TECHNICAL UNIVERSITY OF CRETE

DIPLOMA THESIS

Feature Reduction for FPGA Based Implementation of Learning Classifiers

---

Author:
Konstantinos VOGIATZIS

Thesis Committee:
Prof. Aposolos DOLLAS
Prof. Michail G. LAGOUDAKIS
Prof. Ioannis PAPAEFSTATHIOU
(AUTH)

A thesis submitted in fulfillment of the requirements for the diploma of Electrical and Computer Engineer

in the

School of Electrical and Computer Engineering

February 12, 2024

Πολυτεχνείο Κρήτης

Διπλωματική Εργασία

Μείωση Γνωρισμάτων για Υλοποίηση σε Αναδιατασσόμενο Υλικό Ταξινομητών Μάθησης

Συγγραφέας:
Κωνσταντίνος Βογιατζής

Επιτροπή Διατριβής:
Καθ. Απόστολος Δόλλας
Καθ. Μιχαήλ Γ. Λαγουδάκης
Καθ. Ιωάννης Παπαευσταθίου
(ΑΠΘ)

Διπλωματική εργασία που υποβλήθηκε για την εκπλήρωση των απαιτήσεων για το δίπλωμα του Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών

στη

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

12 Φεβρουαρίου 2024

TECHNICAL UNIVERSITY OF CRETE

School of Electrical and Computer Engineering

Abstract

Feature Reduction for FPGA Based Implementation of Learning Classifiers

by Konstantinos VOGIATZIS

During recent years data sets have grown rapidly in size, mainly because they are collectively gathered by numerous consumer information-sensing internet of things (IoT) devices or services, such as mobile devices, software logs, cameras, wireless sensor networks, etc. Heterogeneous hardware, such as FPGAs, seem to be a promising alternative in terms of acceleration, even from GPUs, in complex machine learning problems. They still suffer though from low on-chip memory resources making scaling to high dimensionality tasks difficult, as input/output (I/O) traffic may dominate the overall latency. Due to such restrictions, FPGAs currently, are mostly used for the inference task and not the training one, as it usually requires fewer memory resources. In this work, we propose a general dimensionality reduction scheme for learning classifiers, operating both as training and inference accelerators which could be applied in low resource hardware devices, such as FPGAs. We achieve impressive improvements, with on-chip memory utilization during training reduced by $10\times$ to $32\times$ for online and batch learning, with around 5% loss in accuracy. We implement a pipelined hardware architecture, using a learning classifier coupled with a dimensionality reduction scheme implementing two different methods: Hash Kernel and Sparse Random Projection.

ΠΟΛΥΤΕΧΝΕΊΟ ΚΡΉΤΗΣ

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Περίληψη

Μείωση Γνωρισμάτων για Υλοποίηση σε Αναδιατασσόμενο Υλικό Ταξινομητών Μάθησης

Κωνσταντίνος Βογιατζής

Τα τελευταία χρόνια τα σύνολα δεδομένων έχουν αυξηθεί ραγδαία σε μέγεθος, κυρίως επειδή συλλέγονται μαζικά από πολυάριθμες συσκεύες για τους καταναλωτές στο διαδίκτυο των πραγμάτων ή υπηρεσιών, όπως κινητές συσκευές, αρχεία καταγραφής λογισμικού, κάμερες, ασύρματα δίκτυα αισθητήρων, κ.λπ. Ετερογενές υλικό, όπως η αναδιατασσόμενη λογική (Field Programmable Gate Arrays - FPGA) , φαίνεται να είναι μια πολλά υποσχόμενη εναλλακτική από άποψη επιτάχυνσης, ακόμη και από επεξεργαστή γραφικών Graphics Processing Unit (GPU), σε πολύπλοκα προβλήματα μηχανικής μάθησης. Όμως εξακολουθούν να υποφέρουν από χαμηλούς πόρους μνήμης στο ολοκληρωμένο κύκλωμα, καθιστώντας δύσκολη την κλιμάκωση σε εργασίες υψηλής διάστασης, καθώς το ινπυτ/ουτπυτ (I/O) μπορεί μπορεί να κυριαρχεί στη συνολική καθυστέρηση. Λόγω τέτοιων περιορισμών, οι FPGAs επί του παρόντος χρησιμοποιούνται κυρίως για την εξαγωγή συμπερασμάτων και όχι για την διαδικασία εκπαίδευσης, καθώς συνήθως αυτή απαιτεί λιγότερους πόρους μνήμης. Στην παρούσα διπλωματική εργασία προτείνουμε ένα γενικό σχήμα μείωσης διαστάσεων για ταξινομητές εκμάθησης που λειτουργούν με διπλό ρόλο ως επιταχυντές τόσο εκπαίδευσης όσο και συμπερασμάτων, και θα μπορούσαν να εφαρμοστούν σε συσκευές υλικού με λίγους πόρους, όπως οι FPGAs. Τα αποτελέσματα της παρούσας διπλωματικής εργασίας καταδεικνύουν εντυπωσιακές βελτιώσεις, με τη χρήση μνήμης στο ολοκληρωμένο κύκλωμα κατά τη διάρκεια της εκμάθησης, μειωμένη κατά 10× έως 32× για διαδικτυακή και μαζική εκμάθηση, με περίπου 5% απώλεια σε ακρίβεια. Υλοποιούμε μια αρχιτεκτονική υλικού με διοχέτευση pipelining χρησιμοποιώντας έναν ταξινομητή εκμάθησης σε συνδυασμό με ένα σχήμα μείωσης διαστάσεων που εφαρμόζει δύο διαφορετικές μεθόδους: πυρήνα κατακερματισμού και αραιή τυχαία προβολή.

Acknowledgements

I would like to pay my sincerest gratitude to Dr. Antonis Nikitakis for his invaluable guidance and unwavering support during the early stages of my research. Without his mentorship, the trajectory of this work would not have unfolded with the same depth and precision.

My heartfelt thanks also go to Dr. Ioannis Papaeustathiou for his contribution as my supervisor. I am truly appreciative of his mentorship, and it is through his initial acceptance and introduction to Dr. Nikitakis, that brought this academic pursuit to reality.

A special acknowledgment is reserved for Dr. Apostolos Dollas, my current supervisor, whose continuous support and guidance over the past year have been indispensable. His consistent encouragement, particularly during challenging moments, played a pivotal role in the successful completion of this thesis.

Last but not least I would like to thank Dr Michail Lagoudakis for been a valuable member of this committee and helping with his constructive feedback and corrections.

I am deeply grateful to my family and friends for their unwavering support and encouragement throughout this long journey. There were many ups and downs, as changes and life happens. Their love, understanding, and encouragement have been invaluable to me, especially the ones that believed in me, as it was a long journey. From a friendly phone call of support in a bad moment, to extending a helping hand when I needed it most, I just have to say a great thank you to all.

At last, I would like to say a special and big thank you to my partner, Abi, for all her support these last few months and the patience she has shown. Also, for her help with grammar and spell checking this work.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

High-dimensional datasets present many mathematical challenges, but also many opportunities, as they usually ought to give rise to new theoretical developments. The main problem with high-dimensional datasets is that, not all measured variables are important for understanding the underlying phenomena of interest or, in other cases, only a small percentage of them is actually even present in each observation (sparse datasets). Most statistical learning methods though, assign memory space to feature variables (i.e. through learning coefficients), even though they, most of the time, are not present in the observations. As a result, not only the computation complexity but also the memory footprint of such methods, could get out of control. In this work, we deal with the machine learning problem for low memory resource devices, such as FPGAs, in a more general context. The key motivation for our scheme is to design general purpose, efficient streaming accelerators for learning classifiers in a single architecture that reduces off-chip DRAM data access. Although the adopted methods have originated from database and big data retrieval/classification, we demonstrate that they could be adapted to perform equally well on smaller general machine learning problems in the context of FPGAs accelerators. The main idea in our proposed architecture is to maintain the learning problem in such low dimensionality in order to allow for the learning parameters to lie on-chip at all times, both during training and inference. As for the training data, since they cannot lie on-chip, we iterate them through DRAM across the training epochs. As we treat training data in batches, the I/O latency could be easily hidden using double buffering; but more importantly, as training data after the first pass through the accelerator are compressed by at least an order of magnitude, the I/O overhead is almost zero even without any buffering technique. We implement two different dimensionality reduction solutions: one utilizing Sparse Random Projections and one utilizing Hash Kernels. We compared

the effectiveness of both solutions in various datasets, when pipelined with an on-line learning classifier, forming a single unified learning scheme. The proposed scheme is a modular, pipelined architecture that could be applied in any gradient-based learning scheme. For our reference implementation, we use a logistic regression classifier, trained using a mini-batch gradient descent algorithm.

The contributions of this work are the following:

- We propose a novel hardware architecture, transparently coupling an on-chip trained classifier with a dimensionality reduction scheme.
- We implement two different dimensionality reduction schemes with very low memory footprint, in order to target efficiently a broad range of datasets.
- We achieve from $10x$ to $32x$ improvements in the on-chip memory utilization during training, for both data streaming (online) and batch learning problems.
- For batch learning problems, we further achieve from $10x$ to $20x$ speedup, as we reuse the dimensionality reduction operations for the subsequent training iterations.

Chapter 2

Related work

2.1 Motivation

Our motivation and idea came from [1] the task of predicting ad click-through rates (CTR) which is a significant challenge at a large scale. It also plays a crucial role in the multi-billion dollar online advertising sector. The data in reference is sparse, making it useful to identify suitable methods for feature selection that can reduce the number of features per data entry. Additionally, the demand for solving such problems has increased significantly and continues to grow with the expansion of the internet. In an industrial context, it is common to make predictions on billions of events per day using a large feature space and subsequently analyzing the resulting data. This leads us the the idea of an online algorithm that after making a prediction it takes feedback and retrains the model.

Although predicting click rates is a profitable task, it is important to keep the energy cost per prediction low since the earnings per click are not very high.

Our goal is to develop an embedded system on an FPGA that can handle a large scale Machine Learning problem with acceptable accuracy. To achieve this, we need to look for algorithms that run without requiring large amounts of memory.

This demand is divided in two categories. The first one has to do with the answer to the question "How much memory will our algorithm use to reduce the size of the initial sample?". The second is related to the length of features for the reduced data-set, as each feature will occupy a coefficient in our RAM.

In the next section, the pros and cons of the related work are outlined.

2.2 Feature Selection

Feature selection can be categorized and characterized in various ways. One approach is to consider whether the algorithms are lossless or not. Additionally, some algorithms may have high memory requirements, while others are more memory-efficient.

While the main purpose of feature selection is to choose relevant and informative features, it can also serve other objectives, including:

General data reduction: This involves limiting storage requirements and improving algorithm speed by reducing the overall amount of data.

Feature set reduction: By reducing the number of features, resources can be conserved in subsequent rounds of data collection or during utilization.

Performance improvement: Feature selection can lead to enhanced predictive accuracy by focusing on the most important features for the specific task at hand.

Data understanding: Feature selection allows for gaining insights into the underlying data generation process and can facilitate data visualization for better comprehension.

In the following sections the focus will be on a few lossless algorithms that have been examined in detail, and reasons will be outlined as to why they can not be used. Additionally, alternative algorithms will be explored that, while not lossless, present compelling advantages.

2.2.1 Decomposition Techniques

Some decomposition techniques algorithms were explored, like Singular Value Decomposition (SVD) [2] and Truncated SVD [3] or other similar algorithms like PCA [4]. Since it is lossless the accuracy was similar to the original dataset.

Singular Value Decomposition (SVD) is a compelling technique for dimensional reduction due to its ability to reveal the essential structure of a dataset by decomposing it into its constituent singular values and vectors. By retaining the most significant singular values, which capture the dominant patterns and variability in the data, one can achieve a reduced-dimensional representation that retains crucial information. SVD excels in capturing latent

relationships and patterns within high-dimensional datasets, making it particularly useful for tasks such as noise reduction, feature extraction, and compression. The resulting reduced-dimensional representation allows for more efficient storage, computation, and analysis of complex datasets while preserving the key characteristics that contribute to the overall understanding of the underlying data structure. The versatility of SVD in capturing intrinsic patterns makes it a valuable tool for practitioners in various fields seeking to manage the computational complexity of high-dimensional data without sacrificing critical information.

Principal Component Analysis (PCA) is another powerful technique for dimensional reduction with distinct advantages and applications. PCA identifies the principal components, which are linear combinations of the original features that capture the maximum variance in the data.

PCA is particularly effective in decorrelating features, reducing redundancy, and highlighting the dominant patterns within the data. This reduction in dimensionality not only facilitates more efficient storage and computation but also aids in visualizing and interpreting complex datasets. Additionally, PCA is widely employed in feature engineering, data preprocessing, and noise reduction, contributing to enhanced model performance in machine learning applications.

2.2.2 Feature Hashing

Hash kernel methods have several usages, among them been the conversion of non numeric data to numbers. In literature several innovative usages can be found like protein sequence classification [5], although the most common one is converting text data to numbers. Another usage is dimensional reduction. A very simplistic explanation about feature hashing would be the following. Having a sample of 100 features we want to reduce it to 20, so we randomly pick 5 of 100, we add them and create the 1st feature of a the new featured sample. Repeating this 20 times with attention not to ever pick the same feature of the initial sample we have a newly created sample with 20 features reducing the initial size 5 times. If the initial sample was sparse meaning that many of the 100 features where zeros then addition does not affect the result and in a way it was very similar to just remove these features. Also if the numeric values are of the features are similar it will be a linear addition without affecting the result. A simple example is that if the 100 features where just liquids in liters and the volumes where always between 0-10

then it will give better result than when one features is volumes in liter and the other is micro-grams.

Even though our main motivation paper [1] mentions the technique saying they did not got any significant improvement we thought that in our case even this improvement could be something usable. Also mainly [6] but also some others [7] [8] found this method useful.

While [7] talks about the importance of feature hashing focusing in text mining studies, particularly in the context of big data. It presents the problem for the scope of how it is used to reduce the size of feature vectors in text mining tasks, resulting in faster data mining processes such as classification, clustering, and association. The paper categorizes feature hashing approaches into two groups: those related to natural language processing algorithms that aim to extract more meaningful hash results, and mathematical hashing algorithms that focus solely on generating binary outputs without considering the context or meaning of the input text.

Then [8] discusses the effectiveness of feature hashing as a dimensionality reduction strategy and presents exponential tail bounds for feature hashing. The study demonstrates that the interaction between random subspaces in feature hashing is negligible with high probability. The paper also explores the application of feature hashing in multitask learning, specifically in scenarios with a large number of tasks. The paper introduces specialized hash functions with unbiased inner-products applicable to various kernel methods. Exponential tail bounds are provided to explain the strong empirical results of hashed feature vectors. Additionally, the study shows that independently hashed subspaces have negligible interference, making large-scale multitask learning feasible in a compressed space. The paper also presents collaborative email spam filtering as a novel application for hash representations and provides experimental results using real-world spam datasets. The result here was eye-catchy with the global-hashed curve for the spam catch-rate achieved by the global classifier following the utilization of the hashing function. When $m = 2^{26}$ (=67.108.864), the global-hashed curve mirrors the performance of the baseline classifier. The early convergence observed at $m = 2^{22}$ (=4.194.304) indicates that hash collisions have negligible impact on the classification error, indicating that the baseline classifier and the hashed method yield comparable outcomes.

A very commonly referred work is [6] introduces hashing as a means to enhance the efficiency of kernels. This approach extends prior research that employed sampling techniques, and presents a systematic method for computing the kernel matrix in scenarios involving data streams and sparse feature spaces. Additionally, they provide deviation bounds compared to the exact kernel matrix, offering practical applications for estimation tasks involving strings and graphs.

In their study, the researchers tested the effectiveness of their approach by applying hashing to various problems. First, they used it for classification on the Reuters RCV1 dataset, which has a large number of features. Then, they applied it to the DMOZ ontology of web page topics, which has a high number of topics. The third experiment focused on biochemistry and bio informatics graph classification, where their hashing scheme was used to compare graphs.

For the Reuters dataset, they used linear kernel SVM with stochastic gradient descent (SGD) as the main algorithm. They applied their hash kernels and random projection techniques to the SVM. They compared their approach to Bottou's SGD, Vowpal Wabbit, and random projections, and found that their hash kernel method performed well in terms of run-time and error rate.

