Towards Continuous In-Situ Learning Using Online Semi-Supervised Support Vector Machines

Daniel Coombs
ECE598NS Course Project
University of Illinois at Urbana-Champaign
Email: dcoombs2@illinois.edu

***Abstract*—In this report a continuous in-situ learning system is described with the goal of implementation into custom hardware. The system is able to adapt its classification function online and on-chip by observing the incoming unlabeled data using techniques from semi-supervised learning literature. This approach achieves mixed results on synthetic datasets proving that this could be a promising area for future research. A high-level fixed-point architecture for the system is shown identifying each of the major building blocks that need to be studied for a full hardware implementation.**

I. Introduction

Machine learning (ML) has emerged in the past few decades as an extremely effective way to extract value from the massive amounts of data collected today. In a problem called classification, a system is given a vector of features corresponding to an observation. The system's goal is to classify that observation as belonging to a certain class in a discrete set of classes. Given a set of data upon which to run, a machine learning system can be trained to construct a function that accurately predicts the class of a given observation.

As we find more and more interesting applications for machine learning systems, it has become increasingly important to embedded these algorithms into our devices. As the Internet of Things (IoT) continues to grow, the demand for intelligent devices increases along with it. Embedding machine learning into IoT devices allows for data analytics to occur at the "edge", as opposed to the processing being offloaded to the cloud. However, machine learning systems tend to be very computationally intensive, and making them efficient enough to be deployed in the energy constrained world of edge computing has been an emerging research topic.

In [1] a machine learning accelerator is demonstrated that is able to achieve orders of magnitude energy reduction compared to a CPU based implementation when classifying medical signals. This study showed that in an energy constrained application, it is beneficial to use a custom accelerator to perform classification. Training of the system takes place off chip and the classifier's weights are sent to the SoC through a Bluetooth link. This system also includes an active learning (AL) accelerator which allows the system to adapt its classifier to a particular patient. The AL system works as follows. First, the SoC identifies a data sample that it believes could be used to improve the classifier. This sample is then transmitted over the Bluetooth link to a base station. A human expert is consulted to assign a label to the data. The sample and label are then added to the data set and the classifier is re-trained on the expanded set. Finally, the new classifier weights are transmitted to the SoC for use in further classification. The study shows that this process can improve the accuracy of the system by close to 5%.

There are, however, some flaws with this approach. First, it requires human intervention each time the system identifies a sample for active learning. Deployed at scale, this becomes an infeasible requirement if the number of experts able to accurately label the data is small. For this reason, the feedback loop for updating the classifier is long. It requires the active learning samples to occur frequently enough and the expert labeling to occur quickly. Second, the system requires a wireless link which consumes a large amount of energy. In an energy constrained scenario this could quickly consume the entire energy budget of the system. Last, the system does not take advantage of all of the information that is available to it. The system essentially throws out information every time it determines that a sample is not part of the active learning system.

In order to overcome these challenges, this report contains an idea for a continuous in-situ learning system. This system is able to update its classifier online and on-chip, using the streaming input data. Ideas are drawn from semi-supervised learning (SSL) techniques and prior implementations of machine learning kernels in silicon systems. The result is a practical system that could be deployed and autonomously adapt to its surroundings without need for any human intervention. The system allows the user to encode all prior information that is available at the time of deployment, then leave the system to improve its classification over time.

The report is structured as follows. Section II presents the requirements for a continuous in-situ learning system. Section III provides a review of semi-supervised learning methods. Section IV describes a mock-up of a continuous in-situ learning system that achieves the aforementioned requirements. Finally, Section V concludes the report.

II. Continuous In-Situ Learning System

A continuous in-situ learning system should be able to constantly improve its classification abilities of an input signal without the need for any human intervention. Imagine an implantable device similar to the one described in [1] but that

is able to improve its medical signal classification accuracy over time as its sees more samples without the need for a human expert to label extra data. There are many possible applications for these devices spanning medicine, agriculture, smart cities, etc.

Figure 1 contains a generic continuous in-situ learning system. The system consists of an analog front end and ADC for sensing and digital conversion, a machine learning kernel for classification, and an actuator to act upon the class information. The machine learning kernel itself consists of two high level blocks. The classification block takes the feature vector x_i and outputs its predicted class y_i. The model update block uses semi-supervised learning (SSL). It takes an unlabeled feature vector as input, and uses this information to update the classification block without a predetermined label.

