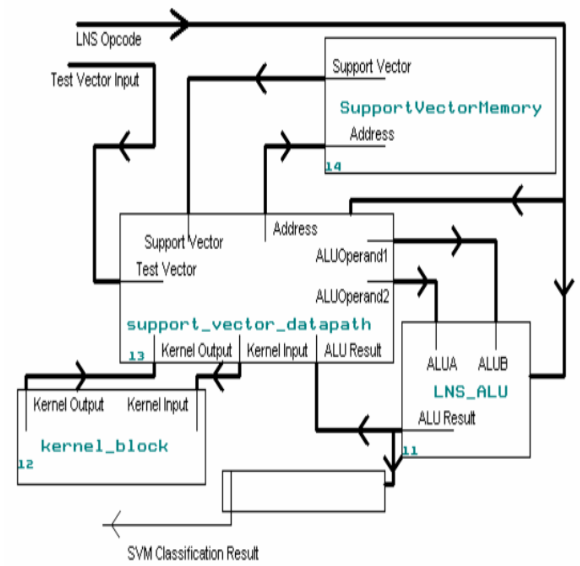

Figure 1: LNS SVM Classification

For classification analysis, we employed three different datasets commonly utilized for benchmarking purposes within the machine-learning community. The first dataset is used to classify diabetes based upon eight different attributes. The second dataset serves to classify members of the United States House of Representatives as Democrats or Republicans based on their voting record for sixteen bills. The third dataset is employed to classify SONAR signals as rocks or mines, and is employed to compare results with [1]. The diverse properties and natures of machine learning datasets are well represented within these three choices.

The datasets were scaled and normalized to prevent any single attribute from dominating the learning process [8]. Empirical results confirmed better performance of scaled data. Furthermore, scaled numbers centered on zero are better represented within the context of LNS precision [3].

The double-precision floating-point SVM was employed to generate conventional mathematical architecture results. Each dataset was processed through four SVM kernels: linear, Radial Basis (RBF_1 with $2s^2 = 1$ and RBF_2 with $2s^2 = 2$) and the sigmoid (? =0.1, T=0). Finally the LNS precision in the SVM algorithm was varied to ascertain optimal LNS precision with performance comparable to double-precision floating-point.

6. Experimental Results

Table 1 summarizes our analysis. For each dataset, it represents the LNS precision required for stabilized results equivalent to double-precision floating-point, and the LNS precision required for stabilized results within 1% of double-precision floating-point results.

Table 1. Summary of required LNS precision bits

	Kernel Type			
	Linear	RBF1	RBF2	Sigmoid
Diabetes Data Set				
Results equivalent to Floating Point	9	10	10	9
Results within 1% of Floating Point	4	7	7	7
Votes Data Set				
Results equivalent to Floating Point	3	2	6	8
Results within 1% of Floating Point	1	2	6	2
Sonar Data Set				
Results equivalent to Floating Point	8	8	7	14
Results within 1% of Floating Point	7	8	7	14

The LNS precision analysis summary indicates an architecture of 10 bits is virtually guaranteed to match the performance of a double-precision floating-point system. Furthermore, an architecture with an LNS precision of seven or eight bits yields results within 1% of double-precision floating-point. (Note that different kernels and datasets may lead to better performance.)

In the following discussion True Positives (TP) and True Negatives (TN) refer to test instances properly classified, similarly False Positives (FP) and False Negatives (FN) indicate test instances improperly classified. The percentage of Accuracy is calculated by:

$$\frac{TP + TN}{TP + TN + FP + FN} * 100$$

6.1 Diabetes Data set

The diabetes dataset consists of 512 training instances and 256 testing instances. Diabetes classification is a complex task with attributes representing different ranges of values; thus the SVM algorithm in LNS needed approximately 9 or 10 bits to stabilize. It begins to oscillate around the correct value at 7 bits, therefore 7 bits of LNS precision leads to results within 1% of double-precision floating point.

Table 2: Diabetes Linear Kernel

	Acc %	TP	TN	FP	FN
Floating Point	79.2969	37	166	7	46
LNS Bits					
1	73.4375	19	169	4	64
2	79.2969	43	160	13	40
3	77.3438	30	168	5	53
4	78.5156	38	163	10	45
5	79.2969	39	164	9	44
6	80.0781	38	167	6	45
7	79.2969	37	166	7	46
8	79.6875	38	166	7	45
9	79.2969	37	166	7	46
10	79.2969	37	166	7	46

Table 3: Diabetes RBF_1 Kernel

	Acc %	TP	TN	FP	FN
Floating Point	78.9063	35	167	6	48
LNS Bits					
1	62.8906	9	152	21	74
2	48.0469	38	85	88	45
3	70.7031	18	163	10	65
4	68.75	57	119	54	26
5	79.6875	61	143	30	22
6	76.1719	27	168	5	56
7	78.5156	33	168	5	50
8	79.2969	35	168	5	48
9	78.5156	34	167	6	49
10	78.9063	35	167	6	48

