POLITECNICO DI TORINO

Department of Electronics and Telecommunications — DET

Ph.D. in

ELECTRICAL, ELECTRONICS AND COMMUNICATIONS ENGINEERING

Cycle XXXV

Settore Concorsuale: 09/E3 — ELETTRONICA

Settore Scientifico Disciplinare: ING-INF/01 — ELETTRONICA

METHODS AND APPLICATIONS FOR LOW-POWER DEEP NEURAL NETWORKS ON EDGE DEVICES

Presented by:
Luciano PRONO

Ph.D. Coordinator:
Prof. Stefano GRIVET TALOCIA

Supervisor:
Prof. Gianluca SETTI

Final Exam Year 2022

POLITECNICO DI TORINO

Abstract

Department of Electronics and Telecommunications (DET)

Doctor of Philosophy

Methods and Applications for Low-Power Deep Neural Networks on Edge Devices

by Luciano PRONO

Machine Learning (ML) techniques introduce the possibility of solving many complex tasks requiring humongous amounts of data. In particular, Deep Neural Networks (DNNs) are structures that provide excellent results with the use of a massive number of trainable parameters. At the same time, the implementation of DNNs on cheap and low-power devices with high resource constraints, referred to as tinyML, would unlock a large number of entirely new applications possible in the fields of Internet of Things (IoT) and Edge Computing. In this dissertation, we focus on the design of low-power, low-memory DNN structures and we implement them with novel Edge Computing approaches such as an innovative Compressed Sensing (CS) DNN-based decoder.

In the first part, we focus on the methods and architectures for the design of low-power and tiny DNN structures. Firstly, we introduce parameters quantization and interconnection pruning methods. Secondly, we present two novel map-reduce alternatives to Multiply and Accumulate (MAC) paradigm, namely Sum and Max (SAM) and Multiply and Max-Min (MAM²), and we use them to build DNN layers that are highly resistant to the pruning process. Additionally, we present an on-device implementation of a MAM² layer, where the implementation with standard MAC-based layers would not be possible. Finally, we present an on-device implementation of a Recurrent Spiking Neural Network (RSNN), intrinsically low-power due to its sparsity properties.

In the second part, we introduce a novel CS decoder based on DNNs. Firstly, we describe the novel encoding-decoding CS framework, with the decoding phase split in two distinct working blocks, namely support retrieval and signal reconstruction. This novel approach outperforms state-of-the-art techniques in terms of quality of reconstruction. Finally, we apply some of the methods of the first part of this dissertation to study how this structure can be compressed to reduce the hardware requirements, showing that this novel approach to CS needs very few resources.

Contents

Abstract iii

Introduction 1

I Low-power Architectures and Non-conventional efficient paradigms for DNNs 3

1 Introduction to DNNs model compression 5
1.1 DNN quantization . . . 5
1.1.1 Quantization-aware training . . . 6
1.1.2 Limiting the parameters range . . . 7
1.1.3 Limiting the activations range . . . 7
1.2 A Gardener's Job: DNN pruning . . . 8
1.2.1 One-shot pruning techniques . . . 9
1.2.2 Iterative pruning techniques . . . 10
1.3 Chapter conclusion . . . 11

2 Avoiding the Multiplications: the Sum and Max (SAM) paradigm 13
2.1 SAM-based DNN layers . . . 13
2.2 Performance and cost . . . 16
2.2.1 Classification task . . . 17
2.2.2 DNN configuration . . . 17
2.2.3 Training and performance . . . 17
2.2.4 Optimization by pruning and cost . . . 19
2.3 Chapter conclusion . . . 22

3 A Gardener's Delight: the Multiply and Max&Min (MAM²) paradigm 23
3.1 The MAM²-based layer architecture . . . 23
3.1.1 Layer description . . . 23
3.1.2 A bridge from MAC to MAM²: the vanishing contributes . . . 24
3.2 MAM² training method and pruning: a toy case . . . 25
3.2.1 Training method . . . 26
3.2.2 The natural sparsity of MAM² layers . . . 26
3.3 Performance and cost . . . 29
3.3.1 Model architectures . . . 29

3.3.2 Datasets . . . 29
3.3.3 MAM² performance: one-shot pruning . . . 32
3.3.4 MAM² performance: iterative pruning . . . 37
3.4 Case study for implementation: autoencoder for ECG signals . . . 39
3.4.1 Autoencoder architecture . . . 39
3.4.2 Dataset . . . 40
3.4.3 Quantization and training . . . 40
3.4.4 Pruning the autoencoder . . . 41
3.5 Hardware implementation of MAM² based layer . . . 42
3.5.1 Sparse Matrix Representation . . . 42
3.5.2 On-device implementation . . . 43
3.6 Chapter conclusion . . . 45

4 A sparse approach to data: implementation of a Spiking Neural Network 47
4.1 Event-based data: an overview . . . 47
4.2 Recurrent SNN architecture . . . 48
4.2.1 Model description . . . 48
4.3 Dataset and Performance . . . 50
4.3.1 N-MNIST event-based dataset for digit recognition . . . 50
4.3.2 Training setup and preliminary performance . . . 51
4.4 Implementation and performance . . . 52
4.4.1 Implementation on MCU . . . 52
4.4.2 Performance . . . 52
4.5 Chapter conclusion . . . 54

II An Edge Computing Application for DNNs: Decoder for Compressed Sensing 55

5 Compressed Sensing Decoder based on Deep Neural Oracles 57
5.1 Compressed Sensing basics . . . 60
5.1.1 Rakeness CS . . . 61
5.2 Bio-signal datasets and performance indexes . . . 62
5.2.1 Synthetic ECG . . . 62
5.2.2 Synthetic EEG . . . 62
5.2.3 Performance indexes . . . 63
5.3 Signal encoding and pros/cons of short windows . . . 63
5.4 A two-step trained structure for signal reconstruction: TCSSO decoder . . . 65
5.4.1 Oracle training and decoder operations . . . 67
5.4.2 Reconstruction performance . . . 70
5.4.3 Preliminary evidence on real signals . . . 72
5.5 Decoder self-assessment . . . 73
5.6 Computational requirements . . . 77

5.6.1 Encoder . . . 77
5.6.2 Decoder . . . 78
5.7 Dealing with the flawed sparsity of realistic signals: TSOC decoder . . . 79
5.7.1 Support Oracle for Compressed Signals . . . 80
5.7.2 Training the support oracle . . . 81
5.7.3 Results for ECGs . . . 83
5.8 Chapter conclusion . . . 85

6 Reducing the Complexity of the Deep Neural Oracles 87
6.1 Pruning the oracle: a MAM² application . . . 87
6.1.1 Oracle DNN configuration . . . 87
6.1.2 Oracle DNN training . . . 88
6.1.3 Performance and size . . . 89
6.2 Post-training quantized TCSSO . . . 90
6.2.1 Support Oracle with fixed-point precision . . . 92
6.2.2 Pseudoinverse operation with fixed-point precision . . . 92
6.2.3 Full decoder with fixed-point precision . . . 92
6.3 Quantization-aware trained TSOC and implementation . . . 95
6.3.1 The quantization-aware trained oracle . . . 96
6.3.2 The LMS-based reconstructor . . . 97
6.3.3 Architecture design and results . . . 98
6.4 Chapter conclusion . . . 102

Conclusion 103

Bibliography 105

Introduction

With the exponential increase in quantity and complexity of the available data for the most diverse applications, Machine Learning (ML) provides effective ways of employing such information for many different interesting tasks [1, 2, 3]. ML is the subset of Artificial Intelligence (AI) dealing with the design of systems able to learn and improve their performance based on the statistics of the employed data. Common examples of ML structures – even if not remotely sufficient to frame the whole field – are Deep Neural Networks (DNNs), whose growing popularity is justified by the excellent learning capabilities and computational performance [4].