The rest of this report will focus entirely on the machine learning kernel, its requirements, and the challenges that arise when attempting to build such a block. First, an important question to answer is "Why do we need continuous learning? Couldn't we train a classifier using labeled data and simply implement the feed forward classification on chip?" For an application where there is enough labeled data that is fully representative of the particular signal, this would be the best strategy. However, there are many systems that have limited labeled data or for which the labeled data collected for one system does not generalize to another. Medical signals are a good example of this. Due to privacy concerns and limited amounts of experts to label data, training data for supervised machine learning kernels is hard to obtain. If this data does exist, it sometimes does not generalize well to a particular patient due to biological reasons or biases in the medical sensors themselves. Because of this, a classifier that is trained solely on available data might not preform well on data collected in a new environment. A system that can take advantage of data collected while deployed to improve its classification is highly desirable.

This is the promise of semi-supervised learning. For problems in which there is a limited amount of labeled data, but an abundant amount of unlabeled data (which is how this system can be thought of after enough time passes), semi-supervised learning methods have been developed to improve classification, in certain situations, compared to when using the labeled data alone [2]. Section III will discuss semi-supervised learning in more depth.

Other than SSL, a requirement of a continuous in-situ learning system is that it can update its classifier as new samples are recorded. In literature, this is referred to as online learning (OL). Also, the system needs to do both of these things in an energy constrained environment. In order for the system to be justified, the update block needs to consume less power than a system where the training is done off-chip (requiring a wireless connection). To summarize, here are the 3 important requirements of a continuous in-situ learning system:

- Semi-Supervised Learning
- Online Learning
- Energy Efficient Implementation

III. Semi-Supervised Learning

Semi-supervised learning has been studied by machine learning researchers for the past few decades. Researchers believe that humans utilize SSL techniques as they learn, using labeled and unlabeled experiences to construct intelligence. In [2], Zhu and Goldberg explain many techniques that have been developed to solve this problem including mixture-models with expectation maximization, graph-based learning, and semi-supervised support vector machines (S3VMs). Due to the wide range of possible techniques, this report will focus entirely on S3VMs and how they can be applied to continuous in-situ learning.

A. Semi-Supervised Support Vector Machines

SVMs are widely applied machine learning systems that have shown good performance in many different applications. They can be thought of as maximum margin classifiers that find the optimal decision boundary between two classes [3]. S3VMs extend this theory to include unlabeled data along with the labeled data in traditional SVMs.

B. S3VM Cost Function

In [2], [4] the classical S3VM cost function is derived. (1) describes the optimization problem that allows unlabeled samples to influence the decision boundary. There are L labeled samples in X and U unlabeled samples in X^*. (2) determines the distance from the decision boundary and the sign of this function is taken as the binary classification label. (3) is the familiar hinge loss function that is widely used in SVMs.

$$\min_{w,b} \gamma||w||^2 + \sum_{i=1}^{L} l(f(x_i), y_i) + \lambda \sum_{i=1}^{U} l(f(x_i^*), sgn(f(x_i^*))) \tag{1}$$

$$f(x) = w^T x + b \tag{2}$$

$$l(f(x), y) = max(0, 1 - yf(x)) \tag{3}$$

Where (1) differs from the standard SVM primal equation is in the third term. This term adds the unsupervised samples to the optimization. Instead of using a label, like in the second term, the equation estimates the label of the sample the same way that it classifies samples, by taking the sign of (2). This S3VM optimization problem allows for the unlabeled samples to influence the decision boundary by requiring that they too are separated from the decision boundary by a maximum amount. This third term, however, is non-convex and therefore greatly complicates the process by which this optimization problem can be solved.