In the experiments with DMOZ, they compared their hash kernel method with baseline methods such as uniform classifiers and majority voting. They also compared it to K Nearest Neighbor (KNN) and K-means algorithms. The hash kernel approach achieved lower misclassification rates and memory footprint compared to the baseline methods.

Overall, the results demonstrated the efficacy of the hashing approach for structured data classification tasks, providing faster processing times and competitive accuracy rates compared to other methods.

2.2.3 Random Projections

A more mathematical approach from feature hashing with better generic results is random projections. The Johnson-Lindenstrauss lemma is the mathematical basis for that. The Johnson-Lindenstrauss lemma states that if the data points lie in a very high-dimensional space, then projecting such points on simple random directions preserves their pairwise distances. Paraphrasing Achlioptas [9] a nice explanation of this technique is to imagine a 3D

sculpture being in a 2D paper sketch. You lose some information but still have an idea of what is represented.

The work of [9] the probabilistic method has played a crucial role in simplifying and refining Johnson and Lindenstrauss's original proof while introducing randomized algorithms for constructing embeddings. To address these challenges, there is a need for non-adaptive alternatives to projection methods like [4]. Their main result, presents a novel approach that simplifies the projection process while maintaining the quality of the embedding. This approach leverages simple probability distributions and can be efficiently implemented using standard SQL primitives in a database environment. Their main finding, as stated in a Theorem they provide, demonstrates that projections onto spherically random hyperplanes can be replaced with simpler and faster operations. These operations can be efficiently implemented using standard SQL primitives in a database environment. Surprisingly, this simplification does not compromise the quality of the embedding.

Upon closer examination of the embedding computation, we observe that each row (vector) of A is projected onto k random vectors with independent coordinates, characterized by random variables, having a mean of 0 and variance of 1. If these random variables were independent and normally distributed with a mean of 0 and variance of 1, the resulting vectors would point uniformly in random directions in space. Projections onto such vectors have been explored in various settings, including approximate nearest neighbors, learning intersections of half-spaces, and learning mixture of Gaussian models. Their proof reveals that the behavior of a fixed vector a projected onto a random vector is determined by the even moments of the random variable. The concept of randomization in JL-projections serves as a means to mitigate issues such as distance disparities caused by single dimensions. An initial random rotation applied to the original point set in Rd effectively corresponds to projecting onto a spherically random k-dimensional hyperplane by selecting the first k coordinates. Therefore, randomization in JL-projections acts as a precautionary measure against axis-alignment, akin to employing a random permutation before running Quick-sort.

On work of [10] they conducted comprehensive experiments to demonstrate the advantages of the learning of high-dimensional mixtures of Gaussians which seems to benefit from projecting the data into a randomly chosen low-dimensional subspace. Two main theoretical results regarding random projection are noteworthy. Firstly, data from a mixture of k Gaussians can

be projected into $O(\log(k))$ dimensions while preserving cluster separation. The projected dimension is independent of the number of data points and their original dimension. Secondly, random projection can make highly eccentric clusters more spherical, which is crucial when dealing with raw high-dimensional data that often exhibits eccentric clusters due to varying measurement units. Eccentric clusters pose challenges in algorithms like EM, which require restrictions to avoid singular or near-singular covariance matrices. These benefits have made random projection a key ingredient in the first polynomial-time, provably correct algorithm for learning mixtures of Gaussians. Random projection can also be easily combined with EM. In their experiments on synthetic data from Gaussian mixtures, EM with random projection consistently produced comparable or better quality models (log-likelihood on a test set) compared to regular EM, while significantly reducing dimensionality and computation time. We also used random projection to construct a classifier for handwritten digits using the USPS dataset, where each digit is represented as a 256-dimensional vector. By randomly projecting the training data into a 40-dimensional space, we successfully fit a mixture of fifty Gaussians (five per digit) without manual adjustments or covariance restrictions. These experimental results validate their theoretical findings.

2.3 Learning Algorithm

In the research of the algorithm we are going to use, it is essential to provide an explanation of the rationale and methodology employed during the investigation. In order to simplify the problem, we reduced it to two dimensions and converted it into a binary classification task. Additionally, we required an algorithm capable of training on individual samples and functioning in an online fashion.

2.3.1 Gradient-based methods

Gradient-based methods, including variants like stochastic gradient descent (SGD), can be applied to train on one sample at a time. This approach is known as online learning or online stochastic gradient descent.

Unlike batch learning algorithms that require all data to be available upfront like Logistic Regression, online algorithms can update their models incrementally as new data points arrive.

We chose the online gradient descent family of algorithms because they scale up well and it is possible with small changes to expand this to a neural network, so it lets us expand our current work in future. An other very important aspect while choosing and algorithm is to be able to fit on an FPGA, and the design and debug to be feasible on the time we have.

An important work on these algorithms is shown in [11] were online learning is referred as a well-established and highly appealing learning paradigm that has garnered attention in various research fields, including game theory, information theory, and machine learning. It has also gained practical significance with the rise of large-scale applications such as online advertisement placement and web ranking. In this survey, their objective is to provide a contemporary overview of online learning, shedding light on intriguing concepts and emphasizing the importance of convexity in developing efficient online learning algorithms.

Another helpful research of [12] talks about cases where the functions are smooth, gradient-based methods such as gradient descent are commonly used for optimization. These methods typically require evaluating the gradient, which can be obtained by computing the average of the gradients of each function. However, computing the gradient becomes computationally expensive as the number of functions N increases, making these methods inefficient for large datasets in practical machine learning applications.

In traditional gradient-based methods, such as batch gradient descent, the algorithm calculates the gradient of the entire dataset (the full gradient) to determine the direction in which the parameters of the model should be updated. This approach ensures a more accurate estimate of the gradient but can be computationally expensive, especially with large datasets.

Instead of calculating the full gradient using the entire dataset, they use a subset of the data (a mini-batch) to compute an approximation of the gradient. These methods do not calculate the full gradient rf, but instead approximate it using a cheaper stochastic estimate rf. Typically, this estimate is computed using a subset of functions fn. This approximation is known as a "stochastic estimate" because it introduces randomness into the optimization process. Due to their stochastic nature, these methods may converge at a slower rate compared to deterministic methods.

They conduct a comparison between two classic gradient-based methods

(one deterministic and one stochastic) and two modern semi-stochastic methods. They evaluate their performance in practical machine learning scenarios through two sets of experiments. The first experiment involves logistic regression on a synthetically generated dataset, while the second experiment focuses on classification using softmax regression on the MNIST dataset, which consists of handwritten digits.

2.3.2 Regularization (L1 and L2)

An important concept on Machine Learning field is the regularization. It is a commonly used techniques in machine learning to prevent overfitting and improve the generalization of models. Here it is going to be analysed the L1 and L2 Regularization as our algorithm uses it.

L1 regularization, also known as Lasso regularization, its primary objective is to instill sparsity in the model, encouraging some of its parameters to assume exact zero values. This has profound implications, particularly in feature selection, as it automatically identifies and eliminates irrelevant or less significant features. Mathematically, L1 regularization introduces an extra term to the loss function, calculated as the absolute sum of the model's weights, weighted by a regularization strength hyperparameter (more details in Appendix C) denoted as λ. A higher λ leads to a more pronounced regularization effect, pushing more weights toward zero. L1 regularization simplifies models by choosing a subset of features with substantial predictive power while disregarding others, making it particularly advantageous when dealing with high-dimensional data or when interpretability is a priority.

L2 Regularization, or Ridge regularization, serves as a powerful tool for averting over-fitting by penalizing the magnitudes of model parameters. Unlike L1, it does not force any parameters to become exactly zero but instead promotes the model's weights to be small. The mathematics behind L2 regularization involves augmenting the loss function with the squared sum of the model's weights, multiplied by a regularization strength hyperparameter, λ. While L2 discourages excessively large parameter values, it allows them to remain non-zero. This regularization technique enhances model stability, reducing the influence of extreme parameter values and contributing to better generalization. It is extensively employed in regression and neural networks to foster generalization without the need for explicit feature selection. Its

applicability is particularly advantageous when there is no strong prior belief that certain features should be precisely zero, ensuring models remain adaptable and effective.

In practice, a combination of L1 and L2 regularization, referred to as Elastic Net regularization, is often employed to leverage the unique advantages of both techniques in achieving well-regularized models.

2.3.3 Online learning and concept drift

The reason and inspiration behind using online learning was not only the fact that feeding lot's of data on a chip will need a lot of memory. But the main inspiration came from an article [13] with the fancy title "Why Do Machine Learning Models Die In Silence?". As the article states one of the critical challenges faced by companies when integrating machine learning into their business processes is the deterioration in model performance over time. That is often referred to as "concept drift." This phenomenon occurs when the underlying data distribution shifts, causing once-reliable models to lose their predictive accuracy.

The [1] is a concept which will definitely suffer from something like that. Concept drift arises from changes in the distribution of the training data over time. It can cause data points once associated with one concept to be considered part of another concept. For instance, in fraud detection, the concept of fraud is continually evolving, posing a significant risk of concept drift. Such drift can go unnoticed, leading to a gradual decline in model performance. Concept drift primarily occurs due to the evolving nature of data in real-world applications. Changes in customer behavior, external factors, or even the recommendations generated by the model can contribute to this shift. While it may seem undesirable, frequent model updates can harness valuable information from new data, improving prediction accuracy.

Monitoring, proactive retraining, and ensemble techniques are effective strategies for addressing concept drift. It is not easy if it is not impossible to prove that online learning will solve this problem for sure, although theoretically this seems like a valid solution.

2.4 FPGA

In this section, our focus was to examine whether individuals have successfully utilized FPGA technology and ascertain any potential advantages associated with its implementation.

On [14] they have a study on the financial sector. Where seems that it has experienced significant growth in computer-based applications, driven by the increasing volume of financial computations and network transactions. This surge in financial data necessitates fast processing in energy-intensive data centers. To address the dual requirements of low latency and power consumption, specialized hardware accelerators, known as field-programmable gate arrays (FPGAs), have emerged as promising solutions.

This paper focuses on demonstrating the energy-efficient acceleration of option pricing algorithms, specifically Monte Carlo approaches for European options, using FPGA-based Application Specific Processors (ASPs). The development of ASPs is expedited through the utilization of FloPoCo, a tool for generating FPGA-optimized arithmetic cores.

To evaluate the effectiveness of the ASPs, they compare the execution time and power dissipation of the Monte Carlo algorithm implemented on an FPGA-based soft-core processor with that of the ASP. Additionally, they benchmark our design against a previous work that compared an FPGA-based accelerator to a dual-core processor. The experimental results validate that the ASP not only achieves significantly faster execution but also demonstrates orders of magnitude lower energy consumption compared to alternative approaches.

On [15], the primary problem addressed is the exponential growth of new malware, facilitated by easily accessible malware morphing engines. The volume of unique malware samples is projected to reach over a million per day within the next seven years, necessitating automatic methods for large-scale malware triage. Triage involves two main steps: per-sample malware analysis to extract features and pairwise comparison to determine similarity. The challenge lies in the ever-advancing sophistication of malware and the need for scalable techniques to handle the sheer volume. The text introduces BitShred, a system for large-scale malware similarity analysis and clustering, which is agnostic to specific per-malware analysis routines. BitShred employs feature hashing to efficiently represent malware features, enabling dimensionality reduction for memory efficiency. The system addresses the

central issues of efficiently representing, comparing, and correlating features within large volumes of malware. Its main contribution is scalability to datasets orders of magnitude larger than existing approaches, with theoretical analysis and empirical evaluation demonstrating its effectiveness in terms of speed, accuracy, and adaptability to various per-sample analyses.

The work of [16] discusses the compounded challenges arising from advancements in FPGA technology and the growing scale and complexity of deep learning algorithms. The problem being addressed is the inefficiency of general-purpose processors for the implementation of Convolutional Neural Networks (CNNs) due to the specific computation pattern of CNNs. The unique requirements of CNNs, particularly in terms of computation and memory utilization, pose challenges for conventional processors, making them unable to meet the performance demands. To overcome this, various accelerators based on Field-Programmable Gate Arrays (FPGAs), Graphics Processing Units (GPUs), and Application-Specific Integrated Circuits (ASICs) have been proposed. On one hand, state-of-the-art FPGA platforms offer increased logic resources and memory bandwidth, expanding the design space. Simultaneously, the application of various FPGA optimization techniques further complicates the exploration process. The escalating complexity of deep learning algorithms to meet modern application requirements exacerbates the difficulty of finding the optimal solution within this expanded design space.

Recognizing the urgency for an efficient exploration method, the text introduces an analytical design scheme in this work, which surpasses previous approaches in two key aspects. Firstly, while previous studies focused mainly on computation engine optimization, neglecting external memory operation, or connecting accelerators directly to external memory, the proposed scheme incorporates buffer management and bandwidth optimization for enhanced FPGA resource utilization and performance. Secondly, compared to prior studies that reduce external data access through delicate data reuse, their method, which does not necessarily lead to the best overall performance, and often requires FPGA reconfiguration for different layers, their accelerator executes jobs seamlessly across different layers without FPGA reprogramming. The primary contributions of this work include a quantitative analysis of computing throughput and required memory bandwidth for potential CNN design solutions on an FPGA platform, as well as the identification and discussion of optimal solutions for each layer within the constraints of

computation resource and memory bandwidth.

2.5 Metrics

Accuracy, precision, recall, and F-score (F1 or Fb) are common metrics used to evaluate the performance of classification models. ROC (Receiver Operating Characteristic) curves are graphical tools used to visualize and analyze the performance of binary classification models. Also it works as a tool to help enhance the algorithm performance.

2.5.1 Confusion Matrix

A confusion matrix is a table or matrix that is often used to evaluate the performance of a machine learning classification model. It provides a detailed breakdown of the model's predictions and how they compare to the actual ground truth. A confusion matrix typically consists of four values:

- True Positives (TP): These are cases where the model correctly predicted the positive class. In binary classification, it means the model predicted a positive outcome, and that outcome was indeed positive.
- True Negatives (TN): These are cases where the model correctly predicted the negative class. In binary classification, it means the model predicted a negative outcome, and that outcome was indeed negative.
- False Positives (FP): Also known as Type I errors, these are cases where the model incorrectly predicted the positive class when the actual class was negative. In binary classification, it means the model made a positive prediction when it should have been negative.
- False Negatives (FN): Also known as Type II errors, these are cases where the model incorrectly predicted the negative class when the actual class was positive. In binary classification, it means the model made a negative prediction when it should have been positive.

2.5.2 Accuracy

Accuracy is a straightforward and intuitive metric.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{True}{Sample} \tag{2.1}$$

It calculates the percentage of correctly predicted instances out of all instances in a dataset. It may not be the best choice when dealing with imbalanced datasets. In cases where one class significantly outnumbers the other, a high accuracy value can be misleading. For example, in a medical diagnosis scenario where only a small percentage of patients have a rare disease, a model that predicts "not having the disease" for all instances can still achieve a high accuracy due to the imbalance. It's essential to consider the class distribution and the specific problem context when interpreting accuracy. In such cases, other metrics like precision and recall become more informative.

2.5.3 Precision

Precision measures the accuracy of positive predictions made by a model.

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

It answers the question: "Of all the instances predicted as positive, how many were actually positive?" It is particularly valuable when the cost of false positives is high. In applications like spam email detection, a false positive (classifying a non-spam email as spam) can be more disruptive than a false negative (missing an actual spam email). High precision indicates that the model has a low rate of false positive errors, making it suitable for tasks where false positives are costly or undesirable.

2.5.4 Recall

Recall, also known as sensitivity or true positive rate, quantifies a model's ability to identify all relevant instances of a particular class.

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

It answers the question: "Of all the actual positive instances, how many did the model correctly identify?"

It's important to note that there is often a trade-off between precision and recall. As you optimize a model for higher precision, recall may decrease, and vice versa. Finding the right balance between these two metrics depends on the specific goals and requirements of your application.