Typical applications of DNNs are in the field of Internet of Things (IoT) [5]. IoT is referred to as the paradigm composed of many interconnected objects, i.e., things which communicate in the Internet, where each element can be a sensor node, a computational unit, a storage device and so on. In conventional IoT systems, data is acquired on site with sensor nodes and the fully uploaded to a central node – which we may refer to as a server – that stores and elaborates it.

This simple acquire-and-send philosophy is sometimes critical for different reasons. In particular, due to the high volumes of transmitted data, transmission requires large bandwidths and an excellent stability to avoid data losses, which is, in most cases, impractical. Then, once data is transmitted, storage and computation operations introduce high maintenance costs on the central node. Additionally, in the case of real-time applications, the delay due to decentralized elaboration could render this setup unfeasible. Finally, many applications requires a certain degree of privacy and security that often cannot be guaranteed when data is transmitted often back and forth from sensor nodes to centralized severs.

One methodology that partially solves the aforementioned matters could lie in the compression of the acquired data right after its acquisition. An extreme example of this is the Compressed Sensing (CS) paradigm [6, 7, 8], that unifies the acquisition and compression of data by sampling signals in an already compressed domain.

Conversely, a more general approach consists in moving the processing of data from the central node to the sensor nodes i.e., data is processed right after its acquisition. This philosophy takes form in the Edge Computing paradigm [9]. In particular, with reference to the implementation of ML on low-end, cheap devices that drain less than $1\,\mathrm{mW}$ of power, a fast growing field is tinyML [10, 11].

The focus of this dissertation is on the implementation of low-power tiny DNN structures to be fit on tinyML devices, in order to exploit them for IoT and Edge Computing applications. In particular, we test the compression of a DNN structure on a peculiar case studio, i.e., a

novel DNN-based decoder for CS used to acquire biomedical signals. The dissertation is organized as follows:

Part I revolves around the model compression techniques and DNN architectures that can be used to implement neural network structures on low-end, low-power devices with high resource constraints.

In Chapter 1 state-of-the-art model compression techniques, namely quantization of the parameters and pruning of the interconnections, are introduced. The contents of this introductory chapter are recalled thorough any of the remaining chapters in the dissertation.

The novel Sum-and-Max based map-reduce paradigm, used to build multiplier free DNN layers, is introduced in Chapter 2. The structure is proven to work on a simple classification task and, more interestingly, is found to be highly prunable.

Following the example of the Sum-and-Max paradigm, in Chapter 3 the Multiply-and-Max&Min paradigm is introduced, with performance comparable to standard Multiply and Accumulate structures but highly resistant to the pruning process. This structure is tested on big-size standard image-vision datasets and networks. Additionally, an on-device implementation of a MAM^2-based pruned DNN autoencoder is tested and analyzed.

Finally, Chapter 4 completes the introduction of non-standard MAC-based DNNs by presenting an on-device Recurrent Spiking Neural Network (RSNN), whose intrinsic sparse characterization both in space and time allows a big reduction in terms of energy consumption of the structure.

Part II introduces a novel approach to the decoding phase of the CS paradigm which uses a DNN structure. The aim of the part is the presentation of the novel CS decoder and the assessment of its reconstruction performance on bio-signals, even when compressed by means of parameters quantization and pruning.

In Chapter 5, the novel decoder, composed of two distinct working blocks, namely a DNN-based Support Oracle and a pseudoinverse-based reconstruction block, is described and its performance is assessed with synthetic bio-signal datasets. Two different versions of the decoder are presented, namely the Trained CS with Support Oracle (TCSSO), which trains both the encoding and decoding phases together to recover sparse signals, and the Trained Support Oracle for Compressed signals (TSOC), that introduces a preliminary phase to maximize the performance with signals that are not perfectly sparse.

In Chapter 6, the techniques and special architectures introduced in Part I are applied to the CS decoder structure, and the performance with the compressed models is assessed. In particular, MAM^2 layers are used to prune the support oracle in TCSSO, a preliminary post-training quantization analysis is performed on the TCSSO structure and finally the improved TSOC structure is fully quantized using quantization-aware techniques.

Part I

Low-power Architectures and Non-conventional efficient paradigms for DNNs

Chapter 1

Introduction to DNNs model compression

In order to use DNNs to solve complex tasks on low-end, strongly resource constrained devices, such as Microcontroller Units (MCUs), Field Programmable Gate Arrays (FPGAs) or more in general low-power custom hardware accelerators in the form of Application Specific Integrated Circuits (ASICs), the size and computational complexity of these structures need to be minimized.

This can be achieved by using model compression techniques as well as with the use of non-standard architectures, such as Spiking Neural Networks (SNN) or ad-hoc alternatives to the classic Multiply and Accumulate (MAC) map-reduce paradigm.

In particular, in this chapter we introduce quantization of data and parameters as well as interconnections pruning, two DNN model compression methods that we employ multiple times in the following chapters of this dissertation.

1.1 DNN quantization

The search for small and portable DNN structures can be addressed by employing quantized parameters and data in the implementation, which allows a reduction of the memory requirements, a reduction of the latency due to the computation and an overall reduction of the energy consumption of the system [12]. Notably, this can even be brought to extremes: two examples of this are binary neural networks [13, 14], that use 2's complement operations instead of multiplications, and XNOR-net (and more recently, XOR-net) [15, 16], an extreme quantized version of classic DNNs that uses XNOR gates instead of the expensive multiply operations and employs only 1-bit boolean values.