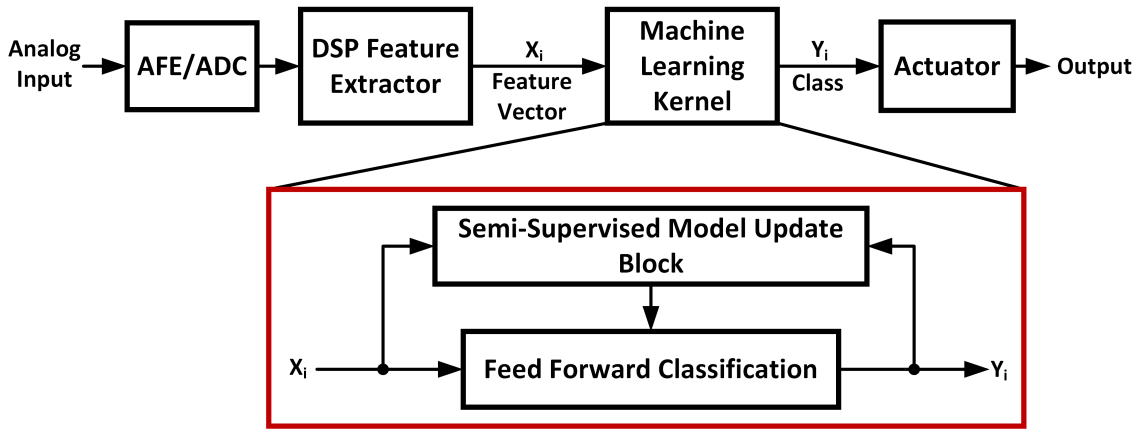


Fig. 1. Continuous In-Situ Learning System

C. S3VM Optimization Strategies

Because solving S3VM is a non-convex optimization problem, the bulk of the research in this field has been devoted to finding the best optimization technique. The solutions described in [5], [6], [7] contain large, complicated algorithms that are not suitable for implementation in a continuous learning system. Each of these techniques has an optimization step that requires all of the data to be available during training. If any new data becomes available, the entire process needs to be re-run and these algorithms have problems scaling past a few thousand unlabeled samples. This, combined with the computational complexity of these algorithms, make them unsuitable for embedded online learning. After a long search through S3VM literature, the approach in [4] was found to be suitable for online learning in an energy constrained setting.

IV. Online S3VM for Hardware Implementation

The approach by Karlen *et al.* in [4] is deemed "Large Scale Manifold Transduction." They modify the S3VM optimization problem to allow for an online approach that uses stochastic gradient decent (SGD) to minimize the cost function. Their cost function is shown in (4).

$$min\ \frac{1}{L}\sum_{i=1}^{L} l(f(x_i), y_i) + \frac{\lambda}{U^2}\sum_{i,j=1}^{U} W_{ij}\ l(f(x_i^*), y^*(\{i,j\})) \tag{4}$$

where

$$y^*(N) = sgn(\sum_{k\in N} f(x_k^*)) \tag{5}$$

These equations use a different approach to solving S3VM. The instead of taking the prediction for x_i as the label (as in (1)), equation 4 uses a k-nearest-neighbors (KNN) algorithm to find the closest samples to x_i. W_{ij} for these samples is set to 1 (else $W_{ij} = 0$) and the label $y*$ is calculated by summing up the distance of the KNN and taking the sign. This incorporates what is referred to as a *clustering assumption* into the standard S3VM equation. The clustering assumption states that samples in a certain class tend to be found closer to other samples in their class than in other classes. The authors claim this is superior to the standard S3VM approach. They show that this optimization problem can be solved using stochastic gradient decent and demonstrate state of the art semi-supervised learning results.

Due to its ability to be optimized using SGD and its proven SSL abilities, (4) was chosen to be the objective function for this implementation. The following approach is related to that described in [4] but contains some changes. Most of these changes were due to convergence issues with the approach in [4] and others were made in order to simplify for demonstration purposes. More work could be done by someone with sufficient ML algorithm experience in order to implement the full algorithm.

A. Algorithm

Algorithm 1 describes the continuous in-situ learning procedure using a variation of the approach in [4]. The first while loop can be performed off chip before the system is deployed, allowing the user to encode whatever prior information they have in the form of an initial classifier. This loop preforms standard SVM SGD on the labeled samples. Once the system is deployed, the second while loop allows the system to adapt to new data. The label for the new data is predicted using a majority vote of from the labels of the k-nearest-neighbor labeled samples. This predicted label is then used in a standard SVM SGD step to optimize the loss function. When not learning, Algorithm 2 is used to classify the incoming data using the weights found by Algorithm 1.