2.5.5 F-score

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}. \tag{2.4}$$

F1 Score is The harmonic mean of precision and recall. It provides a balance between precision and recall and is particularly useful when dealing with imbalanced datasets.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{2.5}$$

2.6 Navigating Today's Infrastructure Landscape

In this section we will navigate through some contemporary problems. To highlight why this research is still relevant even the higher RAM FPGAs that have been on the market the last few years.

According to the modern ML-Ops [17] here are some Performance Bottlenecks in AI Systems:

- Computation Limitations: A frequent bottleneck in AI systems, particularly in tasks involving complex calculations, large-scale data processing, or training deep learning models.
- Memory Constraints: Insufficient memory can restrict the system's capacity to manage large datasets, store intermediate results, or train complex models.
- I/O Latency: Slow data transfer between storage devices, networks, or external data sources can markedly affect the overall system's speed and responsiveness.

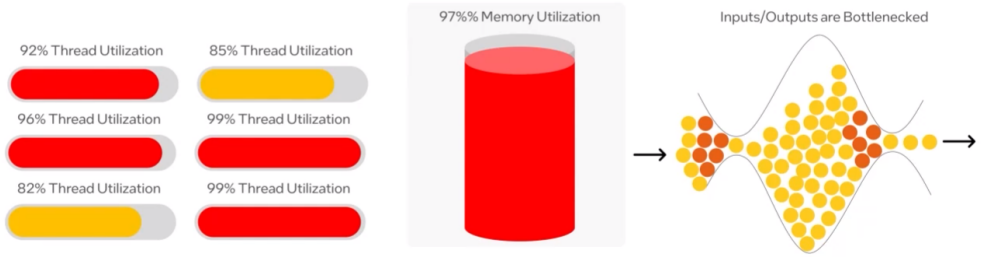

FIGURE 2.1: Common Bottlenecks in Al systems, Source: Intel MLOps course

As we can see from the above memory is still one of the core problems in modern system design.

The reason been that even though the hardware continues to advance, the challenges faced in the technological landscape become more complicated. Also the demand for technology has expanded exponentially across diverse sectors. Plus, there's a growing need for technology in different areas like small smart devices, not just in high-tech stuff. In these instances, the emphasis shifts from the cutting edge of technology to the fundamental need for reliability and seamless functionality. We don't always need the latest and greatest tech, we just want things to work in a descent state.

Chapter 3

Modelling and Algorithm

In this chapter we are going to discuss the model setup, including what challenges initially appeared and how they were resolved.

3.1 Navigating Challenges in Initial Experiments

In our initial experiments there were several challenges like finding the right dataset to experiment on. There are several data bases online like these [18] [19] with free to use dataset categorised by the type of classification, like in our case binary class that one could take and experiment. Our initial approach involved developing an algorithm and comparing results with and without dimensional reduction. To figure out the consistency of our dimensional reduction we looked in different kind of data like medical, image recognition, text, chemistry and others. That would help to identify the kind of data our method is more appropriate. However, in this research we faced some serious problems could have resulted in consistency issues.

We should note here that specific methodologies have been employed to analyse the text data; it is considered as a separate discipline, using some of the techniques that we examine but with a different approach. Some example use cases of text data are referenced in Appendix A

3.1.1 Decomposition Techniques

We achieved remarkable results by reducing the number of attributes to 5-10 using decomposition techniques, as detailed in section 2.2.1. The primary motivation for exploring these techniques was mainly because of the experimental results. The accuracy levels where very close to the original dataset.

Since the process is lossless, the accuracy remained comparable to the original.

Unfortunately, as it is later realized, these techniques require a substantial amount of memory to generate the final set of 5 attributes.

The matrix A is constructed row by row from samples. Consequently, a substantial number of rows from a sample must be loaded to perform the decomposition, expressed as:

$$A = U \cdot S \cdot V^T$$

Here, U and V represent orthogonal matrices, while S is a diagonal matrix containing the singular values of A.

The singular values are the square roots of the eigenvalues of `AA^T` (or `A^TA`) and they provide information about the importance of each dimension of the original matrix.

The matrix U contains the left singular vectors of A, and the matrix V contains the right singular vectors of A. The columns of U and V are orthonormal, meaning that the dot product of any two columns is zero.

All these matrixes should be stored in the FPGA which was impossible. The constraints were such that storing coefficients for even a single row was unfeasible or just enough, let alone accommodating multiple rows and two additional matrices.

Conclusively, to apply PCA we need to process all of the data points beforehand in order to compute the projection. This is too computationally costly if the dataset is very large or not possible at all if the aim is to project a stream of data in real time. For such cases we need a non-adaptive alternative to PCA that chooses the projection before actually seeing the data.

3.2 Prepossessing data

The data preprocessing is a fundamental step in the data analysis pipeline that involves cleaning, transforming, and organizing raw data into a format suitable for machine learning or statistical analysis. It plays a crucial role in ensuring the quality and reliability of the data used for decision-making and model building. A whole thesis could be devoted to this topic alone or

to each of the sub steps needed to be taken. Each of the steps required will not be examined in detail, but some crucial steps that have been essential for this work will be covered here. It should be noted that the dimensional reduction is considered as data preprocessing but here it is discussed as a separate concept.

3.2.1 Data nature/type

How to prepare the data when a dataset has numeric and non numeric values? Unfortunately we do not always know the kind of data we use, sometimes we just have a dataset with numbers while others we have mixed types. In Wiley's book (Applied logistic regression) an example is presented where an independent variable can be coded in 3 different strings. Then the appropriate design is to use two dummy variables (a variable with True/False values only) for coding it, than coding it with a variable taking the values 0,1,2.

A very nice explanation on that could be the signal modulations 4PAM (4-level Pulse Amplitude Modulation) vs 4QAM (4-level Quadrature Amplitude Modulation) where is it very easy to prove [20] that the 4QAM works as 2 dummy variables (boolean variables) plotted in an axis while 4PAM in a single dimension variable

The advantage of 4QAM becomes apparent when considering noise resilience. Due to its two-dimensional nature, 4QAM modulation tends to suffer less from the impact of noise, contributing to improved performance in signal transmission. This observation holds relevance not only for 4PAM and 4QAM but can be extended to more complex modulation types and various scenarios, demonstrating a generalized principle of enhanced noise resilience and performance in multi-dimensional signal modulations.

An ideal dataset for an algorithm to perform well would be a dataset full of boolean variables. Unfortunately, we do not deal with such case most of the time and even complicated types usually matters. As we explain later feature hashing does not have any issues with the type of data, although sometimes that leads to other problems.

3.2.2 Missing values

One main problem was the missing values in datasets; some datasets contain missing values. In some cases the missing data can just be ignored, in others

artificially replaced with several methods or we can even drop this sample completely. In this [21] article there are 13 ways to handle this, which can be sorted in four main groups:

1. **Deletion:** Removing entire rows or columns with missing values.
2. **Imputation:**
   - Replacing missing values with the mean or median of the available data for that variable.
   - Imputing missing categorical values with the mode (most frequent category) of the available data for that variable.
   - Predicting missing values using regression models based on other variables in the dataset.
   - Generating multiple probable values for missing data to account for uncertainty.
3. **Indicator Variable:** Creating an additional binary variable ("indicator") to flag whether a value is missing or not. This allows the missing data to be treated as a separate category in analyses.
4. **Domain-specific Knowledge:** Using some domain knowledge or expert judgment to estimate the values based on the context and understanding of the data.

3.2.3 Causation vs Correlation

Here comes the need to explain the difference between these two concepts and the reason why manually selecting attributes may cause significant problems for someone without specialisation (Domain-specific knowledge) in a field. Hence the reason it is avoided in this thesis and should be avoided without the opinion of an expert in the field, which still does not guarantee that they are not mistaken.

Causation refers to the relationship between cause and effect. It suggests that one event or variable directly influences or brings about another event or outcome. In other words, it is the idea that one thing causes another thing to happen.

Example of Causation: A simple example is the relationship between smoking and lung cancer. Research has shown that smoking is a direct cause of an

increased risk of developing lung cancer. When someone smokes, it increases the likelihood of developing this specific type of cancer.

Correlation refers to a statistical relationship between two variables, where a change in one variable is associated with a change in another variable. However, correlation does not necessarily imply causation.

Example of Correlation: Let's take the example of crime. Ice cream sales and crime could be very closely correlated, but one of course does not cause the other. There is another variable that could underlay this: warmer weather. However, even in this example, we cannot say that the warmer weather is causing crime. There are many other factors at play, such as poverty, upbringing, substance abuse etc.

In summary, causation focuses on the direct cause-and-effect relationship between two events, while correlation examines how changes in one variable are associated with changes in another variable. It's important to note that correlation does not imply causation, as there may be other factors or variables at play influencing the observed relationship between two variables.

3.2.4 Dataset Selection approach

In the process of determining an appropriate dataset for our analysis, all the above sections were carefully evaluated.

The chosen dataset was specifically selected with preprocessing considerations in mind. By opting for a dataset with well-organized and cleanly formatted information, we were able to to streamline the preprocessing phase, enabling a more efficient and focused exploration of the core research questions.

The nature and type of data play an important role in the effectiveness of any analysis. In selecting a dataset, we prioritized those with a clear and well-defined structure, ensuring that the inherent characteristics of the data are conducive to meaningful insights. This careful consideration helps mitigate potential challenges associated with ambiguous or unstructured data.

Addressing missing values can be a time-consuming aspect of data analysis. In light of this, the selected dataset was scrutinized for completeness, aiming to minimize the prevalence of missing values. This approach enables a more straightforward and reliable analysis, as it reduces the need for complex imputation strategies. Given the intricacies and potential challenges associated

with deciphering causation and correlation in datasets, the selected dataset was chosen with a focus on minimizing these complexities.

In essence, the dataset selection process involved a thoughtful balance between the aforementioned considerations. By prioritizing datasets that align with our preprocessing preferences, possess a favorable data nature, exhibit minimal missing values, and present manageable causation relationships, we ensure that our analytical efforts are directed towards addressing the core research questions effectively. This strategic approach not only enhances the efficiency of the analysis but also contributes to the robustness and reliability of the results.

3.2.5 Dataset exclusion criteria

The decision to exclude certain datasets is an important aspect of our research process, warranting explicit mention for the benefit of future researchers. Given the considerable time invested in scrutinizing and rejecting datasets, it seems valuable to dedicate a section that serves as a guide for subsequent researchers. This section will elaborate on the reasons behind dataset rejections and outline the methods used in this discernment process. By sharing insights into the dos and don'ts of dataset selection, we aim to provide a helpful resource for future endeavors in this domain.

Bellow a list of reasons (dos and don'ts):

1. **Inappropriate Output Format:**
    - **Problem:** The dataset did not align with our binary output requirements. Although that seems very obvious as a point it is one of the main things one should pay attention to.
    - **Do not:** Do not consider using datasets that have no mention of binary outputs in their description.
    - **Do:** Check to see whether the datasets contains binary outputs before proceeding.
2. **Inadequate Sample Size:**
    - **Problem:** Many datasets have less than 100 samples which is not something we would look for a big data problem. A limited sample size not only introduces bias but also poses challenges for robust algorithmic performance.

- **Do not:** Do not use techniques to expand the data.
- **Do:** Use dataset with a sufficient amount of data.

3. **Insufficient Attributes:** That is not a problem in general but in our case specifically.
4. **Excessive Size:**
   - **Problem:** Although a very small dataset can be a problem, a huge one could also be. As a student / researcher someone may not have access to a machine that has enough memory to handle a huge dataset. Even if such an access is given, downloading and running experiments with a 100G dataset can be very time consuming for such a work, introducing practical challenges.
   - **Do not:** Do not overestimate the hardware resources.
   - **Do:** Define clear specifications on what the available equipment can handle.
5. **Missing Values:** Analysed in 3.2.2 section
6. **Class Imbalance:**
   - **Problem:** Class imbalance can affect the performance of machine learning models, and our selection process took this into account to ensure robust and fair evaluations. Even with a high number of samples rectifying a class imbalance can cause data biases.
   - **Do not:** Avoid under or over sampling data as it may cause extra bias by not adding edge cases.
   - **Do:** Select balanced datasets, unless the algorithm is immune or the specifc task requires you to have an imbalance.
7. **Limited Usage Dataset:** While not exactly problematic, it is generally advisable to choose datasets with broader usage. This ensures the availability of benchmarks and facilitates comparisons with existing research.

3.3 Test set up

Here we elaborate some on some information about the test set up, including how it is set.

3.3.1 Data formatting

In python simulations we used sparse methods because they are more efficient than dense methods in certain scenarios due to the nature of data structures and computations involved.

Dense methods refer to algorithms and representations that consider and store all elements in a dataset, including zero values. They are straightforward to implement and are well-suited for scenarios where the data is dense and small to moderately sized.

Compare this with sparse methods, where only non-zero elements are explicitly represented, resulting in more efficient storage and computations when dealing with datasets where a significant number of entries are zero.

Sparse matrices are particularly useful when dealing with large datasets where the majority of elements are zero. Instead of explicitly storing zero entries, sparse matrices only store the non-zero elements along with their indices.

3.3.2 Dataset

Presented below are the details of the last four datasets, identified with widely recognized names on the internet. It's noteworthy that this selection followed a thorough examination of approximately 50 datasets, after using the techniques discussed earlier and then applying the selection criteria mentioned in the previous paragraph. Emphasizing the meticulous process lead to the inclusion of these final four.

All the datasets were primarily obtained from two main sources [18] and [19].

1. **Farm Ads**

   This data was collected from text ads found on twelve websites that deal with various farm animal related topics. Information from the ad creative and the ad landing page is included. The binary labels are based on whether or not the content owner approves of the ad.

   Exported Dataset Information:

   - Total samples: 4143
   - Class balance: 53.31% (0) 1933 - (1) 2210
   - Total attributes: 54877
   - Density: 0,00358

2. **GISETTE** Is a handwritten digit recognition problem. The problem is to separate the highly confusible digits '4' and '9'. This dataset is one of five datasets of the NIPS 2003 feature selection challenge.

   Exported Dataset Information:

   - Total samples: 6000
   - Class balance: 50%
   - Total attributes: 5000
   - Density: 0,1297

3. **Dexter** The original data were formatted by Thorsten Joachims in the "bag-of-words" representation. There were 9947 features (of which 2562 are always zeros for all the examples) representing frequencies of occurrence of word stems in text. The task is to learn which Reuters articles are about 'corporate acquisitions'. We added a number of distractor feature called 'probes' having no predictive power. The order of the features and patterns were randomized.

   This dataset is one of five datasets used in the NIPS 2003 feature selection challenge.

   Exported Dataset Information:

   - Total samples: 600
   - Class balance: 50%
   - Total attributes: 20000 - Real: 9947 Probes: 10053
   - Density: 0,004703

4. **Real sim** SRAA: Simulated/Real/Aviation/Auto UseNet data [document classification] 73,218 UseNet articles from four discussion groups, for simulated auto racing, simulated aviation, real autos, real aviation. This data was gathered by Andrew McCallum while at Just Research.