Quantization of data and parameters in DNNs fundamentally consists in constraining the allowed values in the network to a few pre-defined levels. Typically, from full-precision DNN models, data is uniformly quantized to a pre-defined set of values with the following operation

$$x_{\text{quant}} = \Delta \left\lceil \frac{x}{\Delta} \right\rfloor \tag{1.1}$$

where $x \in [-q, q)$ is the full-precision value, Δ is the quantization step, $\lceil \cdot \rfloor$ is the rounding operation and x_{quant} is the quantized value. If x is not comprised in the interval $[-q, q)$, it can be either re-scaled beforehand or clipped afterwards. The quantization step can be defined as

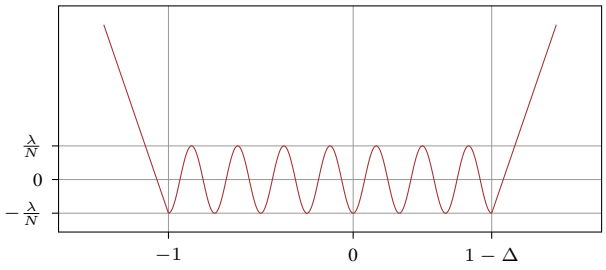


FIGURE 1.1: Plot of the cosine regularization function for quantization-aware training along with the "bathtub" regularization which keeps the values inside the desired working interval.

$\Delta = \frac{2q}{N}$, with N the number of allowed quantized levels. Typically, data is quantized in order to be employed with fixed point representation and $N = 2^{n_b}$, where n_b is the number of bits used for the representation.

1.1.1 Quantization-aware training

While training a DNN with the aim of fitting it on low-end devices with quantized parameters, it is fundamental to consider the quantization error introduced in order to maximize the performance. This means that training needs to be *quantization-aware*, i.e., special methods are used both to consider the quantization error during the update of the parameters and to promote the discretization of the values. Among the many solutions proposed in the literature, we investigate two techniques, namely fake-quantization [17, 12] and cosine regularization [18].

Fake-quatization Fake-quantization [17, 12] suggests to train the network with full precision parameters and to adopt quantized values only during the feed-forward phase. This has the effect of emulating the loss of precision due to quantization while retaining the possibility of performing small updates (below the value of Δ) to the weights, which is fundamental for the convergence of the training process.

Cosine regularization Cosine regularization [18] technique consists in the addition of regularization term $R_{\cos}(\mathcal{W})$ added to the training loss function and defined as follows

$$R_{\cos}(\mathcal{W}) = -\sum_{w \in \mathcal{W}} \frac{\lambda}{N} \cos(Nw\pi) \tag{1.2}$$

where $\mathcal{W}$ is the set of all the weights in the DNN, $w \in [-q, q)$ is each weight in the DNN and λ is a parameter that defines the strength of the regularization. As a consequence, the parameters are pushed near the N values allowed by the quantizer and, therefore, quantization error is reduced. A visual representation of the cosine regularization can be seen in Figure 1.1.

1.1.2 Limiting the parameters range

In many applications, the range of data and parameters cannot be effectively defined as the maximum and minimum of their set of values. In most cases, the values are not known beforehand so the use of maximum and minimum to select the range $[-q, q)$ for the values is actually not feasible. For example, in the field of compressed sensing (CS) acquired and compressed data can be considered as a zero mean Gaussian distributed random variable, so setting a conversion range that includes all possible values is not possible, or simply not convenient [19, 20].

For example, being σ_x^2 the variance of the distribution of each value and γ a positive coefficient, it is possible to set $q = \gamma\sigma_x$, so that each entry of x ranges from $x_{\text{min}} = -q = -\gamma\sigma_x$ to $x_{\text{max}} = q - \Delta \approx \gamma\sigma_x$. After quantization, each element of x can be considered a fixed point number belonging to the set $\Sigma_x = \{-q, -q + \Delta, \dots, q - \Delta\}$. In this way it is possible to change γ to modify the range for x_{quant}, in order to find the best compromise between the number of values that are clipped and the quantization step (i.e., precision) Δ.

Additionally, when updating the DNN parameters during training, there is no control on their range, i.e., the magnitude of some parameters can grow too large compared to the desired range. To avoid this, it is possible to force the parameters to assume values in the range $[-q, q)$ with the combination of two methods, namely "bathtub" regularization and parameter "recycling" [21].

Bathtub regularization The "bathtub" regularization consists in a regularization term that is added to the cost function and penalizes the parameters that are outside the desired range. The "bathtub" regularization function is easily defined with its derivative

$$\frac{\partial R_{\text{bathtub}}(w)}{\partial w} = \begin{cases} -q & \text{for } w < -q \\ 0 & \text{for } -q \leq w \leq q - \Delta \\ q & \text{for } w > q - \Delta \end{cases} \quad \text{for } w \in \mathcal{W} \tag{1.3}$$

where w is each parameter in the set $\mathcal{W}$ of all the parameters in the DNN. This regularization term affects the cost gradient that is used to update the parameters in the back-propagation algorithm and its effect consists in pushing the parameters value inside the desired set. In Figure 1.1 the "bathtub" regularization is shown combined with the cosine regularization.

Parameter recycling Parameter "recycling" is applied during training at the end of each training epoch. In order to keep the parameters inside the desired interval, any parameter that has a value outside the range $[-q, q)$ are re-initialized randomly.

1.1.3 Limiting the activations range

When working with quantized DNNs, a special attention has to be paid to the activations, i.e., the outputs of each DNN layer. It is difficult to know beforehand which will be the range of these values, so it is possible to artificially force these values in a range $[0, p)$. To do so, for

all hidden layers, the Saturated ReLU (SReLU) – in place of the standard ReLU function – is employed as activation function. SReLU is defined as

$$\mathrm{SReLU}(v) = \begin{cases} 0 & \text{for } y < 0 \\ y & \text{for } 0 \leq y < p \\ p & \text{for } y \geq p \end{cases} \tag{1.4}$$

where y is the output of the neuron before the activation function is applied. As a further advantage, ReLU and SReLU outputs only positive values, removing the need of memorizing the sign of the activations.

1.2 A Gardener's Job: DNN pruning

Given a total number of weights (i.e., interconnections between layers), *pruning* refers to the removal of a certain number of parameters while minimizing the effects on the accuracy of the DNN. The problem of DNN pruning has been addressed in multiple ways and many solutions have been proposed in the literature [22, 23].

The most simple approaches are performed post-training, without further fine-tuning, in a non-structured fashion. With these one-shot methods, a score is given to each parameter and then the parameters with the lowest scores are removed, without looking at the spatial disposition of the kept interconnections.

Many works apply iteratively the aforementioned one-shot methods in order to achieve better results. In [24], authors retrain iteratively the DNN, removing each time a growing number of parameters, selected among the ones with the smallest absolute values. The key concept here is using always the original set of starting weights for each training iteration.

Also, some works remove weights layer by layer [25] while others [24] generate weight scores based on the global state of the network.

Some works leverage on the possibility of predicting the interconnections to be pruned before training. Authors in [26] explain how it is possible to select the most relevant parameters in the DNN by looking at their *connection sensitivity* (i.e., the influence of an interconnection to the output) or by looking at their gradient, before training is performed.

Many works use regularization terms in the cost function during training in order to encourage the sparsity in the DNNs, as in [27].

Also, some work use advanced integrated frameworks to prune the network. In [28] the training process is split in two parts in order to impose a sparsity condition to the parameters matrix in the DNN and the resulting two problems are solved iteratively through stochastic gradient descent (SGD) and in an analytical form, respectively.