The stochastic gradient decent step is shown in (6) and is the same one derived in [3]. In practice, learning rate decay is used according to (7) from [8], [9] where η_0 is the initial learning rate and λ controls the rate of decay.

$$w_{n+1} = (1-\gamma_n\lambda)w_n + \gamma_n \begin{cases} 0 & \text{if } y_n {w_n}^t x_n > 1 \\ y_n x_n & \text{otherwise} \end{cases} \tag{6}$$

$$\gamma_n = \frac{\eta_0}{1+(\lambda\eta_0)n} \tag{7}$$

Algorithm 1 Continuous In-Situ S3VM Optimization

Input: labeled data (x_i, y_i) and unlabeled data x_i^*
  while $i <= Number\ of Labeled\ Samples$ **do**
    Pick a random labeled sample without replacement (x_i, y_i)
    Transform x_i using kernel approximation $\rightarrow x_{i,tran}$
    Make a gradient step to optimize $l(f(x_{i,tran}), y_i)$
  end while
  while there is new unlabeled sample x_i^* **do**
    Find KNN of x_i^* from labeled samples
    Predict y_i^* by majority vote of KNN labels
    Transform x_i^* using kernel approximation $\rightarrow x_{i,tran}^*$
    Make a gradient step to optimize $l(f(x_{i,tran}^*), y_i^*)$
  end while
Output: weights w and bias b

Algorithm 2 S3VM Classification

Input: sample x_i^*, optimized weights w and bias b
  Transform x_i^* using kernel approximation $\rightarrow x_{i,tran}^*$
  Evaluate $sgn(f(x_{i,tran}^*)) = sgn(w^T x_{i,tran}^* + b) \rightarrow y_i^*$
Output: Predicted class y_i^*

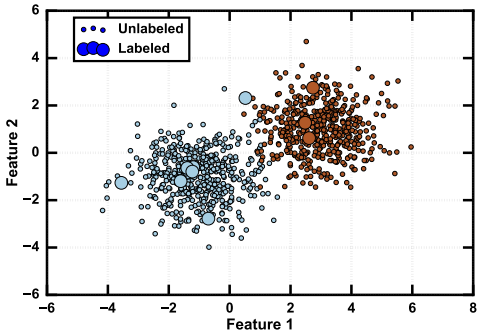
Because the function $f(x_i)$ is linear, this optimization technique can only learn linear decision boundaries. Therefore, a non-liner input mapping (NLIM) is required before the SGD step that maps the input into higher dimensions. This allows the SGD to learn a non-linear decision boundary. In conventional SVMs, the kernel trick is used to learn a non-linear boundary. In [10], Rahimi and Recht provide a method to use NLIM followed by a linear SVM to approximate an SVM trained with the radial basis function (RBF) kernel. This method uses an algorithm that maps each dimension of x_i to K dimensions. Larger K leads to a better approximation of the RBF kernel. Details of this procedure are deferred to their paper.

B. Performance on Synthetic Data

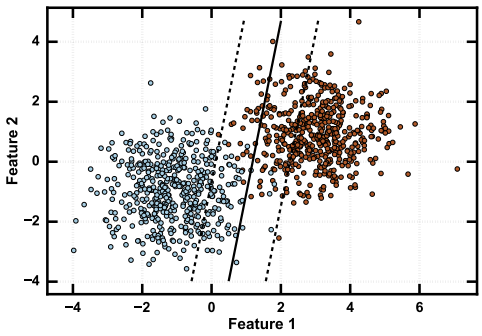
The above algorithm was implemented in Python with mixed results[1]. The following sections describe the performance of the code on synthetic data.

1) Gaussian Blobs: Figure 2 depicts the performance of the S3VM on a standard dataset consisting of 2 Gaussian blobs with different centers. Figure 2a shows the training data. 10 samples were chosen as labeled data points and 1000 samples were used as unlabeled data points. Comparing figure 2b to figure 2c shows that the S3VM can achieve better performance than the SVM on this data set under the right conditions. Careful observation of 2a shows a labeled sample in the blue class that causes the standard SVM to choose a sub-optimal decision boundary. The S3VM sees that same data point, but through the 1000 extra unlabeled sample SGD steps, it can learn a more optimal boundary and achieve 2% better accuracy. This case is an outlier, however, and if the labeled data is more representative of the whole dataset, SVM and S3VM achieve similar performance.