   Exported Dataset Information:

   - Total samples: 20958
   - Class balance: 69.27% (1) 22238, (-1) 50071
   - Total attributes: 72309
   - Density: 0.002449

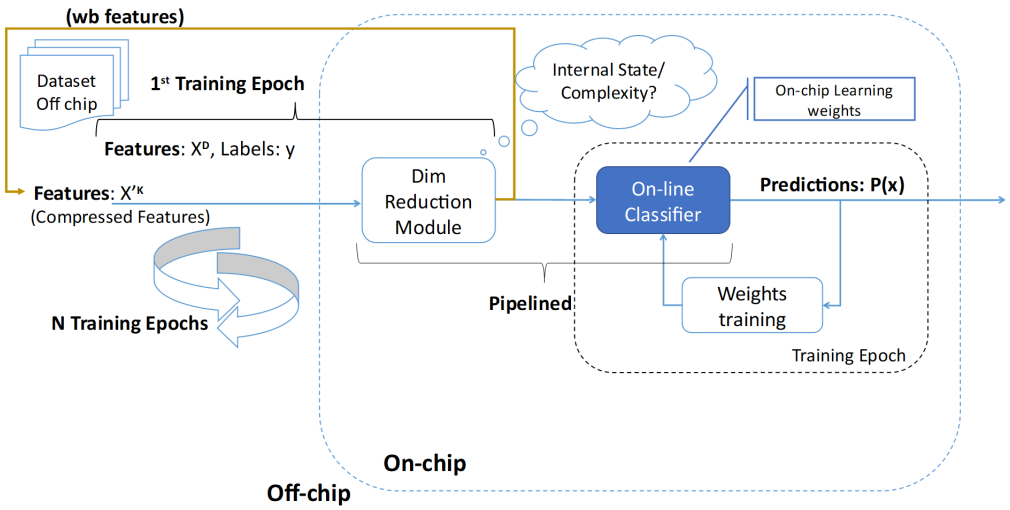

FIGURE 3.1: Proposed system - block diagram

3.3.3 Procedure

In the high-level architecture, we adopt an online learning classifier to decouple FPGA memory resources from the dataset size, specifically in terms of training examples. As the training will be performed in small batches (e.g using a gradient descent solver) the limiting factor upon scaling this setup will be only the dimensionality of the dataset as it explicitly determines the learning parameters we need to estimate. The scalability bottleneck primarily stems from the dataset's dimensionality, as it explicitly dictates the learning parameters we must estimate. In the proposed high-level architecture, we consider the learning classifier to run in an on-line setting in order to decouple the memory resources of an FPGA from the dataset size (in terms of training examples). In this work we don't examine other parameter reduction techniques such as weight sharing etc. as those techniques can be complementary to the proposing scheme. By coupling with the learning classifier the Dim Reduction Module we are able to reduce the dimensionality of the dataset in a transparent way and hence the on-chip learning parameters used by the classifier.

As illustrated below, the training dataset is always stored off chip or could be streamed on-the-fly. The Dim Reduction Module reduces the features dimension from D to K and directly feeds data to the Learning module at the desired batch size. The novel idea behind the Dim Reduction module is that it generates the projection matrix on-the-fly, column by column without storing any of the coefficients. At the same time, it performs the dot product

with the feature input in a single pipelined process. During the 1st training Epoch the Dim Reduction module, also performs a write back step and sends to the off-chip memory (wb features) each training batch with Compressed Features (X') in order for the next training epoch to use the lower dimensionality features directly, saving in this way computations and I/O latency. In this regard, the added latency of the Dim Reduction module it affects only the very first iteration of the data out of usually hundreds or even thousands of iterations. In this case the proposed scheme is a win-win scenario which improves both speed and memory footprint, if only at a very slight cost in classification accuracy. More details are going to be given in the Evaluation Section.

3.3.4 Test evaluation

The evaluation strategy employed was to use both train-test split and Stratified K-Fold cross-validation techniques. As denoted in [22]: StratifiedKFold is a variation of k-fold which returns stratified folds: each set contains approximately the same percentage of samples of each target class as the complete set. The train-test split initially gauged the model's performance on unseen data by dividing the dataset into training and testing sets. The choice made here was to divide the dataset into five parts, so 80% train and 20% test. Additionally, Stratified K-Fold cross-validation ensured a more robust evaluation by maintaining class distribution proportions across folds. This approach allowed for a thorough assessment of the model's generalization across different subsets of the data, providing reliable insights into its overall performance.

3.4 Feature Hashing

Feature hashing is also referred as the hast trick and it is a domain of hash kernel methods. Hash kernel methods have several uses as we show in the previous section. In this work we are going to use them to reduce the dimensions of our dataset. I use the term feature hashing because it is more describable for what we do, we hash the features.

3.4.1 Hash function

To implement the this method we firstly need a hash algorithm. We found that there are ways and some relevant work on how to chose an algorithm

according to criteria like [23] and also we conducted a few experiments which trialled several algorithms.

Our requirement was a fast function, easily implemented in hardware that can hash text and will spread the data as much as evenly as possible. There was no need of using a complex cryptography function that it is not reversible or any other feature that a hash function may offer. Python offers a hash function which met our expectations results wise but after some research and experiments as shown on table 3.1 we found out that any hash function was actually doing the job.

The function used is called MurmurHash3, the selection criteria were met and we found several people using this. Also there was an implemented python version which was very helpful.

We should note here, before ending this section, that there were actually some results variations and it seemed like some functions given actually a better accuracy score in the end, but these results were only around 1% different so there were considered random and not taken as usable.

Below is an array with some experimental numbers verifying the above claim. The comparison is between [24] MurmurHash3 version 2.3 and the build in hash function [25] of python.

The Seed column refers to the value of the seed fed to the random number generator build in python which can help us reproduce the result as it initializes deterministically the random number generation.

The D value refers to the number we used to **modulo** the result that each function returns.

3.4.2 Hash Arguments

In this section we are going to discuss what exactly were the arguments we gave to the function and why. The obvious way was to just hash the value of each feature, but this did not work well experimentally and that is normal. The reason behind that is if we just hash the value, and more than one features have the same value they will collide in the same feature in our new row we create. The solution was to hash also the index of the value. But for code readability reasons and to make it a little more random, a new string format was created which was feat_<index>_<value>. For example the example[20] = "frogs" is hashed as feat_20_frogs.

	D = 256		D = 1024	
Seed	Murmur	Py Hash	Murmur	Py Hash
0	0,55	0,84	0,92	0,83
1	0,54	0,79	0,85	0,79
2	0,80	0,42	0,64	0,55
3	0,78	0,58	0,79	0,83
4	0,54	0,69	0,86	0,90
5	0,71	0,83	0,70	0,91
6	0,65	0,45	0,84	0,84
7	0,70	0,58	0,91	0,90
8	0,74	0,59	0,85	0,78
9	0,58	0,85	0,74	0,84
Mean	0,66	0,66	0,81	0,82

TABLE 3.1: MurmurHash3 vs Python Hash

Here we realised that this algorithm will probably struggle a lot with float numeric data. The reason being that the value 1.12345 and 1.12346 are actually very close and they should be considered as the same value but our aforementioned algorithm does consider them as two totally different values and only by randomness they will be considered as similar. One solution on that would be to preprocess the data and add them in big buckets or eliminate some digits depending the variance of the sample. So if the variance is 0.05 we can consider buckets of 0.05 ranges or consider the important digits to be only 2.

We ended up not looking that on detail but it was something that came up during the research.

3.4.3 Algorithm

The Feature hashing is mainly based on collision counting. The fundamental concept involves applying a hash function to the data points in a way that maximizes the likelihood of collisions occurring between objects that are in close proximity, compared to those that are distant. It uses a hash function in order to find the index of our new reduced array where the indexed position is incremented by one.

Except the previously used algorithm where everything is hashed, we propose an edition where only the non-zero attributes are hashed, so we call the hash function less times, making our implementation faster.

The algorithm consists of the following parts:

- Input: The input raw we are about to process and the value *D* where is the dimension of the output array.
- Hash Function: A function that hashes the words we create.
- Output: Is the *D* size array where the new data are exported

Bellow is an example code in python

LISTING 3.1: Feature Hasing

```
for e, value in enumerate(InputData):
    if(value!=0):
        word = f"feat_{e}_{value}"
        index = abs( mmh3.hash(word) ) % D
        OutputData[index] += 1
```

This is not the final code but a more readable version, as the optimised for speed version uses notation which will be very hard to be understood from people not familiar with python notation.

It takes as argument the *value* and the *index (e)* of the attribute. It hashes the word "feat_E_VALUE". It returns an integer which we apply the modulo operator to bring it in range of *D*. Then the *OutputData* array is being increased at the *index* position by 1. So we actually count the collisions made from a *A*-sized array to a *D*-sized array from our hash function. Our new array of attributes is the *D*-sized array.

The initial motivation of this work was to squeeze big datasets in smaller ones, with focus on sparse datasets. Theoretically zero value attributes should not be considered as useful and they should be skipped. Like in this work [1] where zero means no data, the user did not interact so we do not care about that attribute. Although it is not mentioned in bibliography that zero values could be skipped that was an **improvement** made in this work. After making that the accuracy and the performance of the algorithm raised.

Some further explanations of the algorithm:

The reason for using the e (index of row/position) to create the word we hash is that in case of having the same value in many attributes it does not distinguish in which place is that value. Where by adding the index it does.

3.5 Random projections

Another similar but more mathematical way for dimensional reduction is the random projections.

Despite the term "projection" being used, the random selection of vectors results in transformed points that are not precisely true projections but closely approximate them from a mathematical standpoint. They offer a clever approach to reducing the dimensionality of high-dimensional data while preserving essential properties. This technique finds applications in various fields, including data mining, signal processing, and pattern recognition. The concept behind Random Projections is similar to Principal Component Analysis (PCA) at its core. However, in PCA, the computation of the projection matrix relies on eigenvectors, which can become computationally demanding when dealing with large matrices. Random projections leverage the idea that a random linear transformation can preserve pairwise distances between data points with high probability. Unlike deterministic methods, random projections offer a computationally efficient means of reducing dimensionality without the need for intricate calculations. A matrix $R < ft, D >$ is projected to the original matrix $X < n, ft >$ so the result is a matrix $X\_r < n, D >$. Where $ft =$ original feature size, $D =$ the desired feature size, $n =$ the number of samples. We examine two main methods for this procedure. The Gaussian and the sparse method, which is an alternative more time and size efficient method. Though we studied these methods they do not make any assumption for the data structure and nature so we try some different numbers for sparsity and size of the projection array.

3.5.1 Gaussian Random Projection

Gaussian Random Projection involves generating random Gaussian (normal) vectors. We could construct a Gaussian LUT by choosing elements randomly from a Gaussian distribution with mean zero. The randomness in the Gaussian vectors allows for a diverse set of projections, and it has been shown that such projections can preserve pairwise distances reasonably well. It is commonly used when the data is not sparse, and the goal is to reduce dimensionality while preserving certain structural properties.

3.5.2 Sparse Random Projection

Unlike Gaussian Random Projection, sparse vectors have many zero elements. Sparse projections are generated by randomly choosing a small number of non-zero elements in the vectors In sparse version there is a work from [9] Achliopta's where s is the variable that determines the scarcity, for example if $s = 3$ only the $1/3$ of data is to be processed. This is a comparatively simpler method, where each vector component is a value from the set $-k, 0, +k$, where k is a constant. In our experiments especially in sparse datasets we were able to raise that value to even higher values like 10 or 20 and still get a very useful compression result. Python has an implementation in [26] that it is been use for the experiments and results later.

3.6 Prediction model

Our model algorithm is Stochastic Gradient Descent (SGD) is an optimization algorithm widely used in machine learning and deep learning. It is particularly suited for large datasets and complex models. This algorithm is what used for the python simulation. Another algorithm called FTRL-Proximal (Follow the Regularized Leader - Proximal) uses the logistic regression model with some improvements proposed at [1] with two regularization parameters and a different learning rate "η" rule. The algorithm is very similar to SGD (Stochastic Gradient Descent) but they use different approach for the symbol h. The optimal value for learning rate can be exported experimentally but we used some of the proposed ones without seen any notable positive difference when changing them a little. FTRL is used in our FPGA implementation and some further details about the design will be analysed there. Although in this section we are going throw some more theoretical details and the reason behind the decision to use these to algorithms.

Some characteristics that lead to chose this algorithms:

- Parallelization:

  It can be parallelized to improve training speed on multi-core processors, distributed computing environments or in our case FPGAs.

- Model Flexibility:

  It can be used with a wide range of machine learning models, including linear models and deep neural networks. That gives this work the

opportunity to be used in future without the need to change the basic structure but use it as base for next steps.

- Efficiency:

  It is designed to be computationally efficient, making them suitable for online learning and real-time applications. They process data points sequentially and require minimal memory compared to batch optimization methods.

- Regularization:

  In SGD, the regularization terms like L1 or L2 can be added to the loss function. In FTRL the L1 and L2 regularization are included in its update rules to encourage sparse and stable models.

3.7 Conclusion on Metric Selection

While precision, recall, and F-score are widely recognized and essential metrics, there are scenarios where simplicity and practicality outweigh the nuanced insights they provide. In the context of our current work, the decision to primarily rely on accuracy stems from a thoughtful consideration of various factors.

Precision, recall, and F-score are particularly valuable in tasks where the consequences of false positives and false negatives are markedly distinct, such as in medical diagnoses or fraud detection. However, the intricacies introduced by these metrics also bring challenges and complexities. The desire for a straightforward evaluation process led to the strategic decision to prioritize accuracy.

Accuracy, while a more straightforward metric, aligns with the specific needs of the current work. In situations where the costs of false positives and false negatives are comparable, and the emphasis is on overall model correctness, accuracy provides a clear and easily interpretable measure of performance.

In conclusion, after considering various aspects discussed above, the only metric that will be utilized for this assessments is accuracy.

Chapter 4

FPGA

In this chapter, the implementation of our algorithm is analyzed using the HLS approach. We have a top level function and two classes to implement, the dimensional reduction and the Learner function. Our top level calls the module with a row-sample of A attributes at a time and it gets back a reduced array in size of D«A attributes. We implement 2 different functions of dimensional reduction the Feature hashing and the Random projection (Gaussian, Bernoulli). The next call of our top level is the learner module with a given argument the D-sized array and returns the result of the prediction or,and fits the sample to the learner.

4.1 Essential Aspects of HLS Code

4.1.1 HLS code

All the code written for the FPGA is written on vivado HLS and the simulations run with that software. Although it should be mentioned that all the code is been tested in python 3.6 before it is implemented in C. In cases that a python library used but we present an HSL implementation the code is been written in pure python (without any library usage) and translated / re-written in C for the HLS. That means that the code in C is been used only for sizing and timing reasons while the python gave us all the accuracy and reliability result presented. Also except the cases that it is clearly stated the a python library is used, all other codes where implemented from scratch.

4.1.2 FPGA chip

In further details about the simulation as target device we used a Artix 7 FPGA chip. More specificly the xc7a100t family the model csg324-1.

This is a mid range chip with the following available resources:

- BRAM_18K: 150
- DSP48E: 120
- FF: 65200
- LUT:32600

Because technology changes constantly new editions are available with similar characteristics. Similar chips and information about the pins, voltages and data-sheets can be found in the official page [27].

We did not own this chip and we run only the simulations with that as a target device for reference.

4.1.3 Purpose

It should be noted that this implementation does not have as purpose to produce an optimal result. Tuning a model is a very difficult work and it relates to many factors. In the FPGA-HLS implementation our purpose is to find the sizing and the design. It was not worth spending time on finding the best learning rate for each dataset or plotting the ROC curve via data-points where the HSL simulations gave us. Although it is very important to select the correct types and variables so we can have a precise timing and sizing estimation, from the simulation process. For example our seed for the random function generator we create analyse later needs to be an integer, but it is pointless to have a big integer as any small one does the work, even if someone would like to use more than one seeds to experiment. So having that as an 8-bit int give us 256 which is giving as a lot of options. For other values like the coefficients we need very precise floating point variables, or else we may lose accuracy due to arithmetic losses.

4.1.4 Default module

For comparison reasons a default design is provided. This is actually very similar with a few variations that will be explained. The first variation is that there is no module for dimensional reduction. So the data are obtained as an input and the only processing it happens is to go through the learner function in the fit and predict stages. The second variation is the size of the array that

holds the coefficients in learning function. This is a floating point array with the size of input as each feature has its own coefficient.

One of our main work targets was to make this array as smaller as it could be, since the input and output is for all three implementations the same. It is been tried to hold less coefficients as the RAM on those chips is a luxury.

We compare the sizing for implementations that would be impossible due to the aforementioned array.

4.1.5 Input data

While a considerable amount of this work is dedicated to preprocessing input data, it's essential to understand that, at the High-Level Synthesis (HLS) level, we make an assumption that the input data is already in a valid and processable format. That means that we are not dealing with any other process like cleaning the data or format conversion at the HLS stage. This decision is intentional, as it helps us avoid the creation of entirely unrelated modules that wouldn't have a direct impact on the current work.

It's worth noting that data cleaning is typically a necessary step in the development of any algorithm, as it ensures the quality and integrity of the data. Although as it is shown the feature hashing algorithm would totally skip this part, while this might appear to streamline the process, it can result in less robust outcomes. Therefore, it's essential to keep a balance between optimizing the algorithm and maintaining data quality throughout the pipeline.