In [29], authors take into account the hardware implementation combining binary DNNs with pruning, while in [30] authors start from an already sparse architecture adding and removing interconnections where needed.

Finally, in an attempt of obtaining pruned DNNs which are hardware friendly, many works prune the parameters in structured ways, looking at the spatial disposition of the kept

interconnections in order to ease and speedup the implementation of the sparse weight matrices. In [31, 32, 33, 34], authors remove entire neurons, filters or channels altogether, in order to obtain a pruned structure that does not use hard-to-implement sparse parameter matrices. Additionally, [35] introduces a hybrid technique removing both filters in a structured way and single interconnections in a non-structured fashion.

In this dissertation we specifically employ non-structured pruning techniques, which allow a deeper compression of the network models compared to structured pruning with the drawback of needing more complex physical implementation due to the sparsification of the structures. There are different techniques that can be used to choose which parameters need to be removed when performing non-structured pruning. Here, we explore different pruning strategies.

1.2.1 One-shot pruning techniques

One-shot pruning methods are performed after training without any after-pruning fine tuning. Typically, when performing non-structured pruning, a score is assigned to each parameter in the DNN. When removing interconnections, only the parameters corresponding to the highest scores are kept, for example, when pruning a certain percentage $p\%$ of interconnections, $p\%$ of weights with the lowest scored are removed. A good selection of one-shot non-structured pruning methods is suggested by [22].

Global Magnitude Pruning (GMP) Starting from the assumption that the smaller in magnitude a parameter is, the less it potentially influences the output of the DNN, one of the simplest way of defining the scores for a parameter is by looking at its magnitude (i.e., its absolute value). With this method, scores are compared globally in order to remove a certain number of interconnections over the whole DNN. This can lead to unbalanced layers, as some layers may have sparser parameter matrices when compared to the others.

Layer-wise Magnitude Pruning (LMP) LMP follows the same phylosophy of GMP, but the scores are considered layer-wise instead of global-wise. This means that instead of removing $p\%$ of the total weights in the DNN, for each layer $p\%$ of weights is removed with no influence from the other layers.

Global Gradient Magnitude Pruning (GGMP) While GMP is quite an effective method with a very straight-forward implementation, it does not take into account the statistics of the data that is fed to the DNN when pruning. To alleviate this problem, it is possible to score each parameter by computing the absolute value of the product between the parameter value and its mean gradient, where the mean gradient is evaluated by propagating backwards the inference errors of a dedicated pruning dataset. Scores are evaluated globally.

Layer-wise Gradient Magnitude Pruning (LGMP) With the same reasoning of LMP, LGMP introduces the scoring system of GGMP, but layer-wise.

Random Pruning (RP) With this technique, parameters are scored randomly. This means that each weight is pruned independently with a probability equal to the fraction of the network that is being pruned. This method is typically used as a benchmark for the others, i.e., the validity of the other pruning methods is verified if in comparison the performance with RP are extremely poor.

1.2.2 Iterative pruning techniques

While one-shot pruning methods can be already quite effective by themselves, with a greater computational effort far better results can be achieved. In fact, it is still possible to train a pruned network to improve its performance, and this can be performed iteratively as many times as necessary. In short, the DNN can be trained, then pruned with any one-shot method, then trained again to be then pruned to a still smaller model, and so on. It is important to notice that while this kind of approach can be highly beneficial to the compression of the DNN, it is extremely high in computational complexity, given that the same network – which can be a very large network, of the size of billions of parameters – needs to be trained multiple times to reach good results.

Iterative pruning can be performed mainly in three different ways:

- the DNN is pruned, then trained again with the parameters as-they-are;
- the DNN is pruned, then trained again with randomly re-initialized parameters;
- the DNN is pruned, then trained again with parameters reinizialized to the same exact inizialization values as the first training.

The last option is the core of the *lottery ticket hypotesis* method, which is one of the most effective non-structured iterative pruning methods so far, as shown in practice [24].

Lottery ticket hypotesis The lottery ticket hypothesis [24] suggests that a DNN that is randomly initialized contains a sub-network (i.e., pruned network) that, if trained again with the same starting conditions (i.e., parameters initialization), can achieve the same accuracy of the original network after being trained for, at most, the same number of iterations.

In order to select the right sub-network the authors propose the following approach. Given a DNN with a total number of parameters P_0, training is performed for a certain number of epochs starting from a given set $\mathcal{W}_{\text{init}}$ of randomly initialized parameters. Then, ΔP_0 parameters are removed from the current number of parameters P_0 using GMP and the remaining weights are reset to the same initialized values $\mathcal{W}_{\text{init}}$ as before. Again, the DNN is trained and ΔP_1 parameters are removed from the remaining P_1 weights. This is performed iteratively, removing at each k-th iteration ΔP_k from the remaining P_k parameters, until the desired size of the network is reached. Typically ΔP_k is proportional to the remaining number of weights P_k, i.e, $\Delta P_k = \alpha P_k$, with $\alpha \in (0, 1)$ defining the portion of weights removed for each iteration.

1.3 Chapter conclusion

We have seen in this chapter different methods for DNN structure compression. The quantization of the parameters and data in the DNN can be optimized by performing quantization-aware training. At the same time, a special attention needs to be paid in order to keep all the values in the desired ranges. Additionally, the compression of DNN structure can be performed by means of pruning, i.e., the removal of interconnections (parameters) that minimally influence the performance of the structure. Pruning can be structured and non-structured, and non-structured pruning can be performed once after training (one-shot) or many times after multiple cycles of training (iterative). All these methods will be used in the following chapters of this dissertation.

Chapter 2

Avoiding the Multiplications: the Sum and Max (SAM) paradigm

While on one side there is the possibility of compressing standard DNNs with quantization and pruning techniques, as described in Chapter 1, it is possible to explore different neural network architectures in order to find hardware friendly structures with good training capabilities, with the aim of building energy efficient and low memory footprint architectures to be used in the field of tinyML.

In the search for alternative DNN structures, we work on substituting classical MAC operations with alternative map-reduce paradigms [36, 37].

Specifically, as a starting point, we want to avoid the use of the expensive multiply operations, thus using additions for the map operation. Looking for an alternative reduce operation, one of the best low-power alternatives to accumulation is maximum which has an equivalent cost in terms of operations. In this way we define the novel Sum-and-Max (SAM) map-reduce paradigm for DNNs [36].

In this chapter we define this novel paradigm, apply it to DNN layers to experiment on its properties and then try it on simple DNN tasks.

2.1 SAM-based DNN layers

Classic DNNs rely inference on a massive use of MAC operations that, despite being typically heavily optimized, make a large use of multiplications, greatly expensive both in terms of speed and power consumption. Here we propose a new DNN structure capable of completely avoid multiply operations.