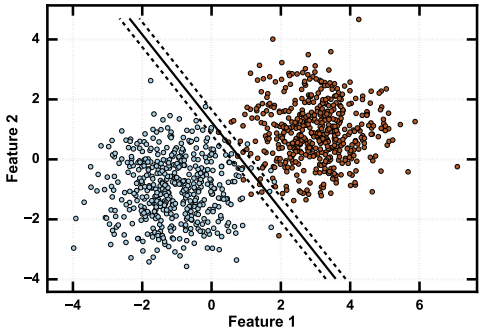
[1]Code can be found at https://github.com/dancoombs/s3vm-sgd



(a) Blobs Training Data



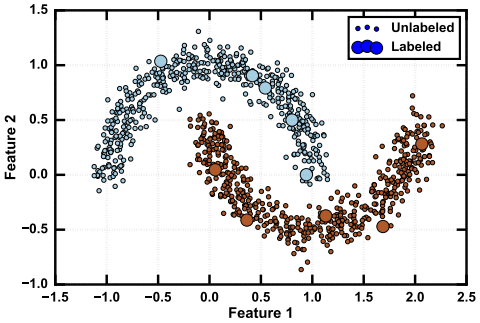
(b) SVM Test Data Results (96.5% Accurate)



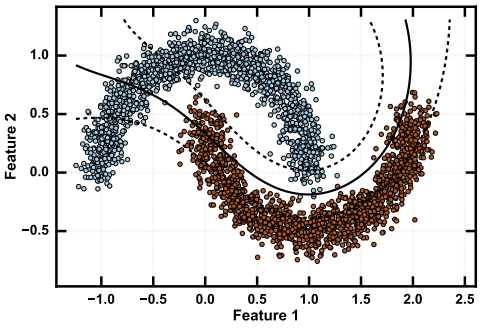
(c) S3VM Test Data Results (98.6% Accurate)

Fig. 2. S3VM Performance on the "Gaussian Blobs" synthetic dataset compared to standard SVM performance.

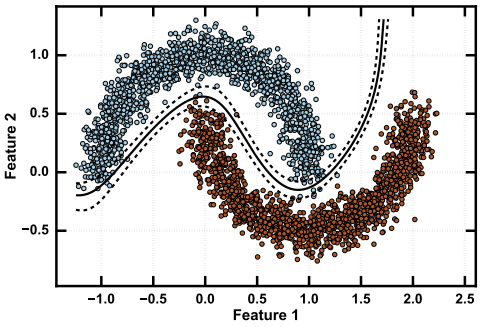
2) Moons Dataset: Figure 3 shows the performance of the system on the well-known "Moons" synthetic dataset. This dataset requires a non-linear decision boundary, so NLIM was used to approximate the RBF kernel for the S3VM. $K = 50$ was chosen, resulting in 100-dimensional data for the linear SGD. For this dataset, the S3VM greatly improved the classification accuracy compared to the standard SVM trained on just the labeled data. Figure 3a shows the training data with 10 labeled samples and 1000 unlabeled samples. Figure 3b shows the performance of the standard SVM was only able to achieve 83.7% accuracy on the test data as the decision boundary did not capture the full distribution of the data. In contrast the S3VM decision boundary in Figure 3c



(a) Moons Training Data

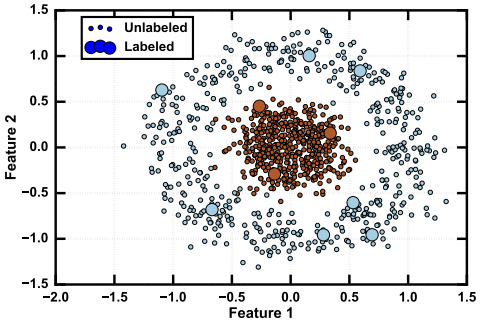


(b) SVM Test Data Results (83.7% Accurate)

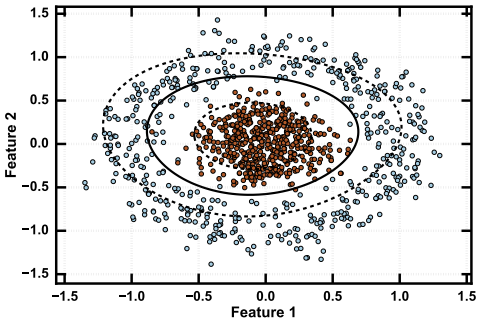


(c) S3VM Test Data Results (99.6% Accurate)