4.1.6 Top level

The top level module has a function to execute our whole functionality. It consists of some control signals and the input data. It gets a reset signal which just resets all the stored weights (coefficients), it is calling the learner.reset() function. An operation control signal which decides if we are in a predict or a fit process. A sample code is the following:

```
int top_lvl_execute(int reset,
    int reduct_mth,
    int operation,
    float data_x[DatasetSize][SizeOfAttribute],
    int data_y[DatasetSize],
    int preds[DatasetSize],
    float compr_data_x[DatasetSize][D],
```

)

In this module it is also being hold the w variable which is the weights. Another universal file that is needed for all the operations is the general definitions header file (gen_def.h) In here except the global definitions, some other variables are been defined, the names are self explanatory for most of them.

```
#define DatasetSize 1
#define SizeOfAttribute 54877
#define D 5487 // Number of weights to use
#define SEED 10 // seed to be used for random numbers
```

4.2 Learner Unit

The learner unit is where the learning algorithm lives in. This unit consists of three basic functions, the reset, fit and predict. Every unit will be analysed further in the following subsections, with the exception of reset, given its straightforward functionality. The reset function is running a simple for loop to initialise a variable that holds the coefficients. Although the functionality is so simple it is also very important for the initialisation and the reset in cases something goes wrong or in situations where issues arise or users observe diverging predictions

Also this unit has a header file which is the following one.

```
#define lrn_rate  0.01
class My_learner
{
public:
        My_learner(/* init */);
        int predict(float data[SizeOfAttribute], float *w);
        void fit(float data_x[SizeOfAttribute],int data_y,float *w);
        void reset(float* w);
}
```

4.2.1 Fit function

In this section it is going to be analysed the functionality for the fit in fpga. Fit also usually refereed to the bibliography as train, although because the convention in python is calling these types of functions as fit I used this term. The fit function is the one that changes the w array where the coefficients are

held. As it is shown above a very important variable also held here is the learning rate of the algorithm which is set to 0.01. Common learning rate values are around 0.01 to 0.05 for these types of algorithms. That number, is a crucial hyperparameter that determines the step size at which a model adjusts its parameters during training.

The functionality of the fit function consists of 3 main stages and the pseudo-code is presented in Algorithm 1.

Firstly around lines 1 to 2 we calculate the dot product of weights and sample values.

The second part lines 3 and 8 is to go through and activation function (details about that can be found on the B) Here I chose to have a sigmoid activation function with cut off at 5 or minus 5. This is done because in after 5 the value is already close to 1 or 0 so we can use less power from calculating the exp which is an expensive function.

The final step lines 9 and 12 on this function is to then edit the coefficients. As it can be seen in case the x equals to zero there is no point making the calculations after it

Algorithm 1 FPGA, Fit Function Learn Unit

1: **for** $i = 0, \ldots, D-1$ **do**
2: $\quad dot\_prod += data\_x[i] \times w[i];$
3: **if** $dot\_p > 5$ **then**
4: $\quad p = 1$
5: **else if** $dot\_p < -5$ **then**
6: $\quad p = 0$
7: **else**
8: $\quad p = 1/(1 + exp(-p))$ ▷ Sigmoid Activation Function
9: **for** $i = 0, \ldots, D-1$ **do**
10: $\quad$ **if** $data\_x[i] \neq 0$ **then**
11: $\quad\quad g = (p - data\_y) \times data\_x[i]$
12: $\quad\quad w[i] = -lrn\_rate \times g$

4.2.2 Prediction Function

The predictive mechanism plays an important role in generating forecasts using the trained model. Once the model parameters have been learned during the training phase, the prediction function is applied to new, unseen data to

generate predictions or classifications. The prediction function involves computing a weighted sum (dot product) of the input features, with each feature multiplied by its corresponding learned coefficient.

for $i = 0, \ldots, D-1$ **do**
    $dot\_prod+ = data\_x[i] \times w[i];$

The result of this computation is often passed through an activation function, depending on the specific problem at hand. In our case this step was not necessary not only because of the time and extra energy cost, but because a simple solution with an if case selecting as positive class (or 1) the sum is greater than 0.5 else the negative class (or 0).

4.3 Dim Reduction - Feature Hashing

Feature Hashing module consist of two main functions and does not need any special variables to hold on, like weights etc. That makes is an exceptional selection algorithm for cases that process power is not an issue but memory it is. That concern is not only in cases we do not have sufficient RAM but for problems that keeping a variable alive is difficult. The main functionality is the one that creates collisions, while the other generates a number from a hash word. The output is a collision array that is then fed into the ML module.

A diagram of how it works is presented below.

4.3.1 Hash trick

The Feature Hashing algorithm also known as the hash trick, has a very similar implementation with the one described in python section were the modeling happened. In more details we firstly initialise the output array with D

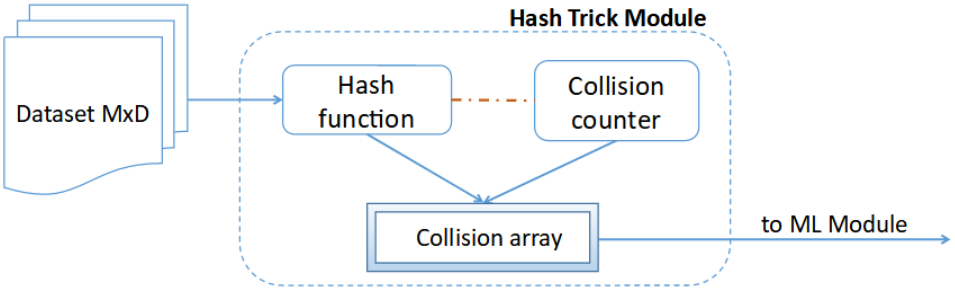

FIGURE 4.1: Feature hash block diagram

length. Then we loop through the sample skipping the zeros, for implementation reasons to take latency in simulations there are provided two versions. One version is the one which just skips the zeros with just a conditional if but loops thought the whole sample. The other is for simulation reasons only that loops only to an average number of non zeros. After that stage we send the word in MurmurHash dividing the result we get with D. The output array is increment by one in that position.

4.3.2 Murmur Hash 3

The Murmur Hash utilized in our implementation is not a proprietary creation; instead, an existing public domain implementation [28] is been used. Specifically, MurmurHash3_x86_32 was chosen based on considerations of throughput and latency. This variant is acknowledged for having the lowest throughput, but it also boasts the advantage of the lowest latency, making it particularly suitable for certain use cases. According to the documentation, "If you're making a hash table that usually has small keys, this is probably the one you want to use on 32-bit machines. It has a 32-bit output." This aligns well with our requirements, given that our keys are characterized by their compact size, making this variant an ideal fit for our application.

4.4 Dim Reduction - The Random Projection

The random projection as a process one could say that has some similarities with the concept of convolution. There are two studied implementations in this work in theoretical level. Only one is implemented for reasons that is been said and explained in details.

A diagram of how it works is presented below.

4.4.1 Gaussian Random Projection

We are not going to give an FPGA implementation for this method and below we will see what are the reasons. One option is to construct a Gaussian LUT, either save it on memory which is and it contradicts the purpose of this thesis to use as little RAM and ROM as possible. The other option is to construct on on the fly as we do with random numbers which is a difficult process out of the boarders of this thesis as [29] and [30] could saw, because the challenges are a lot to guarantee a result that will not cause extra problems. The final

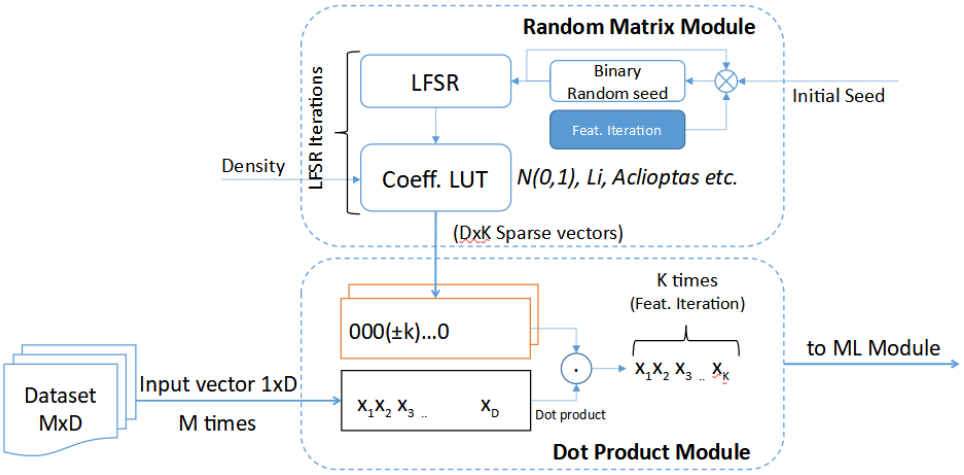

FIGURE 4.2: Random Projection - Block Diagram

reason is the sparse version is proven to work and would work with non perfect Gaussian numbers as soon as they have a mean value of zero. So there is apparently no reason to implement this since the following method can cover this case.

4.4.2 Sparse Random Projection

The implementation of this module is as it follows. Firstly we need a an LFSR, the implementation of which is given below with an example of usage. Secondly we need to move with the actual process of the projection. Due to implementation complexity these two operations are not separated in functions but run in one. Mainly it would require to pass a need updated seed in every round or the return of the whole table. Although the second scenario is memory expensive and the first could be feasible it requires a generator in python where the prototyping happened and a more complex function in C. Which considered as odd in our study.

As the algorithm shows below in line 1 we have calculated how many cells are we taking from the row for each sum. Then in line 2 we loop D times as the size of the output array. Since we are creating a distribution we need half of the elements to be positive and half of them negative. That is being implemented in lines 5 and 8 were in case of odd number we add the element to the sum or else we subtract.

Algorithm 2 FPGA, Random Projection

```
1: cellsProcessed = SizeOfAttribute * density
2: for i = 0,...,D − 1 do
3:     for j = 1,...,cellsProcessed do
4:         rndNum                    ▷ Here we generate the Random Number
5:         if rndNum > 0 then
6:             output[i]+ = input[rndNum]
7:         else
8:             output[i]− = input[rndNum]
```

4.4.3 LFSR - linear feedback shift register

In hardware we use a LFSR [31] function which periodical gives us random numbers depending on the "feedback polynomial function" we use. The random numbers are generated on the fly by an LFSR function, which periodical gives us random numbers depending on the "feedback polynomial function" we use. If our feature size is A=500 and we the willing reduction is D=50, we use the taps[9,5] and the function x^9+x^5+1 (can be written as $x^9 + x^5 + 1$) with a range of 512, to take D different numbers. (NOTE the symbol ^ is the XOR operator [32] according to this table) The term "taps" in a LFSR refer to the positions in the shift register that are used to calculate the next bit in the sequence. These positions are connected to the XOR gates, and their values are XORed to produce the feedback that influences the shift register. In the context of an LFSR, taps are represented by the bit positions that contribute to the feedback mechanism.

In case we take numbers bigger than the D we just skip the number so we avoid overflow. The reason we skip the values and we do not subtract or taking the mod(D) is because this may cause collisions and extra noise to the algorithm. Experimental results show that the numbers generated by LFSR are uniformly distributed in range of the polynomial function we use. Here we faced a problem because every polynomial function has a range based on power of 2. The datasets attribute size is definitely not a power of 2 size but if it is close we just skip some numbers. But may not be very close to it and we may need to skip many values in every sample causing us a huge latency for no reason. The solution is simply to approach the size of attribute using more LFSR functions.

4.4.4 LFSR - use case

Let's take an example of how we solve it in a dataset of 10.000 attributes. The purpose is to produce random numbers ranged up to 10.000 and use them to obtain random features of the input array (the sample with the features).

The closest power of 2 giving a bigger number of 10.000 is 14, resulting a random number generator in range of pow(2,14)=16.384. So now let us assume that we will have a D of 1000, so the reduction will be from 10.000 to 1.000. The average random numbers the lfsr will produce (for us to keep 1000) are about to 1600 and we must skip a mean of 600 numbers every time. Because our sample is 10.000 any number bigger than that should be skipped, like 10.001 because there is not such an attribute in this position. That could cause a big latency problem making us to skip lots of unused cycles every time.

So instead of that we will use a combinations of 2 lfsr one with power of 13 so pow(2,13)= 8.192 and one in power of 11 so pow(2,11)= 2.048. We will have a range of 10.240 and the mean of numbers we have to skip is about 20 every time. The trick here is that we do not add the numbers each lfsr will generate because that may cause collisions as many combinations can give us the same number and it's risky. The best solution is to use the lfsr13 to generate 800 numbers (covering the 8192 values and taking 800 random sample from this area) and the lfsr11 to generate other 200. Adding 8.192 to every index we generate from the second lfsr and skipping every value is bigger than (10000-8192=) 1808. So the lfsr11 is responsible to sample indexes from 8192-10000 while the lfsr13 is responsible to sample indexes from 0-8192.

Chapter 5

Results

On this section are presented the results obtained from the software and hardware implementations.

5.1 Results breakdown

As mentioned in previous chapters, the software is used to give us the algorithmic predictions accuracy and the actual size (as per features) dimensional reductions. While hardware is exclusively employed for design purposes and chip sizing. We only made the port synthesis simulation which is considered as enough to validate that the resources needed.

The machine learning algorithm used is the one described in section 3.5.2

The presentation of results is structured as follows: our baseline is the "gain 0" scenario, where the algorithm predicts without any external influence on the data. The other cases, denoted as gain x, showcase our contributions. In certain graphs, the "gain 0" is referred to as the benchmark case, depicted by a horizontal line.

5.1.1 A brief of table column name explanation

This index is the same for all tables presented below. As an example a 1K (1.000) size of array is refereed.

- Gain It refers to the number we divided the initial feature size to take the final size. So in 1.000 features with gain 10 we have 100. This ratio can be seen as the on-chip memory compression gain. Gain 0 is the default case without any reduction.

- Feats - **Features**: Gives us the exact number of features we kept (here 100)
- NZ - **Average non-zero elements in features**: How many features were non zero on the final sample. (If in 1000 we have 900 zeros and 100 non zero this column has value 100)
- D Red - **Dimensional Reduction (latency)**: The latency for dimensional reduction unit in cycles.
- Pred - **Prediction Module**: The latency for prediction unit in cycles.
- Fit Mod - **Fit Module (latency)** The latency for fit unit in cycles.
- Total lat (DR+Fit) - **Total latency (Dim Red+Fit)** The total latency for a sample to get a size reduction and pass through the fit stage. Note here that since fit and predict times usually are very similar.
- RAM - **BRAM_18K** How many as an absolute number, BRAM_18K units are been used.
- Acc - **Accuracy** The average accuracy of a cross validation with 5 fold sets.

Note that lots of columns are referred to average values, as the simulations were conducted with a cross-validation of 5 folds.

5.2 Farm ads

More details about this dataset can be found section 3.3.2 item 1.

In this dataset both reduction methods work and offer 24x memory reduction with only about 2-4% loss in accuracy. Hash kernel though offers a significant throughput improvement as well with 11.4x speedup from the first iteration and 20.7x after the first iteration.

As we can observe in the tables in random projection method from using the 85% of chip BRAM we end up using only 2%. While the other resource went up only by 3-5% without significant accuracy loss. Is is important to note that the Non-zeros kept around the same absolute value from 197 features falling down to around 170.