For this reason, a new layer structure is proposed. In order to understand the new structure, we start from the MAC structure, the typical map-reduce operations used in most trainable DNN layers. This structure is common of Fully Connected (FC) layers, which can be represented as

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{2.1}$$

$$\boldsymbol{y} = f(\boldsymbol{z}) \tag{2.2}$$

where $\boldsymbol{x} \in \mathbb{R}^N$ is the input column vector of the layer, $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ is the matrix of the parameters, $\boldsymbol{b} \in \mathbb{R}^M$ is the bias column vector, $\boldsymbol{z} \in \mathbb{R}^M$ is the output column vector, $f(\cdot)$ is

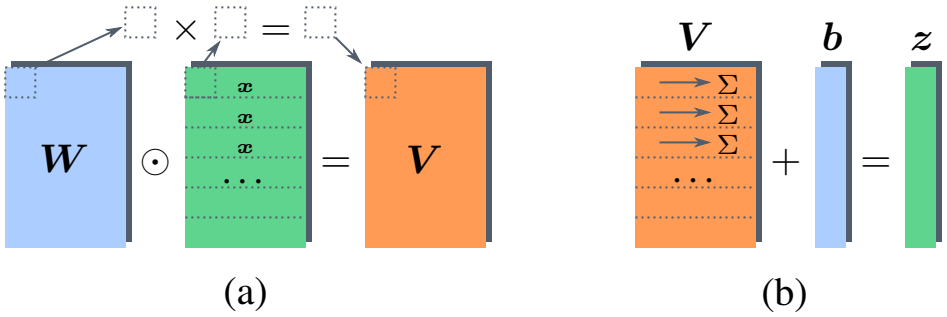


FIGURE 2.1: Split up of the multiply and accumulate operations for the MAC map-reduce paradigm. The classic MAC operation as a matrix-vector dot product $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$ is decomposed in the multiply operation (a), where matrix $\boldsymbol{W}$ is multiplied element-wise with a matrix whose rows are identical and equal to $\boldsymbol{x}$, and in the accumulate operation (b) where the elements of $\boldsymbol{V}$ are summed along the rows of the matrix.

the activation function and $\boldsymbol{y}$ is the activation output column vector. MAC operation in (2.1) can be rewritten more explicitly as

$$z_j = \sum_{i=1}^{M} w_{ji} x_i + b_j \quad \text{for } j = 1, \ldots, M \tag{2.3}$$

where z_j is the j-th scalar element of $\boldsymbol{y}$, x_i is the i-th scalar element of vector $\boldsymbol{x}$, b_j is the j-th scalar element of vector $\boldsymbol{b}$ and $w_{i,j}$ is the scalar element of matrix $\boldsymbol{W}$ at row i and column j.

As we want to search for a new map-reduce paradigm, we need to explicitly separate the multiply and the accumulate operations of (2.1) and (2.3). To do so, first we isolate the multiply operation by defining the *weighted inputs* matrix $\boldsymbol{V} \in \mathbb{R}^{M \times N}$ as

$$\boldsymbol{V} = \boldsymbol{W} * \boldsymbol{x}^{\top} \tag{2.4}$$

where $*$ is the Khatri-Rao product [38] that repeatedly performs element-wise products between each row of $\boldsymbol{W}$ and vector $\boldsymbol{x}^{\top}$. More explicitly

$$v_{ji} = w_{ji} x_i \quad \text{for } i \in 1, \ldots, N,\ j \in 1, \ldots, M \tag{2.5}$$

where v_{ij} and w_{ij} are the scalars at row i and column j of $\boldsymbol{V}$ and $\boldsymbol{W}$, respectively, and x_i is the j-th element of the input vector $\boldsymbol{x}$.

Then, the accumulate operation can be simply indicated as

$$z_j = \sum_{i=1}^{M} v_{ji} + b_j \quad \text{for } j \in 1, \ldots, M \tag{2.6}$$

where z_j is the j-th element of vector $\boldsymbol{z}$. In Figure 2.1 a summary of the multiply and accumulate split up is shown.

We want to avoid the use of multiplications, so we fully substitute the multiplications with additions and (2.5) becomes

$$v_{ji}^{\text{sum}} = w_{ji} + x_i \quad \text{for } i \in 1, \dots, N,\ j \in 1, \dots, M \tag{2.7}$$

Nonetheless, this makes the layer unusable as it is reduced to the accumulation of the inputs plus a constant value, i.e., the summation of all the weights and the bias for each neuron.

To solve this, we search for an alternative reduce operation that would work with sum as the map operation. Let us imagine that among the input values, which are weighted by parameters $\boldsymbol{W}$, only a few are prevalent for influencing the output, while the others have an almost negligible contribution. This concept can be roughly implemented by substituting the accumulate operation with the maximum operation, i.e., we search for the maximum scalar among all the weighted inputs. So, by replacing multiplications with additions and summation with maximum, (2.6) is rethought as

$$z_j = \max_{i \in \{1,\dots,N\}} v_{ji}^{\text{sum}} \quad \text{for } j = 1, \dots, M \tag{2.8}$$

where $\max_{i \in \{1,\dots,N\}} \cdot$ selects the maximum scalar value among the elements of its argument vector. We name this new map-reduce paradigm Sum and Max (SAM) and we build our new type of layer over this. No bias is needed as it is embedded in the values of $\boldsymbol{W}$.

The first thing that we can notice from this new structure is that, contrary to MAC structures and excluding uncommon degenerate cases, only the highest positive values v_{ji}^{sum} are promoted to the output, while anything else is completely ignored. This lead to a potential instability, with output values that are always high and positive with low-variance, eventually killing the information brought by data. A possible solution is the insertion of an inhibitory term in (2.8), such as

$$z_j = \max_{i \in \{1,\dots,N\}} v_{ji}^{\text{sum+}} - \max_{i \in \{1,\dots,N\}} v_{ji}^{\text{sum-}} \quad \text{for } j = 1, \dots, M \tag{2.9}$$

where $v_{ji}^{\text{sum+}} = x_i + w_{ji}^{+}$ and $v_{ji}^{\text{sum-}} = x_i + w_{ji}^{-}$ are obtained from two different weights matrices, namely the potentiation matrix $\boldsymbol{W}^{+}$ and inhibition matrix $\boldsymbol{W}^{-}$. Now, even if (2.9) can effectively work as a valid DNN layer, we observe that both its terms are influenced by the highest positive values x_i, while negatives values are typically ignored. Then, a further modification of (2.9) results in

$$z_j = \max_{i \in \{1,\dots,N\}} v_{ji}^{\text{sum+}} + \min_{i \in \{1,\dots,N\}} v_{ji}^{\text{sum-}} \quad \text{for } j = 1, \dots, M \tag{2.10}$$

where $\min_{i \in \{1,\dots,N\}} \cdot$ selects the minimum scalar value among the elements of its argument vector. Now, the second term of (2.10) promotes negative and high (in module) values of $v_{ji}^{\text{sum-}}$. Because of this, this term works actually as an inhibitory term.