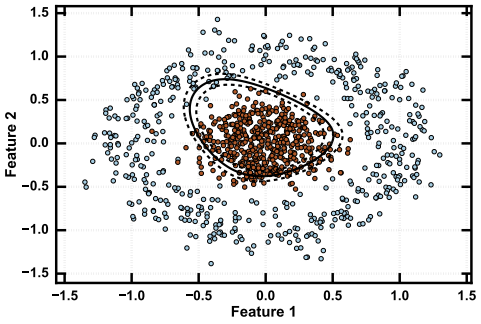
Fig. 3. S3VM Performance on the "Moons" synthetic dataset compared to standard SVM performance.



(a) Circles Training Data



(b) SVM Test Data Results (97.1% Accurate)



(c) S3VM Test Data Results (93.1% Accurate)

Fig. 4. S3VM Performance on the "Circles" synthetic dataset compared to standard SVM performance.

accurately fit the full distribution of the data and achieved a 99.6% classification accuracy on the test data.

3) Concentric Circles: Figure 4 contains the performance of the system on a concentric circles dataset. Again, this dataset requires a non-linear decision boundary and the RBF kernel approximation was used. Comparing figure 4b and 4c on this dataset, however, shows the S3VM achieving worse performance than the SVM. In this case, the labeled data represents the full data distribution well enough that the SVM can learn an accurate decision boundary.

C. Fixed Point Analysis

To reduce the energy consumption of the system when implementing in custom silicon, the arithmetic blocks can be built in a fixed point architecture. Moving from floating point to fixed point results in quantization of the signals within the machine learning kernel. This reduces the signal to noise ratio (SNR) of the classification and therefore reduces its accuracy. However, studies have shown that the amount bits needed to represent a signal can be significantly reduced before they have a noticeable effect on the system performance [1], [11].

A fixed point analysis for this algorithm was conducted empirically using quantization functions in the Python code. There are four main signals in the system that are available for quantization:

- Input feature vectors, x_i
- Transformed feature vectors, $x_{i,trans}$
- SVM weights, w
- Transformation weights, w_{trans}

Figure 5 contains a graph of the accuracy of the classifier as each of the major signals is quantized. Each signal is quantized independently (all other signals are still fixed-point), and the system is trained and evaluated on the Moons dataset. This

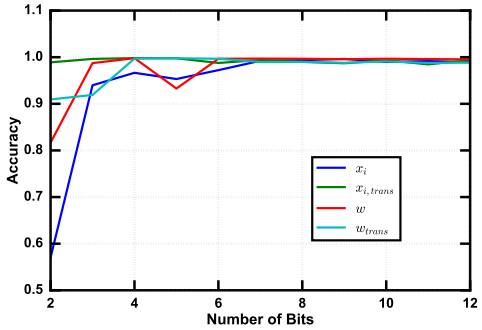


Fig. 5. Fixed Point Analysis of Continuous In-Situ S3VM

data shows that the system is most sensitive to input quantization, followed by quantization of the SVM weights. The system seems to preform well with a very small amount of bits assigned to the transformation weights and the output of the transformation. This could be due to the high dimensionality of the output. Overall, it was found that quantizing each signal to 9 bits has only a small effect on the performance. In fact, it still achieves 98.7% accuracy, a reduction of only 0.9% from the floating point system.

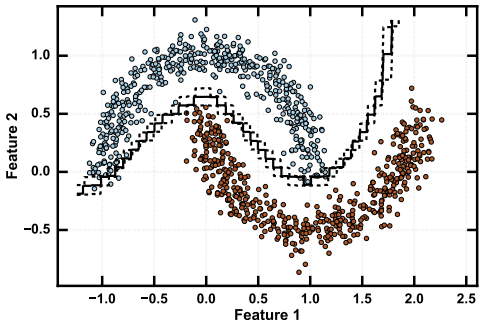
Figure 6a shows what happens to the decision boundary when the input feature vector is quantized to 6 bits. The decision boundary becomes much sharper, showing the quantization steps clearly. In contrast, when the transformation feature vector is quantized, as depicted in figure 6b, the decision boundary behaves much more erratically. However, even when quantized to 3 bits the boundary's shape does not change much due to the high dimensionality of the transformation.