The feature hashing algorithm achieved similar results to RP as presented above. Although the usage of DSPs raised by 20%, the memory usage though

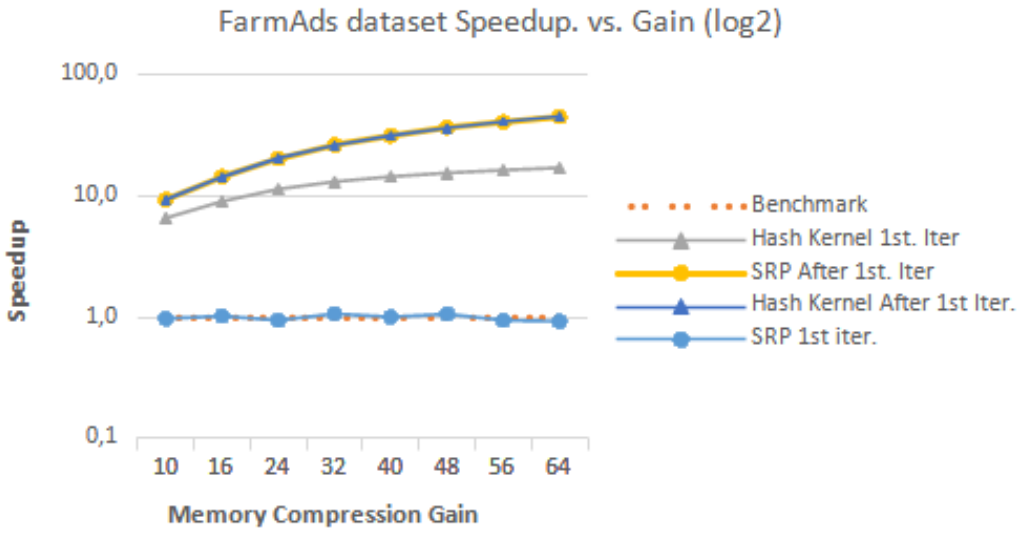

FIGURE 5.1: Farm ads Dataset - Speedup comparison

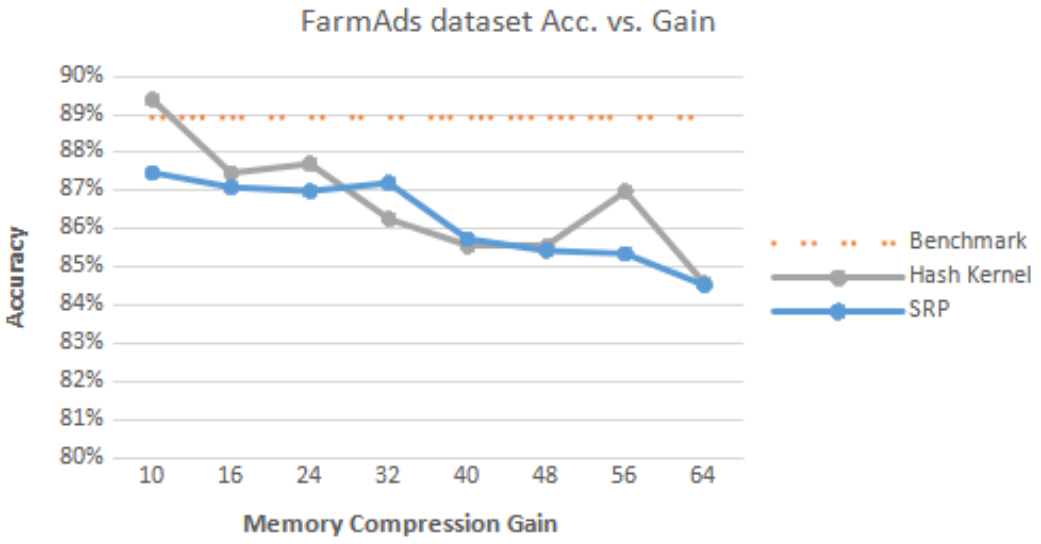

FIGURE 5.2: Farm ads Dataset - Accuracy comparison

Gain	Feats	NZ	Dim Red (cycles)	Pred (cycles)	Fit Mod (cycles)	Total lat (DR+Fit)	BRAM	Acc
0	54.877	197	-	548.770	552.776	552.776	128	88,91%
10	5.487	200	510.291	54.870	59.136	569.427	32	87,47%
16	3.429	183	504.063	34.290	38.199	542.262	16	87,09%
24	2.286	197	562.356	22.860	27.063	589.419	16	86,99%
32	1.714	172	498.774	17.140	20.818	519.592	8	87,20%
40	1.371	177	534.690	13.710	17.493	552.183	8	85,73%
48	1143	163	507.492	11.430	14919	522.411	8	85,43%
56	979	173	575.652	9.790	13.489	589.141	4	85,35%
64	857	169	588.759	8.570	12.185	600.944	4	84,52%

TABLE 5.1: Rand projections - Farm ads

	BRAM 18K		DSP48E		FF		LUT	
	Available 150		Available 120		Available 65200		Available 32600	
Gain	# Used	% Used	# Used	% Used	# Used	% Used	# Used	% Used
0	128	85%	50	41%	7759	11%	12474	38%
10	32	21%	53	44%	8769	13%	14274	43%
16	16	10%	53	44%	8765	13%	14268	43%
24	16	10%	53	44%	8765	13%	14267	43%
32	8	5%	53	44%	8761	13%	14258	43%
40	8	5%	53	44%	8761	13%	14258	43%
48	8	5%	53	44%	8761	13%	14258	43%
56	4	2%	53	44%	8749	13%	14242	43%
64	4	2%	53	44%	8749	13%	14242	43%

TABLE 5.2: Farm ads - Resource usage random projections

felt down by 82% from 85% to only 3%. The positive aspect of that method was the time it took to run, as with the reduction it took only 83K cycles versus the 552K cycles it take a prediction (or to fit) for a normal sample.

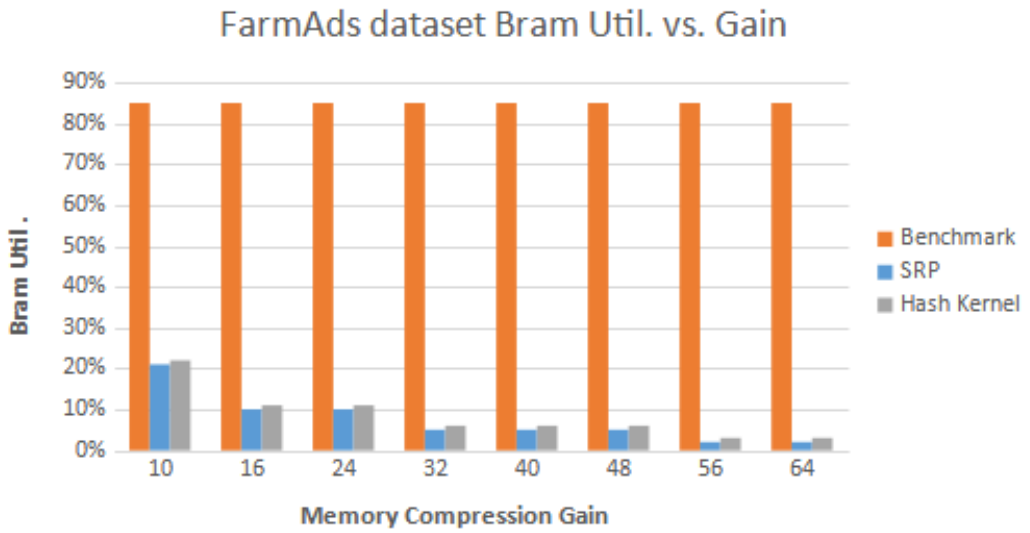

FIGURE 5.3: Farm ads Dataset - Memory comparison

Gain	Feats	NZ	Dim Red (cycles)	Pred (cycles)	Fit Mod (cycles)	Total lat (DR+Fit)	BRAM	Acc
0	54.877	197	-	548.770	552.776	552.718	128	88,91%
10	5.487	191	24.795	54.870	58.947	83.742	33	89,39%
16	3.429	187	22.737	34.290	38.283	61.020	16	87,46%
24	2.286	182	21.594	22.860	26.748	48.342	16	87,71%
32	1.714	177	21.022	17.140	20.923	41.945	8	86,26%
40	1.371	173	20.679	13.710	17.409	38.088	8	85,54%
48	1.143	170	20.451	11.430	15.066	35.517	8	85,54%
56	979	166	20.287	9.790	13.342	33.629	4	86,98%
64	857	163	20.165	8.570	12.059	32.224	4	84,57%

TABLE 5.3: Feature Hashing - Farm adds

	BRAM 18K		DSP48E		FF		LUT	
	Available 150		Available 120		Available 65200		Available 32600	
Gain	# Used	% Used	# Used	% Used	# Used	% Used	# Used	% Used
0	128	85%	50	41%	7759	11%	12474	38%
10	17	11%	74	61%	8719	13%	14147	43%
16	17	11%	74	61%	8719	13%	14147	43%
24	9	6%	74	61%	8712	13%	14131	43%
32	9	6%	74	61%	8712	13%	14131	43%
40	9	6%	74	61%	8712	13%	14131	43%
48	5	3%	74	61%	8705	13%	14118	43%
56	5	3%	74	61%	8705	13%	14118	43%
64	4	2%	74	61%	8749	13%	14242	43%

TABLE 5.4: Farm ads - Resource usage feature hashing

5.3 Gisette

More details about this dataset can be found section 3.3.2 item 2.

The Feature Hashing method, unfortunately, performed poorly when applied to our dataset, resulting in a significant drop in accuracy from 93% to 65%. This loss questions the perceived memory benefits of the Feature Hashing technique. Understanding such limitations is crucial in evaluating the trade-offs between computational efficiency and maintaining high prediction accuracy in our specific dataset.

While the random projections brought about a notable 10-15% reduction in accuracy, the memory footprint show an 8x decrease, and the speed increased approximately 6x after the first iteration. These outcomes, though showing improvements in memory and speed, underscore the challenges encountered when applying random projections to this dataset.

Gain	Feats	NZ	Dim Red (cycles)	Pred (cycles)	Fit Mod (cycles)	Total lat (DR+Fit)	BRAM	Acc
0	5000	649	-	50.000	63.695	63.695	16	92,80%
10	500	498	226.500	5.000	15.524	242.024	2	91,00%
16	312	311	141.336	3.120	9.717	151.053	2	90,00%
24	208	208	187.824	2.080	6.514	194.338	2	85,00%
32	156	156	140.868	1.560	4.902	145.770	2	84,60%
40	125	125	169.125	1.250	3.941	173.066	2	83,60%
48	104	104	140.712	1.040	3.290	144.002	2	80,00%
56	89	89	160.467	890	2.825	163.292	2	77,00%
64	78	78	140.634	780	2.484	143.118	2	77,00%

TABLE 5.5: Rand projection - Gisette

The results for this dataset where not impressive, one of the main reasons we present them is to show that this methods did not work well with high dense data. The density of this set is 0,1297 which means around 1 of 10 values is non zero. In the column NZ we see that in the reductions almost no feature left as zero and we still got only 1-2% accuracy loss. This minimal loss makes the results highly usable, achieving an 8x reduction in total memory usage. Important to note here is that the other resource usage was not changed more than 3% in total making the implementation feasible.

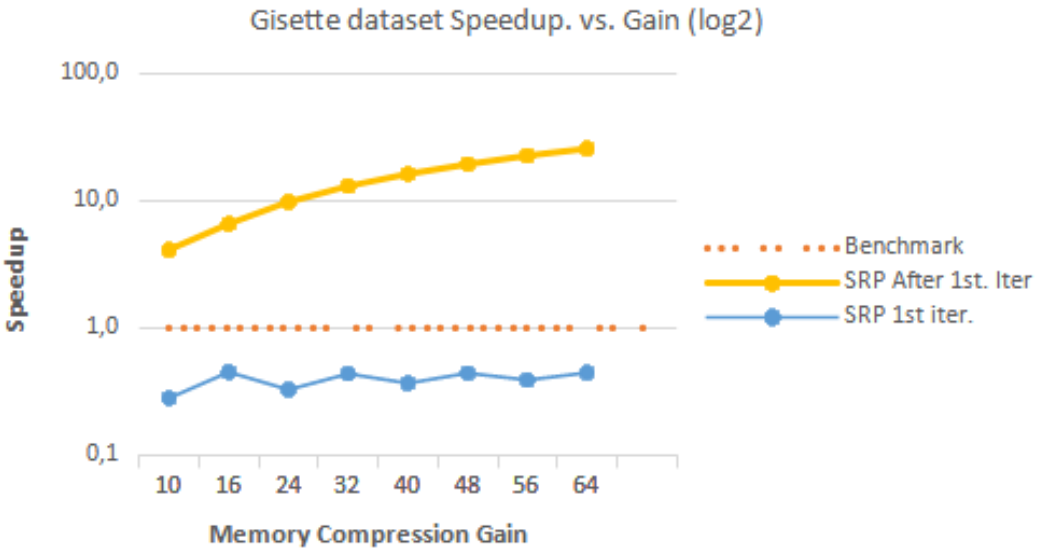

FIGURE 5.4: Gisette Dataset - Speedup comparison

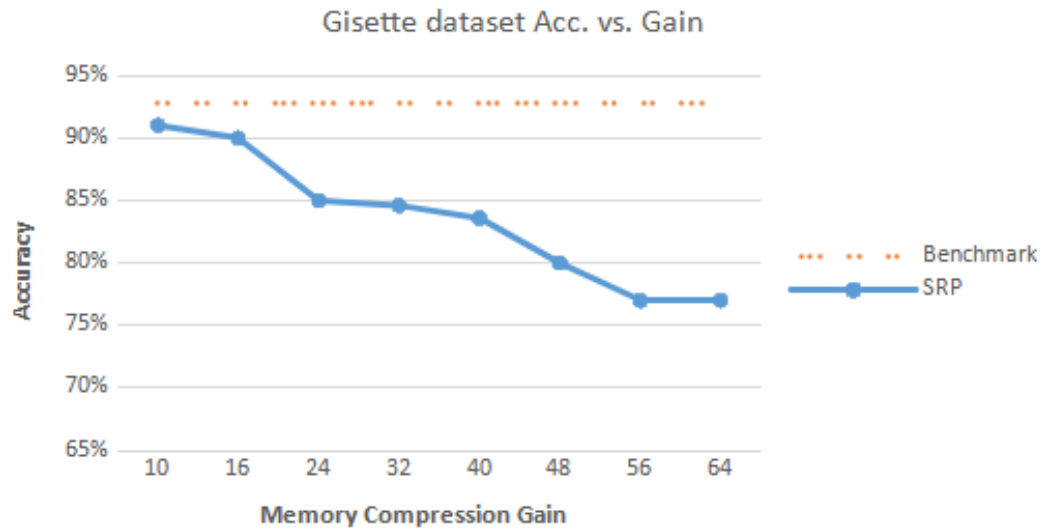

FIGURE 5.5: Gisette Dataset - Accuracy comparison

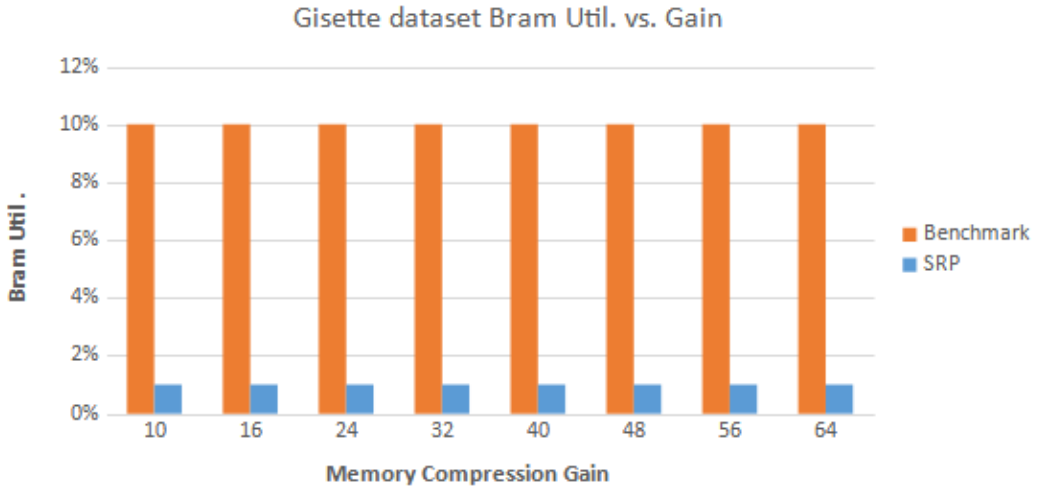

FIGURE 5.6: Gisette Dataset - Memory comparison

	BRAM 18K		DSP48E		FF		LUT	
	Available 150		Available 120		Available 65200		Available 32600	
Gain	# Used	% Used	# Used	% Used	# Used	% Used	# Used	% Used
0	16	10%	50	41%	7755	11%	12453	38%
10	2	1%	53	44%	8753	13%	14233	43%
16	2	1%	53	44%	8749	13%	14230	43%
24	2	1%	53	44%	8745	13%	14224	43%
32	2	1%	53	44%	8742	13%	14220	43%
40	2	1%	53	44%	8739	13%	14218	43%
48	2	1%	53	44%	8737	13%	14215	43%
56	2	1%	53	44%	8735	13%	14212	43%
64	2	1%	53	44%	8732	13%	14210	43%

TABLE 5.6: Gisette - Resource usage

5.4 Dexter

More details about this dataset can be found section 3.3.2 item 3.