Remarkably, the activation function is not required in SAM-based layers as the maximum and minimum reduce functions already introduce a non-linearity.

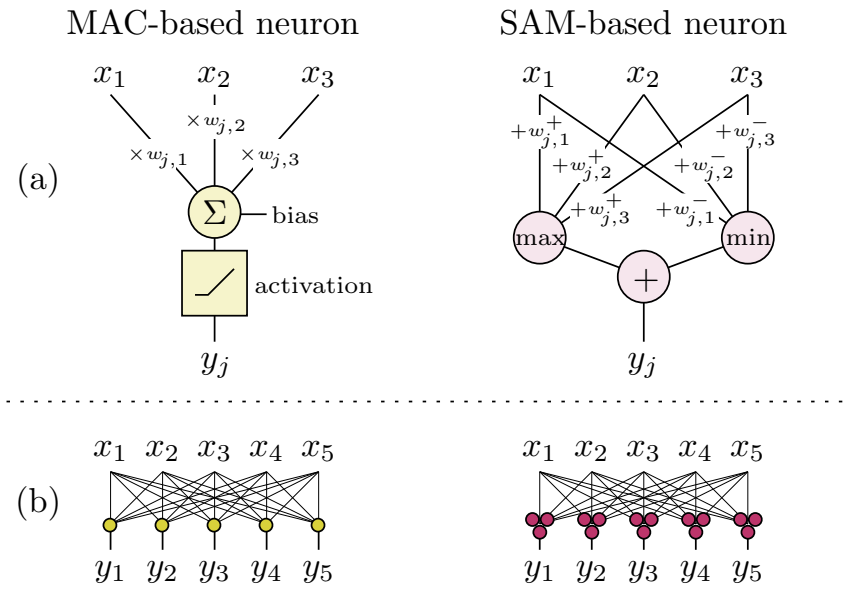


FIGURE 2.2: Comparison of a MAC-based neuron and a SAM-based neuron (a) and their disposition inside a dense layer (b). Note that in the SAM-based neuron parameters are added and not multiplied, while bias and activation function are not needed ($y_j = z_j$), as opposed to MAC-based neurons. (© 2022 IEEE)

In Figure 2.2 we highlight the structures of the DNN neurons based on MAC and on SAM structures, which correspond to the operations for the generation of a single scalar value y_j. Also, it is shown how the neurons are disposed inside a dense layer to generate the whole output $\boldsymbol{y}$, i.e., the graphical representation of (2.3) and (2.10).

The resulting layer uses additions instead of multiplications for the application of the parameters to the inputs. Also, it needs only subtractions to perform the comparisons between values needed by $\max_{i\in\{1,\dots,N\}} \cdot$ and $\min_{i\in\{1,\dots,N\}} \cdot$ function.

As we want to train this structure with supervised Stochastic Gradient Descent (SGD) based training, a gradient is defined for $\max_{i\in\{1,\dots,N\}} \cdot$ and $\min_{i\in\{1,\dots,N\}} \cdot$ functions as follows: it is set to 1 (linear) for input x_i that is selected and used by the operation to produce the output, 0 for all the others.

2.2 Performance and cost

We want to assess the performance of a layer built with (2.10) by using a simple working toy-case. To do so, we prepare a simple classification task with 10 classes and 2 different DNN structures, one based on classic MAC operations, the other one based on the new SAM paradigm. The two DNNs are built and trained so that they have the same final performance in terms of accuracy. After training, pruning – that is the removal of the least influent parameters from the layer – is used to compress the two structures and finally the cost in terms of memory footprint and computational operations is assessed.

2.2.1 Classification task

In order to study the performance of SAM-based layers in DNNs, a simple classification problem is prepared as a case study. Starting from a covariance matrix $\boldsymbol{X}$ of size $N \times N$, a signal $\boldsymbol{x}$ with N samples can be generated with a zero-mean multivariate gaussian random process. Matrix $\boldsymbol{X}$ is generated starting from a value $r \in (-1, 1)$ as

$$\chi_{i,j} = r^{|i-j|} \tag{2.11}$$

where $\chi_{i,j}$ is the element corresponding to row i and column j of matrix $\boldsymbol{X}$. Different values of r corresponds to a different distribution of the resulting signal $\boldsymbol{x}$, such that if $r < 0$ the frequency band of the resulting signal is high-pass, while if $r > 0$ it is low-pass, and the higher is $|r|$, the more localized is the signal, meaning the frequency band is less wide [39]. This means that $r = 0$ corresponds to flat band white noise, while values near -1 and 1 correspond to high frequency and low frequency and thin band signals, respectively.

Given a signal $\boldsymbol{x}$ generated from an unknown covariance matrix $\boldsymbol{X}$, the classification task consists in retrieving the correct value r associated with it. In particular, we generate the data set starting from 10 different values of r equally distributed in the interval $[-0.9, 0.9]$, so the DNN must choose the correct value of r among 10 different possibilities.

2.2.2 DNN configuration

Our test structure consists in a Multi Layer Perceptron (MLP) structure with $n = 512$ inputs, one FC hidden layer and one FC dense layer with 10 outputs and a softmax activation function. The hidden layer can be either a MAC-based FC layer (we call this configuration MLP-HMAC) or a SAM-based layer (we call this configuration MLP-HSAM). The number of outputs of the hidden layer is selected in order to have a similar classification performance for the two configurations, so that it is possible to make a fair comparison between the two structures. To do so, different DNN with different sizes for the hidden layer are trained and compared and a good compromise between hidden layer size and performance of the two configurations is found. With this, the hidden layer of MLP-HMAC is set with $0.6n \simeq 307$ outputs (that is, 157 286 weights plus 307 bias values) and the hidden layer of MLP-HSAM is set with $2n = 1024$ outputs (that is, 1 048 576 parameters). A summary of the structure of the two configurations can be seen in Figure 2.3. Even tough MLP-HSAM initially has a far greater number of parameters, their number can be largely reduced at post-training optimization time.

2.2.3 Training and performance

During training, pairs of signals $\boldsymbol{x}$ and labels associated to one of the 10 values of r are fed to the DNN. Training is performed for both configurations A and B with Adam optimizer [40] through 1000 epochs. As described in Section 2.2.1, the dataset is generated randomly on-the-fly using 10 different covariance matrices generated from as many values r, so that validation data is not required during training. MLP-HMAC uses a batch size of 64 with a learning rate of 0.001 and performs 1000 steps per epoch. MLP-HSAM uses a batch size of