Table 4: Diabetes RBF_2 Kernel

	Acc %	TP	TN	FP	FN
Floating Point	80.4688	60	146	27	23
LNS Bits					
1	55.8594	5	138	35	78
2	40.625	75	29	144	8
3	62.1094	41	118	55	42
4	67.1875	64	108	65	19
5	66.4063	70	100	73	13
6	78.9063	53	149	24	30
7	79.6875	60	144	29	23
8	79.6875	61	143	30	22
9	80.0781	59	146	27	24
10	80.4688	60	146	27	23

Since the different kernels had approximately the same results, the best kernel for hardware implementation is the linear as that is the simplest in terms of hardware complexity.

6.2 Votes Data Set

The votes dataset consists of 290 training and 145 testing instances. Although it is defined by 16 attributes, they are all simple *yes* or *no* votes on bills. Since the linear kernel performed the most accurately, a LNS system of precision 2 or 3 would be sufficient for this classification and would save greatly on hardware complexity.

Table 5: Diabetes Sigmoid Kernel

	Acc %	TP	TN	FP	FN
Floating Point	79.2969	46	157	16	37
LNS Bits					
1	63.2813	10	152	21	73
2	67.5781	0	173	0	83
3	67.9688	2	172	1	81
4	67.1875	57	115	58	26
5	77.3438	62	136	37	21
6	76.5625	40	156	17	43
7	80.4688	46	160	13	37
8	80.0781	46	159	14	37
9	79.2969	45	158	15	38
10	79.2969	46	157	16	37

Table 6: Votes Linear Kernel

	Acc %	TP	TN	FP	FN
Floating Point	94.4828	81	56	1	7
LNS Bits					
1	93.7931	80	56	1	8
2	93.7931	80	56	1	8
3	94.4828	81	56	1	7
4	94.4828	81	56	1	7
5	94.4828	81	56	1	7
6	94.4828	81	56	1	7
7	94.4828	81	56	1	7
8	94.4828	81	56	1	7
9	94.4828	81	56	1	7
10	94.4828	81	56	1	7

Table 7: Votes RBF_1 Kernel

	Acc %	TP	TN	FP	FN
Floating Point	77.2414	88	24	33	0
LNS Bits					
1	75.1724	88	21	36	0
2	77.2414	88	24	33	0
3	77.2414	88	24	33	0
4	77.2414	88	24	33	0
5	77.2414	88	24	33	0
6	77.2414	88	24	33	0
7	77.2414	88	24	33	0
8	77.2414	88	24	33	0
9	77.2414	88	24	33	0
10	77.2414	88	24	33	0

Table 8: Votes RBF_2 Kernel

	Acc %	TP	TN	FP	FN
Floating Point	83.4483	88	33	24	0
LNS Bits					
1	81.3793	88	30	27	0
2	81.3793	88	30	27	0
3	82.7586	88	32	25	0
4	82.069	88	31	26	0
5	82.069	88	31	26	0
6	83.4483	88	33	24	0
7	83.4483	88	33	24	0
8	83.4483	88	33	24	0
9	83.4483	88	33	24	0
10	83.4483	88	33	24	0

Table 9: Votes Sigmoid Kernel

	Acc %	TP	TN	FP	FN
Floating Point	93.7931	80	56	1	8
LNS Bits					
1	62.069	75	15	42	13
2	94.4828	82	55	2	6
3	93.7931	80	56	1	8
4	95.1724	81	57	0	7
5	94.4828	80	57	0	8
6	94.4828	80	57	0	8
7	94.4828	80	57	0	8
8	93.7931	80	56	1	8
9	93.7931	80	56	1	8
10	93.7931	80	56	1	8

6.3 SONAR Data Set

The SONAR dataset is another complex set consisting of 104 training and 104 testing instances. The RBF_2 Kernel performs comparably to double-precision floating point and the results in [1]; it requires only 7 bits of precision. With an additional bit, a more accurate LNS architecture with 8 bits of precision could be leveraged via the RBF_1 Kernel.