For the Dexter dataset we get even an increase in accuracy compared to the reference implementation. The reason for this is that the projection can remove redundant features and thus reduce over-fitting which is caused by multi-collinearity [33]. Our 1st iteration is slower than the reference implementation but we could trade-off this if the we had a big data streaming setting by reducing the density of the random matrix (and thus the complexity of the projection). Increasing the sparsity in the random projection matrix can save computations during the first iteration but may have an additional cost in the classification accuracy if pushed too much.

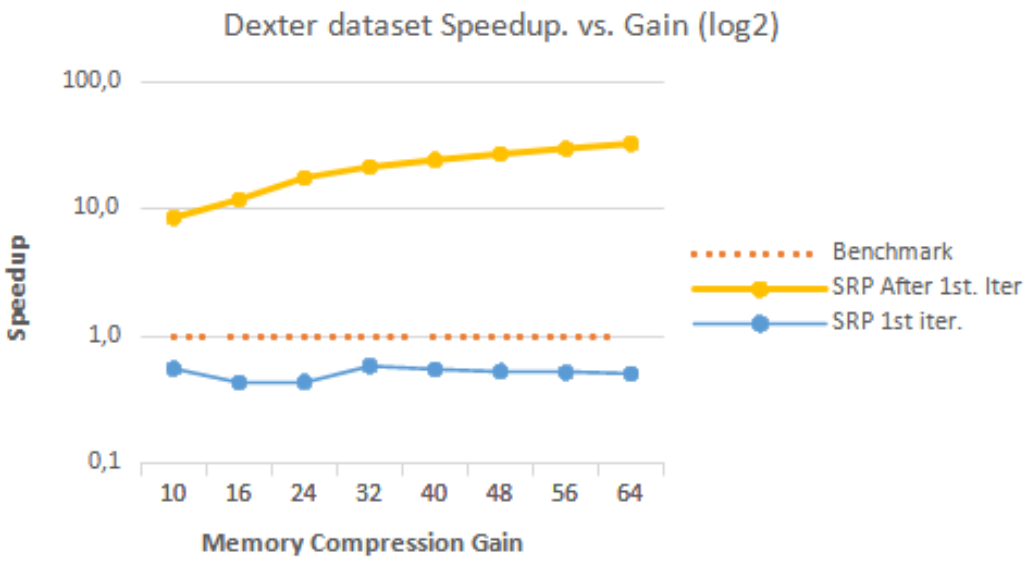

FIGURE 5.7: Dexter Dataset - Speedup comparison

The Hash kernel method does not seem to work very well although the problem is text classification; a possible reason may be that the features are expressed as continuous variables and not as categorical ones (binary features).

For more details on resource usage one can refer to table 5.9 as these two dataset had similar size these two tables were identical.

Gain	Feats	NZ	Dim Red (cycles)	Pred (cycles)	Fit Mod (cycles)	Total lat (DR+Fit)	BRAM	Acc
0	20.000	95	-	200.000	202.061	202.061	64	74,66%
10	2.000	181	366.000	20.000	23.867	389.867	8	79,66%
16	1.250	216	453.750	12.500	17.102	470.852	8	81,00%
24	833	205	452.319	8.330	11.567	463.886	4	75,66%
32	625	151	339.375	6.250	9.487	348.862	2	81,66%
40	500	157	361.500	5.000	8.363	369.863	2	74,66%
48	416	156	375.648	4.160	7.502	383.150	2	76,60%
56	357	152	386.631	3.570	6.828	393.459	2	78,30%
64	312	146	394.056	3.120	6.252	400.308	2	76,00%

TABLE 5.7: Rand projection - Dexter

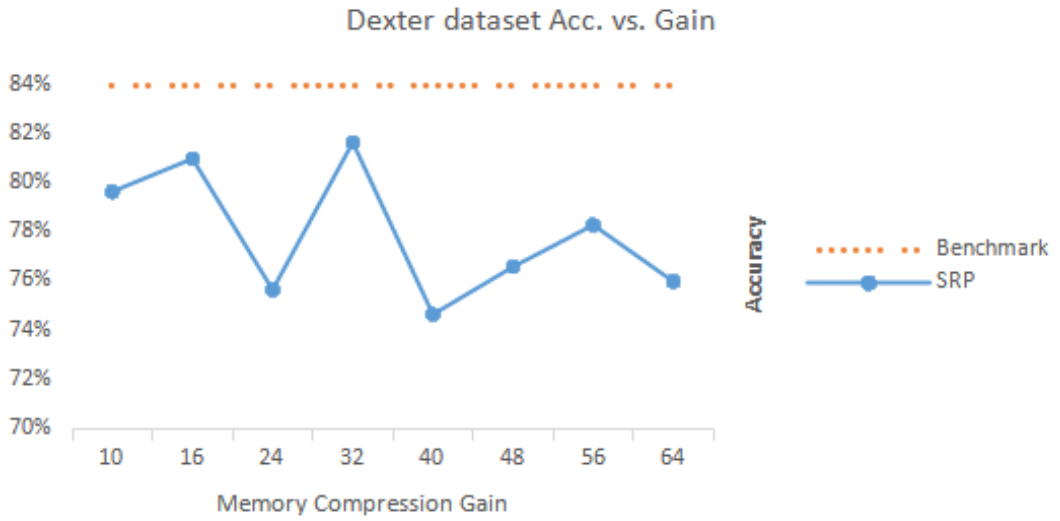

FIGURE 5.8: Dexter Dataset - Accuracy comparison

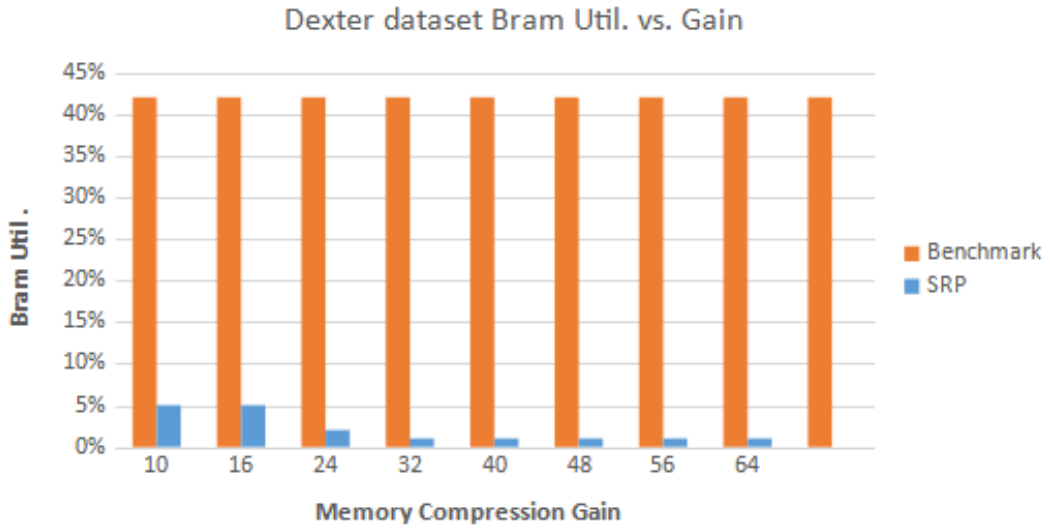

FIGURE 5.9: Dexter Dataset - Memory comparison

5.5 Real sim

More details about this dataset can be found section 3.3.2 item 4.

In this dataset, we observe only a 6% loss in accuracy, accompanied by a substantial reduction in total RAM (64x) and BRAM usage (32x) with the use of random projections. It's crucial to emphasize that despite employing dimension reduction and fitting, the total latency remained consistent with the case of fitting the full set. These observations highlight the trade-offs and considerations involved in choosing dimensionality reduction techniques for this particular dataset. The application of feature hashing resulted in notably poor accuracy.

Gain	Feats	NZ	Dim Red (cycles)	Pred (cycles)	Fit Mod (cycles)	Total lat (DR+Fit)	BRAM	Acc
0	20958	52	-	209.580	210.738	210.738	64	83,00%
10	2095	21	194.835	20.950	21.457	216.292	16	76,64%
16	1309	19	215.985	13.090	13.555	229.540	8	77,16%
24	873	21	199.044	8.730	9.237	208.281	4	78,82%
32	654	19	207.972	6.540	7.005	214.977	4	77,52%
40	523	28	194.556	5.230	5.351	199.907	4	77,28%
48	436	19	205.356	4.360	4.825	210.181	2	76,65%
56	374	22	196.350	3.740	4.268	200.618	2	77,25%
64	327	21	204.048	3.270	3.777	207.825	2	77,15%

TABLE 5.8: Rand projection - Real sim

	BRAM 18K		DSP48E		FF		LUT	
	Available 150		Available 120		Available 65200		Available 32600	
Gain	# Used	% Used	# Used	% Used	# Used	% Used	# Used	% Used
0	64	42%	50	41%	7743	11%	12459	38%
10	16	10%	53	44%	8771	13%	14275	43%
16	8	5%	53	44%	8757	13%	14254	43%
24	4	2%	53	44%	8755	13%	14250	43%
32	4	2%	53	44%	8751	13%	14245	43%
40	4	2%	53	44%	8747	13%	14238	43%
48	2	1%	53	44%	8745	13%	14233	43%
56	2	1%	53	44%	8738	13%	14227	43%
64	2	1%	53	44%	8732	13%	14220	43%

TABLE 5.9: Real sim - Resource Usage Random Projections

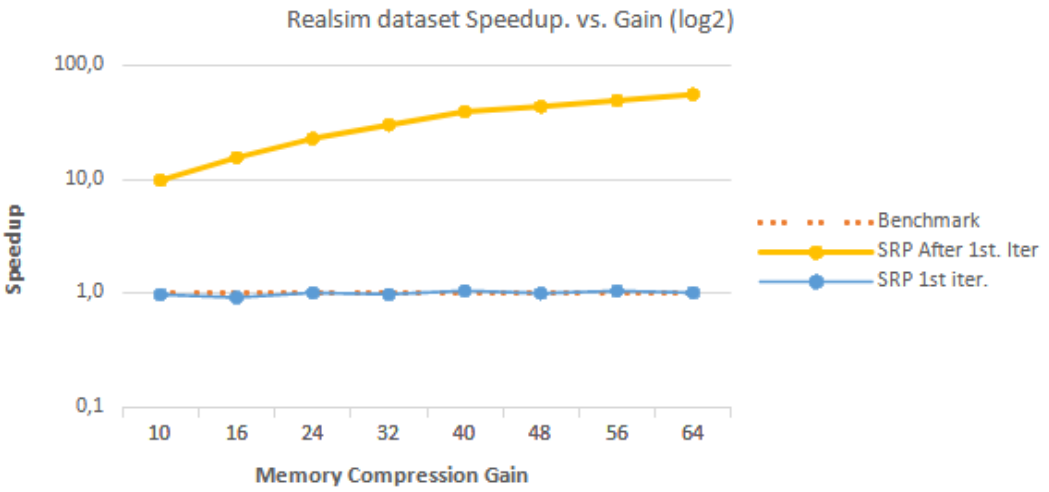

FIGURE 5.10: Real sim Dataset - Speedup comparison

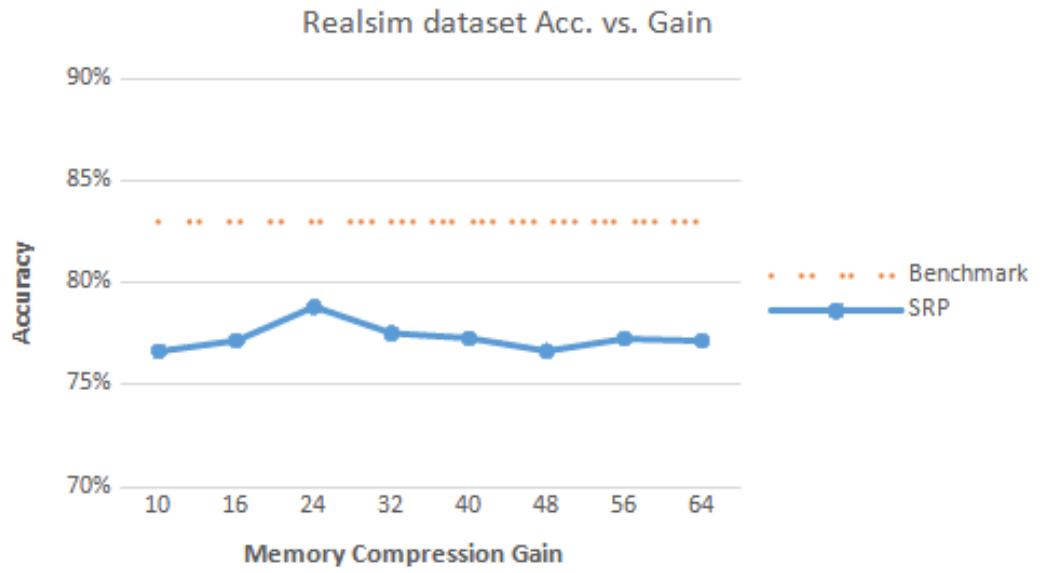

FIGURE 5.11: Real sim Dataset - Accuracy comparison

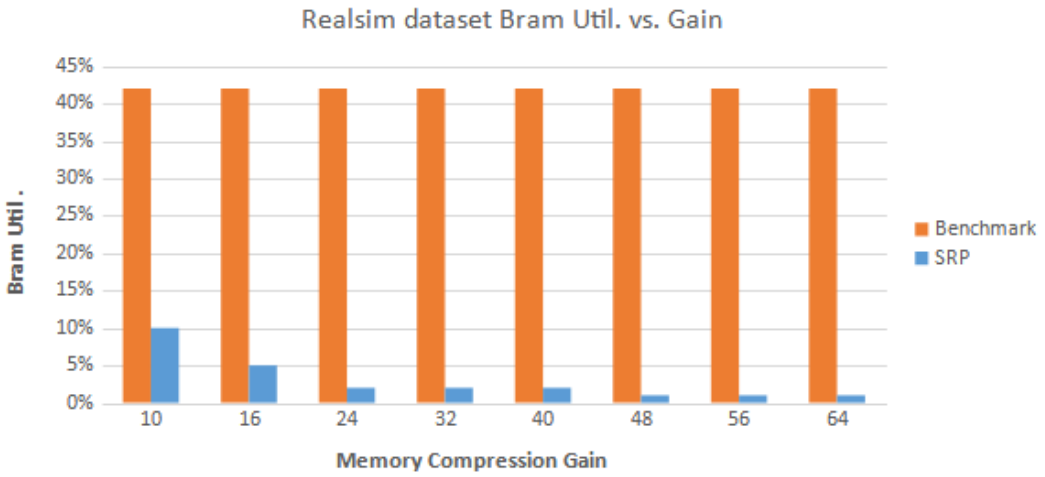

FIGURE 5.12: Real sim Dataset - Memory comparison

5.6 Discussion

As it is observed we managed to significantly reduce the memory usage without consuming lots of others to achieve that. Even the cycles needed to run a reduction process and the a prediction was usually identical to only a prediction with the total number of dataset features. We observed cases that the memory usage did not fall more than 4x which is not considered as impressive as the 64x with only 4% accuracy penalty, although some problems went from been unsolvable to feasible to implement.

In summarizing our findings, it becomes evident that the results have provided us with an advantage, whether in terms of time efficiency or memory utilization. Impressively, these benefits were achieved with only a relatively minor trade-off, highlighting the effectiveness and practicality of the implemented approaches. This delicate balance between improved performance and minimal compromise underscores the significance of our results in enhancing the overall efficiency of the system.