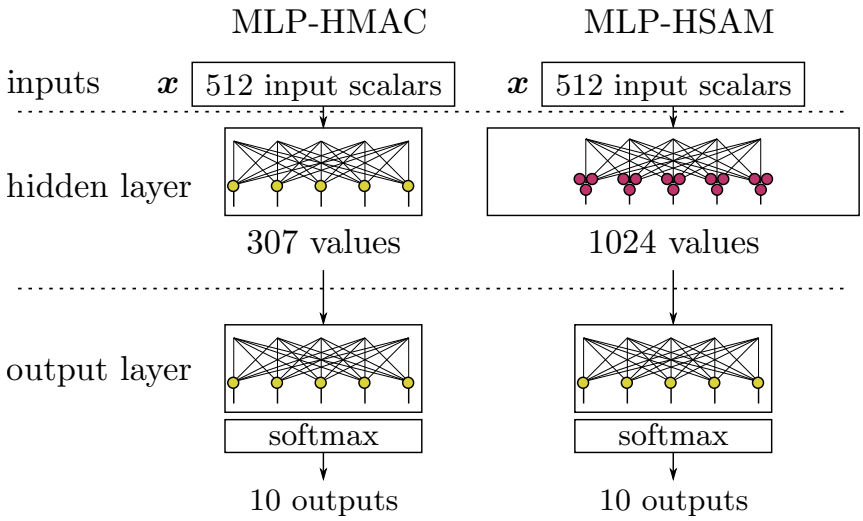


FIGURE 2.3: Structures of MLP-HMAC, that has a MAC-based hidden layer with 307 outputs, and MLP-HSAM, that has a SAM-based hidden layer with 1024 outputs. (© 2022 IEEE)

256 with a learning rate of 0.008 and performs 250 steps per epoch. So, both configurations feed $64\,000$ randomly generated signal-label pairs to the DNN each epoch, with a total of $64 \cdot 10^6$ different instances fed thorough the whole training procedure.

The final accuracy – defined as the number of correct classifications over the total number of inferred input vectors – is the same for both configurations A and B, as they have been structured to have the same performance, and about equal to 0.95. This result is enough to show that in this case (2.10) may be effectively used to substitute a dense MAC-based layer. Nonetheless, MLP-HSAM needs more epochs to converge to the final result, even if the training hyper-parameters (such as number of epochs and learning rate) have been selected to maximize the training speed. This is due to the way the gradients of $\max_{i\in\{1,\dots,N\}} \cdot$ and $\min_{i\in\{1,\dots,N\}} \cdot$ are built, which allows the gradient of each output to propagate to only one input for each signal fed to the SAM structure. Thus, for each signal only one parameter per output is actually updated, slowing the whole process. Training accuracy trend – that coincide with validation accuracy in this case, as signals are synthetically generated – can be seen in Figure 2.4.

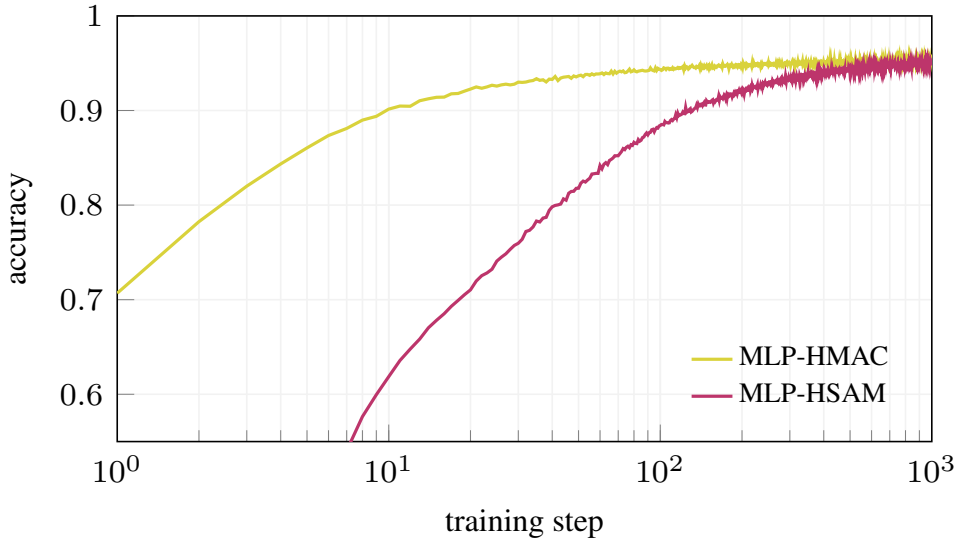


FIGURE 2.4: Accuracy trend during training of MLP-HMAC (containing MAC-based hidden layer) and MLP-HSAM (containing SAM-based hidden layer). The dataset is generated randomly on-the-fly, so no validation curve is required. (© 2022 IEEE)

2.2.4 Optimization by pruning and cost

After training, the 2 configurations are tested against a fixed dataset of 10^5 random generated signals. In particular, the confusion matrix of MLP-HSAM is shown in Figure 2.5, highlighting what are the values of r that the DNN predicts against the actual target values.

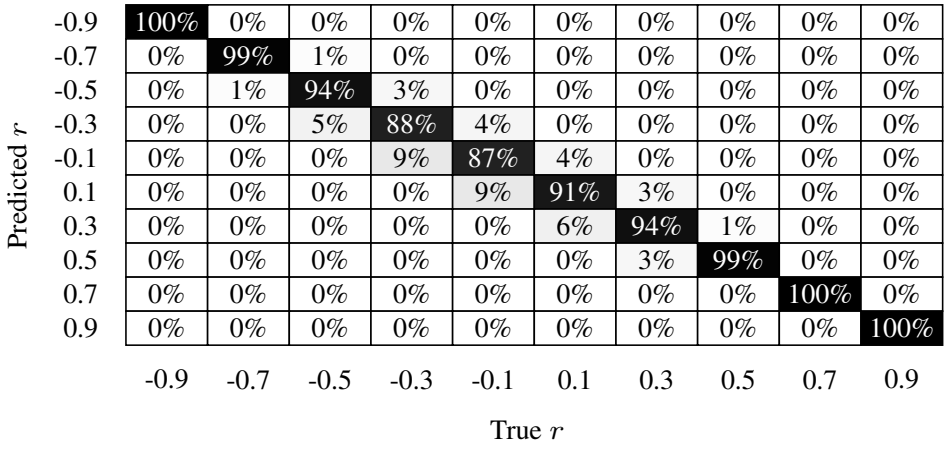


FIGURE 2.5: Confusion matrix of MLP-HSAM (with SAM-based hidden layer) with a test set of 10^5 random signals. As the data set is equally distributed through the 10 classes, values are normalized over 1000 instances for each class. (© 2022 IEEE)

We also apply pruning to the hidden layer [25], i.e., the removal of interconnections to the DNN layer. Removing an interconnection fundamentally means completely ignoring a parameter during the inference of the DNN. While for MAC-based layers this is mathematically obtained by setting the parameter to 0, for SAM-based layers it is necessary to set the parameter

to $-\infty$ or ∞ for $\max_{i\in\{1,\dots,N\}} \cdot$ and $\min_{i\in\{1,\dots,N\}} \cdot$, respectively. Ignoring a parameter means reducing the memory footprint of the DNN and the number of operations that it is necessary to perform.