Table 10: SONAR Linear Kernel

	Acc %	TP	TN	FP	FN
Floating Point	74.0385	41	36	20	7
LNS Bits					
1	54.8077	48	9	47	0
2	61.5385	46	18	38	2
3	67.3077	47	23	33	1
4	78.8462	49	39	17	5
5	73.0769	41	35	21	7
6	72.1154	41	34	22	7
7	75	42	36	20	6
8	74.0385	41	36	20	7
9	74.0385	41	36	20	7
10	74.0385	41	36	20	7

Table 11: SONAR RBF_1 Kernel

	Acc %	TP	TN	FP	FN
Floating Point	81.7308	33	52	4	15
LNS Bits					
1	61.5385	10	54	2	38
2	73.0769	27	49	7	21
3	78.8462	31	51	5	17
4	79.8077	34	49	7	14
5	79.8077	31	52	4	17
6	80.7692	33	51	5	15
7	80.7692	32	52	4	16
8	81.7308	33	52	4	15
9	81.7308	33	52	4	15
10	81.7308	33	52	4	15

Table 12: SONAR RBF_2 Kernel

	Acc %	TP	TN	FP	FN
Floating Point	73.0769	48	28	28	0
LNS Bits					
1	54.8077	19	38	18	29
2	73.0769	31	45	11	17
3	79.8077	33	50	6	15
4	72.1154	46	29	27	2
5	71.1538	48	26	30	0
6	71.1538	48	26	30	0
7	73.0769	48	28	28	0
8	73.0769	48	28	28	0
9	73.0769	48	28	28	0
10	73.0769	48	28	28	0

Table 13: SONAR Sigmoid Kernel

	Acc %	TP	TN	FP	FN
Floating Point	68.2692	16	55	1	32
LNS Bits					
1	53.8462	0	56	0	48
2	49.0385	0	51	5	48
3	47.1154	17	32	24	31
4	46.1538	48	0	56	0
5	64.4231	30	37	19	18
6	54.8077	12	45	11	36
7	60.5769	16	47	9	32
8	66.3462	17	52	4	31
9	65.3846	15	53	3	33
10	66.3462	16	53	3	32
11	68.2692	16	55	1	32
12	66.3462	14	55	1	34
13	67.3077	15	55	1	33
14	68.2692	16	55	1	32

6.4 Related Precision Analysis Work

This study of the LNS SVM classification precision requirements indicates that a general-purpose SVM needs seven or eight bits of precision to perform within 1% of double-precision floating point. Application-specific SVMs may require as little as two bits of precision. The actual LNS word size needs an additional six bits beyond the precision bits to represent the LNS exponent and sign bit, assuming a dynamic range of 2^{-16} to 2^{16}-1. In other words, the total LNS word required is between eight to fourteen bits.

For the SONAR dataset, the digital SVM in [1] requires at least a fixed-point word size of 20 bits, with a dynamic range of 2^{-9} to 2^{11}-1. In contrast, the LNS SVM proposed here requires only 11 bits for equivalent performance.

A related study on the precision requirements of neural-network hardware implementations [14] states that at least eight bits of precision are required for accurate performance with additional bits to cover the required dynamic range, as in [1]. The Kerneltron analog SVM [10] has a system resolution equivalent to eight digital precision bits again, with additional dynamic-range bits. The three- or four-bit LNS precisions which our simulations show are acceptable offer significant hardware savings.

7. Conclusion

We have presented a study of the precision requirements for novel SVM classification within a logarithmic hardware architecture. Leveraging the inherent properties of LNS, we are able to achieve significant savings over double-precision floating point. A general purpose SVM classification in LNS would require seven or eight bits of precision, whereas application-specific devices could be realized with as little as two bits of precision! Furthermore, we are able to achieve a precision comparable to that of an analog based implementation [10]. Additionally, despite the fact that SVM classification is significantly more complex than neural networks [14], we realize an equal or better precision through employing LNS. Moreover, we compare favorably with the only other work done in digital SVM hardware [1].

Logarithmic Number Systems represent an extremely attractive technology for realizing digital hardware implementations of SVMs and possibly other machine learning approaches. The precision requirement for LNS based SVM classification is eight or fewer bits, comparable to simpler digital neural networks or inherently optimal analog SVMs.

8. Future Work

This paper has described the first steps towards developing robust, kernel-based hardware machine-learning platforms employing logarithmic arithmetic. These platforms will serve as foundations for low-power machine-learning research, and for porting software solutions to hardware configurations.

LNS provide an innovatively exciting foundation due to inherently favorable characteristics for reduced precision and energy requirements.

We are currently implementing SVM classification in hardware to simulate and observe performance in terms of execution time and hardware costs. Furthermore, we are exploring precision requirements for hardware LNS-based SVM training. With the singular exception of the (non-LNS) recent work in [1], to the best of our knowledge no research into hardware-based training has been accomplished.

Future goals involve employing some of the innovative SVM training algorithms proposed in recent literature, employing an increased range of possible kernels, and expanding LNS hardware architectures to other machine-learning algorithms.

9. Acknowledgements

The authors would like to acknowledge Jie Ruan and Philip Garcia for their contributions and to thank the reviewers for their comments and suggestions for future work. Co-author William M. Pottenger gratefully acknowledges His Lord and Savior, Yeshua the Messiah (Jesus the Christ).

10. References

[1] Davide Anguita, Andrea Boni and Sandro Ridella. "A Digital Architecture for Support Vector Machines: Theory, Algorithm, and FPGA Implementation." *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 993-1009, Sept. 2003.