Chapter 6

Conclusion and Future work

6.1 Conclusion

In this study we made some experimental measurements in sparse and very sparse datasets and we proved that due to low entropy of the samples we could use fewer features to take a satisfying result. Also in some cases worked with efficiency by removing bias and giving better results. We analyzed the proposed dimensional reduction algorithms, we presented the module diagrams and we provide the array numbers of the experiments in some common dataset.

Our result is considered to be important not only for the cost or the speed but also for the opportunity to solve problems that could not fit on chip without a reduction in their feature dimension.

Another important aspect is that these techniques could be applied without any knowledge of the data nature like mean value or variance and it does not being affect the outcome, contrariwise, we sometime got a better accuracy. We also conducted an analysis of potential risks arising from the manipulation of data in the absence of adequate domain knowledge. Our contributions in algorithm field is that the proposed technic for Random projections and Feature Hashing run without consuming any memory like other mathematical the implementations so far (PCA family algorithms), with a trade off some computational power though which would arise in any similar algorithm.

6.2 Future Work

As future work, it would worth making a more detailed design in hardware level adding edge cases and designing and getting some results from a chip to demonstrate that what is simulated in Python works on chip as well.

This would not only enrich and expand the existing design but also involve the practical implementation of the proposed enhancements directly on the hardware level, by actually loading on chip the implementation.

A nice extension would be, a simple video game or a smart toy that learns from the player's actions and adapts accordingly. In many games, we're used to having opponents that are not too smart, making it easier for us to win. However, imagine a game where the characters learn and get better as you play.

The challenge becomes more personalized and interesting because the game is adapting to your skills. It's like having a virtual opponent that grows with you, providing a unique and dynamic gaming experience. In this scenario, the technology is applied not just for the sake of being cutting-edge but to enhance the user's enjoyment and engagement by creating a more responsive and challenging environment. This intentional 'dumbing down' due to lower accuracy of the algorithm, allows players to experience both the thrill of overcoming challenges and the satisfaction of winning.

Appendix A

Text data

When a problem nature is known to us and we have the raw data we can handle better the dimensional reduction and propose ways for doing so. We can use some intelligent techniques that developed and estimate how much this will cost us depending our target.

A.1 An overview of hashing for text

Hashing is a very popular technique in solving text based problems by randomly representing words by numbers. For example if the word "book" is the number 1543 and from now the algorithm recognises the number 1543 and not a word. This is just because computers do not understand words and it is a way of representation.

Now one way to work for a specific text is every time we find the word book to increase the attribute R[1543]. The technique is called TF-IDF [34] (Term Frequency - Inverse Document Frequency) and it is approach we use to solve our representation problem. There are others as well but this is probably the most popular and the one it is going to be used and so we are not analysing any other techniques further. With hashing the words into numeric values we create a dictionary, also mentioned as a vocabulary in bibliography. In most common programming languages this is represented with a structure called also map or dictionary having the form of sting as a key and integer as value. In this way we use less estimators compared to the total words of the text since many words are repeated and we just keep the repetition count. An other common technique use to clean up the data is removing the stop words [35]. With that term we mean the words that not give any extra meaning to the text.

An example could be the following: in the phrases "This is a nice book" and "This is a bad book", the words "This", "is", "a" do not give us any context about what we speak of while the words "nice" and "bad" can give us the sentiment of the phrase if it is negative or positive and the word "book" can give us the context of what this sentence is concern of. So we can easily eliminate the three words "This", "is", "a" and not lose a lot of the meaning. Note here that the above technique is not actually lose less as it seems and in more complex text analysis algorithms with RNN [36] every piece of a text is valuable, meaning not only the actual words but the order. A good example here would be the phrase "This is not a bad book", which actually means that this book is a good or at least not a bad one. In the TF-IDF analysis the algorithm will find the words "bad" and "not" so it will most possibly assume a negative sentiment. Although we will successfully in both techniques figure out that the topic is the books we are completely mistaken in the sentiment analysis.

That is a risk which is acceptable in some approaches and problems since the benefits of not using a neural network due to the complexity and the demand on resources of neural network could make such an implementation impossible. It is also worth mentioning that although losses like the one described here can sometimes be critical and affect the outcome, this has only to do with the nature of the problem hence there are cases that this do not affect our performance at all.

A.1.1 Dimensions

The above process of TF-IDF creates something that we called a maps containing only a few words which is called bag-of-words a very common problem one can notice here is that a new word may come that we have never seen again. That should create a new entry in our vocabulary but this causes a problem because our model which we analyse in the next section, is not trained in this word. So we have to pause the process retrain the model and then continue. To avoid this there are two ways. One is to use a preset dictionary containing as much words as possible, but this will end up in a very big and sparse matrix where lots of values will be zero (0) since many words are not used. Image that a common English dictionary has around 100.000 word while an easy is usually limited by word count which can be around 4000 words with many duplicates.

A.1.2 N-gram models

Another thing is worth mentioning are the n-gram models. The above described way to work is considering each word as an entity. There is an approach to use more than one words as an entity. That could solve some problems as the "not bad" is now a new word that does not have a negative sentiment and the algorithm could distinguish. It creates some other though like the low frequency of appearance for such terms and the extremely higher dimension of features for any text problem. Of course if we go with this option we are not considering all the English dictionary but we make the extraction form the training text.

A.1.3 Outcome for text based data

In cases where the raw data are available as we analysed here there are ways to preprocess the data and lower the dimensions. So for a text based model we could have a sparse array of 100.000 features to a dense array of 1000 features in case of choosing 1 word model (1-gram). In cases with N-gram with N>2 this feature count is not easily predictable but it will for sure be very sparse.

Appendix B

Activation function

One of the things that came up while developing the algorithm and writing the code is the need of an activation function. First of all we need to define what an activation function is. In the literature, the activation function is often referred to as a mathematical "gate" positioned between the input received and its output. We are not going to analyse all the available functions but only three of them that are the most commonly used and those who we made some experiments.

These are the:

- Sigmoid function (also known as the Logistic function): It is one of the most widely used activation function. It maps the input to a value between 0 and 1, which is useful for binary classification problems.

  It is defined as:

  $\sigma(x) = \frac{1}{1+e^{-x}}$

  Some characteristics are the typical S-shaped curve. Its output is centered at 0.5 and ranges from 0 to 1. This function is differentiable, allowing us to calculate the slope of the sigmoid curve at any given points.

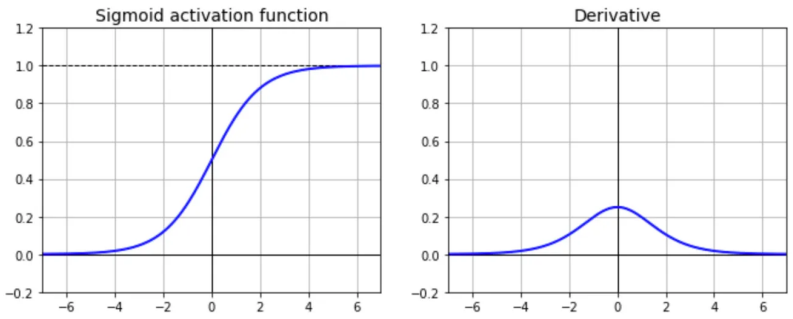

FIGURE B.1: Activation Function: Sigmoid Graph

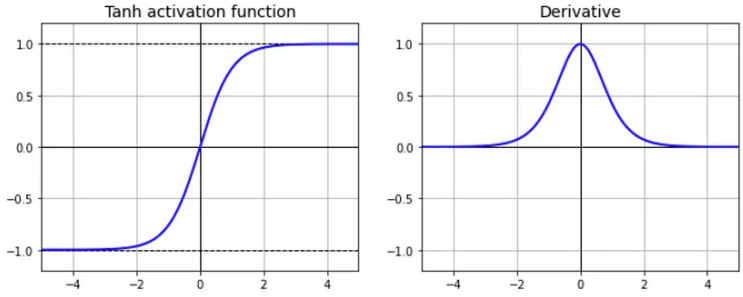

FIGURE B.2: Activation Function: tanh Graph

Although the function itself is monotonic, its derivative is not.

- Tanh (Hyperbolic Tangent): Similar to the sigmoid function, exhibits a typical S-shaped curve. However, the key distinction is that the output of the tanh function is zero-centered, spanning from -1 to 1, unlike the sigmoid function that ranges from 0 to 1

  It is defined as:

  $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

  It is differentiable, allowing us to find the slope of the curve at any given points. Additionally, both functions are monotonic, meaning they always increase or always decrease, but their derivatives are not monotonic. An essential aspect worth noting is that Tanh has a tendency to center output around 0, which can significantly accelerate the convergence of the learning process. This feature proves to be beneficial in various neural network architectures, aiding in more efficient optimization during training.

  A very negative thing which in case our experiments show any benefits from using that function is the it has an exponential operation. With an FGPA implementation this could be a huge speed up in a computetional problem.

- ReLU (Rectified Linear Unit): It sets all negative values to zero and leaves positive values unchanged, making it computationally efficient.

  $$\begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

The graphs and some details for the above are been referenced in [37]

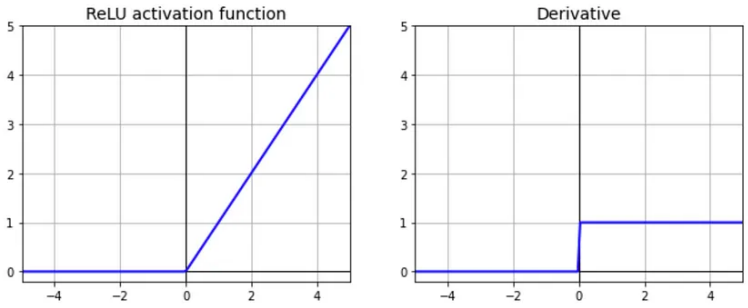

FIGURE B.3: Activation Function: ReLu Graph

Appendix C

Hyperparameters

In machine learning, hyperparameters are external configuration settings that are not learned from the data but are set before the training process begins. In statistics and statistical learning a hyperparameter is defined simply as the fixed parameter to your prior probability function.

These parameters influence the overall behavior of a model, affecting its learning process and performance. Unlike model parameters, which are learned from the training data, hyperparameters are set manual.

Examples of hyperparameters include:

- Learning Rate
- Number of Epochs
- Number of Hidden Layers and Neurons in a Neural Network
- Number of Trees, Depth and branches on Decision Tree algorithms
- Batch Size: The batch size represents the number of training samples used in each iteration.
- Activation Function
- C (Cost) and sigma in Support Vector Machines (SVM):

  The regularization parameter C controls the trade-off between achieving a low training error and a low testing error. The σ parameter is associated with the choice of the kernel function in SVM
- Epsilon in Support Vector Machines (SVM):

  Epsilon is a hyperparameter that determines the margin of error accepted in the optimization process.

- Number of clusters in a clustering algorithm: Example the k in k-nearest neighbors

Tuning hyperparameters is a crucial step in the machine learning workflow, as it can significantly impact the model's performance and generalization ability.

References

[1] H. Brendan McMahan, Gary Holt, and Google Inc. "Ad click prediction a view from the trenches". In: *KDD '13: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* 62 (Aug. 2013), pp. 1222–1230. URL: https://doi.org/10.1145/2487575.2488200.

[5] Mitra P. Caragea C Silvescu A. "Protein sequence classification using feature hashing". In: *Proteome Sci* (June 2012). DOI: https://doi.org/10.1186/1477-5956-10-S1-S14.

[6] Qinfeng Shi et al. "Hash Kernels for Structured Data". In: *Journal of Machine Learning Research* (Oct. 2009), pp. 2615–2637. URL: https://www.jmlr.org/papers/volume10/shi09a/shi09a.pdf.

[7] Seker Sadi and Mert Cihan. "A Novel Feature Hashing for Text Mining". In: *Journal of Technical Science and Technologies* 2 (June 2013), pp. 37–40. URL: https://www.researchgate.net/publication/257652600_A_Novel_Feature_Hashing_for_Text_Mining.

[8] Kilian Weinberger et al. "Feature Hashing for Large Scale Multitask Learning". In: *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning* (June 2009), pp. 1113–1120. URL: https://dl.acm.org/doi/10.1145/1553374.1553516.

[9] Dimitris Achlioptas. "Database-friendly random projections: Johnson-Lindenstrauss with binary coins". In: *Journal of Computer and System Sciences 66 (2003) 671–687* 66 (July 2002), pp. 671–687. URL: http://www.elsevier.com/locate/jcss.

[10] Sanjoy Dasgupta. "Experiments with Random Projection". In: *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence* (Jan. 2013). URL: https://doi.org/10.48550/arXiv.1301.3849.

[11] Shai Shalev-Shwartz. "Online Learning and Online Convex Optimization". In: *Foundations and Trends® in Machine Learning* 4.2 (2012), pp. 107–194. ISSN: 1935-8237. DOI: 10.1561/2200000018. URL: http://dx.doi.org/10.1561/2200000018.

[12] George Papamakarios. “Comparison of Modern Stochastic Optimization Algorithms”. In: (2014).

[14] Jonas Stenbæk Hegner, Joakim Sindholt, and Alberto Nannarelli. “Design of power efficient FPGA based hardware accelerators for financial applications”. In: (2012), pp. 1–4. DOI: 10.1109/NORCHP.2012.6403096.

[23] Mahima Singh and Deepak Garg. “Choosing Best Hashing Strategies and Hash Functions”. In: *2009 IEEE International Advance Computing Conference* (2009), pp. 50–55. DOI: 10.1109/IADCC.2009.4808979.

[33] Roy E. Welsch David A. Belsley Edwin Kuh. “Regression Diagnostics: Identifying Influential Data and Sources of Collinearity”. In: (1980). DOI: 10.1002/0471725153.

External Links

[2] "numpy.linalg.svd". In: (). URL: https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html.

[3] "sklearn.decomposition.TruncatedSVD". In: (). URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html.

[4] "sklearn.decomposition.PCA". In: (). URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[13] "Why Do Machine Learning Models Die In Silence?" In: (). URL: https://www.kdnuggets.com/2022/01/machine-learning-models-die-silence.html.

[17] "MLOps Professional". In: (). URL: https://learning.intel.com/Developer/pages/133/mlops-professional.

[18] "LIBSVM Data: Classification (Binary Class)". In: (). URL: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

[19] "UC Irvine Machine Learning Repository". In: (). URL: http://archive.ics.uci.edu/datasets.

[20] "Communication Performance of PAM vs. QAM Handout". In: (). URL: https://users.ece.utexas.edu/~bevans/courses/rtdsp/handouts/PAMvsQAMHandout.pdf.

[21] "Methods for handling missing values". In: (). URL: https://gallery.azure.ai/Experiment/Methods-for-handling-missing-values-1.

[22] "Cross-validation, stratified-k-fold". In: (). URL: https://scikit-learn.org/stable/modules/cross_validation.html#stratified-k-fold.

[24] "Python extension for MurmurHash (MurmurHash3), a set of fast and robust hash functions." In: (). URL: https://pypi.org/project/mmh3/.

[25] "Built-in Functions Hash". In: (). URL: https://docs.python.org/3/library/functions.html#hash.

[26] "Random Projections Sparse sklearn". In: (). URL: https://scikit-learn.org/stable/modules/generated/sklearn.random_projection.SparseRandomProjection.html.

[27] "XC7A100T-1CSG324C". In: (). URL: https://www.xilinx-adm.com/XC7A100T-1CSG324C.htm?gad_source=1.

[28] "C port of Murmur3 hash". In: (). URL: https://github.com/PeterScott/murmur3.

[31] "lfsr Documentation". In: (). URL: https://media.readthedocs.org/pdf/pylfsr/latest/pylfsr.pdf.

[32] "Linear Feedback Shift Registers in Virtex Devices". In: (). URL: https://docs.xilinx.com/v/u/en-US/xapp210.

[34] "Understanding TF-IDF for Machine Learning". In: (). URL: https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/.

[35] "Dropping common terms: stop words". In: (). URL: https://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html.

[36] "What are recurrent neural networks?" In: (). URL: https://www.ibm.com/topics/recurrent-neural-networks.

[37] "7 popular activation functions". In: (). URL: https://towardsdatascience.com/7-popular-activation-functions-you-should-know-in-deep-learning-and-how-to-use-them-with-keras-and-27b4d838dfe6.