D. Towards Hardware Implementation

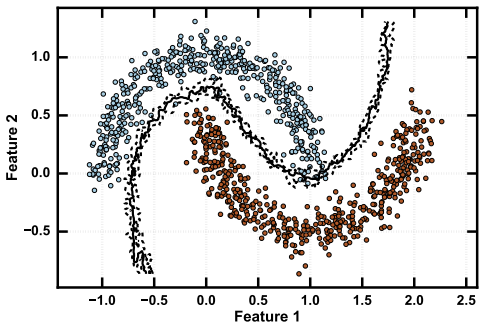
Figure 7 contains a mock-up of what this system could look like if implemented in a custom silicon accelerator. The major blocks are separated and will each be briefly discussed.

1) NLIM: In this system, the NLIM follows the procedure described in [10] called "Weighted Sum of Random Kitchen Sinks." Deferring to the paper on the details of how this approximates the RBF kernel, a few simple steps are taken. Weights are randomly chosen from a scaled normal distribution before training. In a hardware system, these weights can be precomputed and hard-coded. The input feature vector is projected onto the space described by these weights using a simple dot product. Then each element in the resulting matrix is then fed into a scaled exponential and the result is the transformed feature vector. A hardware implementation of this would need a group of multiply-accumulate blocks for the dot product and a CORDIC or a large LUT for the exponential.

2) K-Nearest-Neighbors: As the dimensionality of the input or the number of labeled samples is increased, finding the k-nearest-neighbors to a particular unlabeled sample becomes a computationally intensive task. Recently, accelerators have been developed to address this issue [12]. However, they have been more focused on throughput instead of energy



(a) S3VM Results on Moons Data with x_i quantized to 6 bits



(b) S3VM Results on Moons Data with $x_{i,tran}$ quantized to 3 bits

Fig. 6. S3VM Decision Boundaries with Fixed Point Signals

consumption. Future work could be done to optimize this block for an energy constrained system.

3) SVM SGD: The SVM SGD block is relatively simple and consists only of multiplication and addition. A simple architecture is shown in [3]. A detailed analysis of the precision requirements of the internal signals is done in [11].

V. Conclusion

The majority of this work took place in trying to find an appropriate algorithm that could apply to a continuous in-situ learning system. The algorithm found from [4] fit well, but a roadblock was hit when trying to create a custom implementation of it in Python. A simplified version of the algorithm was used that had mixed results when using simple, 2-dimensional data. Fixed point analysis of this simplified algorithm was also performed showing that quantizing the major signals to 9 bits had little effect on the classification performance. A hardware implementation was then outlined for this algorithm.

Future work in this domain should start on the algorithmic side. With more experience in machine learning algorithms, the approach from [4] could be studied in more depth as it applies to a hardware implementation. There are also a few other online S3VM algorithms that apply incremental/decremental learning [15], and might be better suited. Lastly, this report focused entirely on S3VMs. There are other semi-supervised approaches in literature, including a very interesting algorithm that applies boosting techniques [14], and future work needs

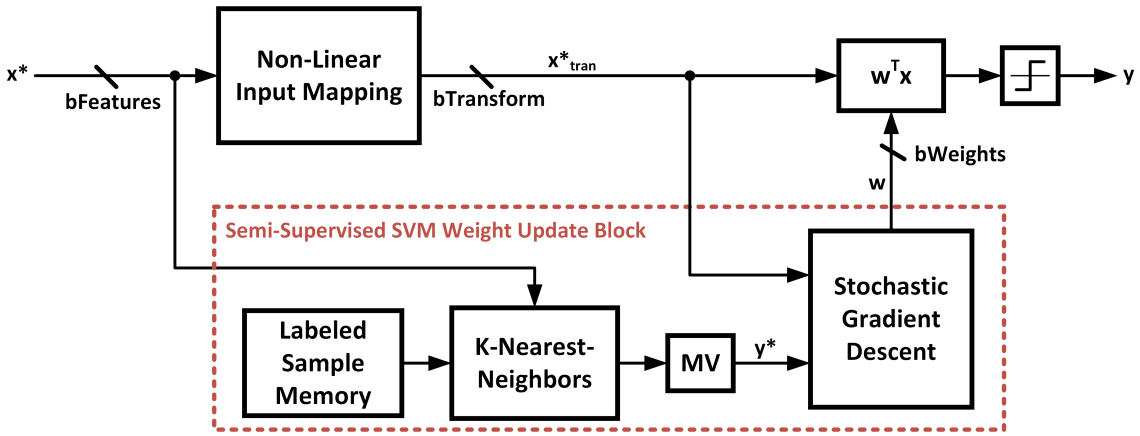


Fig. 7. Continuous In-Situ Learning System

to be done to see how suitable these approaches would be to a continuous in-situ learning system.