[2] Mark Arnold, Thomas Bailey, J. Cowles and Jerry Cupal. "Implementing Back Propagation Neural Nets with Logarithmic Arithmetic." *Proceedings International. AMSE Conference on Neural Networks*, San Diego, CA, May 29-31, (G. Mesnard and R. Swiniarski, Editors), vol. 1, pp. 75-86, 1991.

[3] Mark Arnold, Thomas Bailey, Jerry Cupal and Mark Winkel. "On the Cost Effectiveness of Logarithmic Arithmetic for Back Propagation Training on SIMD Processors." *International Conference on Neural Networks*, Houston, Texas, pp. 933-936, June 9-12, 1997.

[4] Mark Arnold. "Slide Rules for the 21st Century: Logarithmic Arithmetic as a High-speed, Low-cost, Low-power Alternative to Fixed Point Arithmetic." OSEE: *Second Online Symposium for Electronics Engineers*, 2001.

[5] Mark Arnold. "Reduced Power Consumption for MPEG Decoding with LNS." *Application Specific Architectures and Processors*, San Jose, pp. 65-75, July 17-19, 2002.

[6] Christopher Burges. “A Tutorial on Support Vector Machines for Pattern Recognition.” *Knowledge Discovery and Data Mining*, vol. 2, pp. 121-167, 1998.

[7] Gert Cauwenberghs, Editor. “Learning on Silicon: A Survey,” *Learning on Silicon: Adaptive VLSI Neural Systems*. Boston: Kluwer Academic Publishers, 1999.

[8] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: A Library for Support Vector Machines,* 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

[9] Roman Genov and Gert Cauwenberghs. “Stochastic Mixed-Signal VLSI Architecture for High-Dimensional Kernel Machines.” *Advances in Neural Information Processing Systems (NIPS'2001)*, Cambridge, MA: MIT Press, vol. 14, pp. 1099-1105, 2002.

[10] Roman Genov and Gert Cauwenberghs. “Kerneltron: Support Vector Machine in Silicon. ” *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1426-1434, 2003.

[11] Roman Genov, Shantanu Chakrabartty and Gert Cauwenberghs. “Silicon Support Vector Machine with On-Line Learning.” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 17, no. 3, pp. 385-404, 2003.

[12] Marti Hearst. “Support Vector Machines. ” *IEEE Intelligent Systems*, vol. 13, no. 4, pp. 18-28 July 1998.

[13] Ralf Herbich. *Learning Kernel Classifiers: Theory and Algorithms.* Cambridge: The MIT Press, 2002.

[14] Jordan Holt and Jeng-Neng Hwang. “Finite Precision Error Analysis of Neural Network Hardware Implementations. ” *IEEE Transactions on Computers.* vol. 42, no. 3, pp. 281-290, Mar 1993.

[15] Chih-Wei Hsu, Chih-Chung Chang and Chih-Jen Li. “A Practical Guide to Support Vector Classification,” available `www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`

[16] Philip Leong and Marwan Jabri, “A Low Power VLSI Arrhythmia Classifier.” *IEEE Transactions on Neural Networks*, vol. 6, no. 6, pp. 1435-1445, November 1995.

[17] Victo Mak, Kuo Chu Lee and Ophir Frieder. *Exploiting Parallelism in Pattern Matching: An Information Retrieval Application.* ACM Transactions on Information Systems (TOIS), pp. 52-72, 1991.

[18] Tony Martinez, Douglas Campbell, and Brent Hughes. “Priority ASOCS.” *Journal of Artificial Neural Networks*, vol. 1, no. 3, pp. 403-429, 1994.

[19] Bernhard Scholkopf, Alex Smola, and Klaus-Robert Muller. “Support Vector Methods in Learning and Feature Extraction.” *Australian Journal of Intelligent Information Processing Systems*, vol. 1, pp. 3-9, 1998.

[20] Alex Smola and Bernhard Scholkopf. “A Tutorial on Support Vector Regression.” *ESPRIT Working Group in Neural and Computational Learning II, NeuroCOLT2*, Technical Report TR-98-030, 1998.

[21] Thanos Stouraitis and Fred J. Taylor. “Analysis of Logarithmic Number System Processors.” *IEEE Transactions on Circuits and Systems*, vol. 35, no. 5, pp. 519-527, May 1998.

[22] Matthew Stout, George Rudolph, Tony Martinez, and Linton Salmon. “A VLSI Implementation of a Parallel, Self-Organizing Learning Model.” Proceedings of the *International Conference on Pattern Recognition,* vol. 3, pp. 373-376, 1994.

[23] Vladimir Vapnik. *The Nature of Statistical Learning Theory*, Springer Verlag, 1995.

[24] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. New York: Morgan Kaufmann Publishers, 2000.