This work provided some initial insights into the building of a continuous in-situ learning system. Semi-supervised learning comes with its complexities, but the potential benefits of such are system are immense. With more care put into the algorithmic development, a continuous in-situ learning system could certainly be implemented in silicon.

REFERENCES

[1] K. H. Lee and N. Verma, "A Low-Power Processor With Configurable Embedded Machine-Learning Accelerators for High-Order and Adaptive Analysis of Medical-Sensor Signals," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 7, pp. 1625–1637, Jul. 2013.

[2] X. Zhu and A. Goldberg, *Introduction to Semi-Supervised Learning*. Morgan & Claypool Publishers, 2009.

[3] N. Shanbhag, "ECE598ns Course Notes," 2016. [Online]. Available: https://courses.engr.illinois.edu/ece598ns

[4] M. Karlen, J. Weston, A. Erkan, and R. Collobert, "Large scale manifold transduction," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 448–455. [Online]. Available: http://dl.acm.org/citation.cfm?id=1390213

[5] O. Chapelle and A. Zien, "Semi-Supervised Classification by Low Density Separation." in *AISTATS*, 2005, pp. 57–64. [Online]. Available: http://core.ac.uk/download/pdf/22017.pdf

[6] F. Gieseke, A. Airola, T. Pahikkala, and O. Kramer, "Fast and simple gradient-based optimization for semi-supervised support vector machines," *Neurocomputing*, vol. 123, pp. 23 – 32, 2014, contains Special issue articles: Advances in Pattern Recognition Applications and Methods. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231213003706

[7] O. Chapelle, V. Sindhwani, and S. S. Keerthi, "Optimization techniques for semi-supervised support vector machines," *Journal of Machine Learning Research*, vol. 9, no. Feb, pp. 203–233, 2008. [Online]. Available: http://www.jmlr.org/papers/v9/chapelle08a.html

[8] L. Bottou and Y. LeCun, "Large Scale Online Learning," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schlkopf, Eds. Cambridge, MA: MIT Press, 2004. [Online]. Available: http://leon.bottou.org/papers/bottou-lecun-2004

[9] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos: Primal estimated sub-GrAdient SOlver for SVM," in *Proc. 24th International Conference on Machine learning, 2007*. ACM, 2007, pp. 807–814.

[10] A. Rahimi and B. Recht, "Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning," in *Advances in neural information processing systems*, 2009, pp. 1313–1320. [Online]. Available: http://papers.nips.cc/paper/3495-weighted-sums-of-random-kitchen-sinks-replacing-minimization-with-randomization-in-learning

[11] C. Sakr, A. Patil, S. Zhang, Y. Kim, and N. Shanbhag, "Understanding the Energy and Precision Requirements for Online Learning," *arXiv preprint arXiv:1607.00669*, 2016.

[12] H. Kaul, M. A. Anders, S. K. Mathew, G. Chen, S. K. Satpathy, S. K. Hsu, A. Agarwal, and R. K. Krishnamurthy, "14.4 A 21.5m-query-vectors/s 3.37nj/vector reconfigurable k-nearest-neighbor accelerator with adaptive precision in 14nm tri-gate CMOS," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan. 2016, pp. 260–261.

[13] R. Mrzinger and G. Thallinger, "TRECVid 2008 High Level Feature Extraction Experiments at JOANNEUM RESEARCH." in *TRECVID*, 2007. [Online]. Available: http://www.academia.edu/download/3680391/10.1.1.159.3110.pdf

[14] P. K. Mallapragada, R. Jin, A. K. Jain, and Y. Liu, "SemiBoost: Boosting for Semi-Supervised Learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 11, pp. 2000–2014, Nov. 2009.

[15] C. P. Diehl and G. Cauwenberghs, "SVM incremental learning, adaptation and optimization," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 4, Jul. 2003, pp. 2685–2690 vol.4.