For the sake of simplicity, the naive approach of LMP as described in Section 1.2.1 is used for MLP-HMAC: all the parameters whose absolute value is below a given threshold are set to zero. Many thresholds are tested and the resulting accuracy is tested for each of them. Of course, the greater the threshold, the lower the accuracy, as less parameters are employed.

MLP-HSAM does not follow the same procedure. During SAM operations, maximum (or minimum) operation selects only one value $v_{ji}^{\text{sum+}}$ (or $v_{ji}^{\text{sum-}}$) for each $j = 1, 2, .., M$. The parameter w_{ji}^{+} (or w_{ji}^{-}) associated with the selected weighted input is the only value for a given whole input vector $\boldsymbol{x}$ that actually influences output z_j. So, we count how many times each parameter is selected and we use this quantity to define a probability of selection value. A probability of selection threshold is set and all the connections that are below this value are removed, i.e., the parameters that are less selected are ignored. Different probability of selection thresholds are set and for each of them the classification performance is evaluated.

Actually, the analyses show that for most of the parameters in MLP-HSAM the probability of selection is completely null, so this structure is naturally prone to aggressive pruning. Because of this, even if during training MLP-HSAM requires more parameters compared to MLP-HMAC, during inference it actually uses a far lower number.

In Figure 2.6 we see the classification performance of MLP-HMAC and MLP-HSAM versus the number of parameters that are kept in the hidden layer. Without pruning, both configurations have an accuracy of 0.95. The MAC-based hidden layer uses about $157\,000$ parameters, while the SAM-based layer employs only about $10\,000$, after removing the useless (null activation) parameters. If, instead, we apply pruning and we lose 0.05 in accuracy performance, with MLP-HMAC we have about $80\,000$ parameters, while with MLP-HSAM we have only about 5500 parameters.

TABLE 2.1: Memory footprint and computational cost of the hidden layer of configurations A and B

Configuration	Parameters	Multiplications	Additions
A (MAC-based)	$80.0 \cdot 10^3$	$80 \cdot 10^3$	$80 \cdot 10^3$
B (SAM-based)	$5.5 \cdot 10^3$	0	$11 \cdot 10^3$

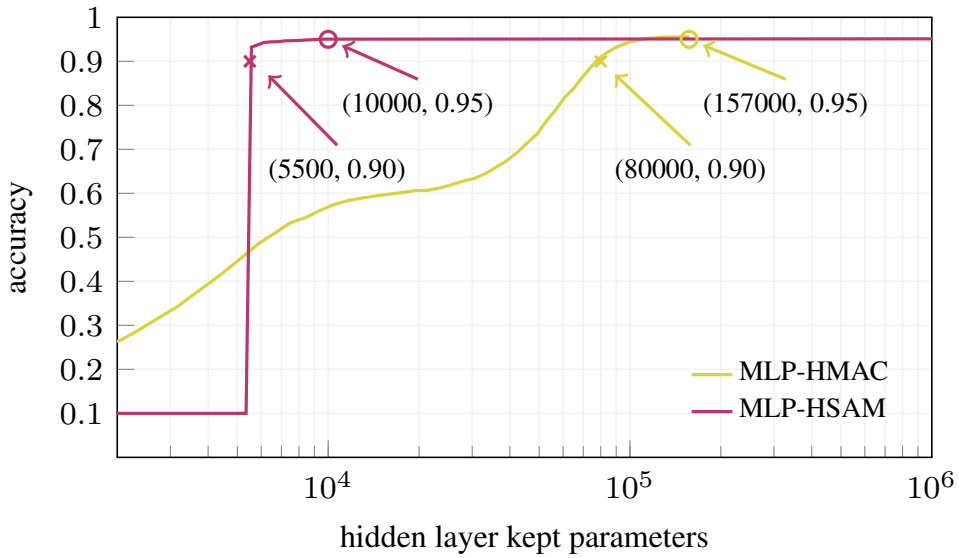


FIGURE 2.6: Accuracy vs number of parameters that are kept after pruning in the hidden layer, both for MLP-HMAC (containing the MAC-based hidden layer) and MLP-HSAM (containing SAM-based hidden layer). The numbers of parameters with accuracy 0.95 and accuracy 0.90 for the two configurations are highlighted. (© 2022 IEEE)

Even if we are focusing on the hidden layer, it is important to notice that the number of parameters in the output layer for MLP-HMAC is lower than MLP-HSAM, having 3080 parameters against 10 250, respectively. Nonetheless, MLP-HSAM remains a far cheaper implementation.

Finally, we evaluate the cost in terms of number of additions and multiplications of the hidden layer of the two configurations. We define N_A as the number of parameters kept in the hidden layer of MLP-HMAC, N_B as the same for MLP-HSAM. In MAC-based MLP-HMAC layer, we perform a number of multiplications $\mathrm{mul}_\mathrm{A} = N_\mathrm{A}$ and a number of additions $\mathrm{add}_\mathrm{A} = N_\mathrm{A}$ for each inference step. In SAM-based MLP-HSAM hidden layer instead we need $\mathrm{mul}_\mathrm{B} = 0$ and $\mathrm{add}_B = 2N_\mathrm{B}$, as $\max_{i\in\{1,\dots,N\}} \cdot$ and $\min_{i\in\{1,\dots,N\}} \cdot$ operations are performed through successive comparisons (i.e., subtractions with sign check of the result). The memory footprint and the cost in terms of multiplications and additions is shown in Tab. 2.1, where the big advantage in terms of hardware resources of the SAM-based layer over the MAC-based layer is highlighted.

2.3 Chapter conclusion

We have presented a multiplier-free DNN layer based on a SAM paradigm, opposed to the classic MAC structure. The performance of this layer has been tested on a simple classification task, and its functioning has been proven, even though a further research on the training process could improve the results. Most interestingly, this new layer has been shown to be prone to aggressive pruning, thus actually needing only a few parameters and because of this being very cheap in terms of memory footprint and number of operations. This is a fundamental property of the structures characterized by the maximum and minimum reduce operations we will use in the next chapters to build a more robust, highly prunable, DNN structure.

Chapter 3

A Gardener's Delight: the Multiply and Max&Min (MAM²) paradigm

With SAM-based layers we have seen that it is possible to substitute classic MAC map-reduce paradigm with different structures that still retain learning capabilities, while introducing new properties that can be leveraged on in order to minimize the memory footprint and computational cost of the network. In particular, SAM paradigm is found to be fit for extreme pruning of the structure interconnections. Nonetheless, SAM is found to be slow in convergence and difficult to train with more than one consecutive hidden layer, because the gradient travels only on the interconnections that are selected by the operations in (2.10) and the weights update is never proportional to the inputs or to the weights in the successive layers, because of the sum map operation.

Because of this, as we want to leverage on the natural sparsity properties of maximum and minimum reduce operations, we propose Multiply and Max&Min (MAM^2) [37], another map-reduce alternative which drops the advantage of using only adders in the structures but guarantees performance comparable to MAC-based layers for many different state-of-the-art DNN structures.

3.1 The MAM²-based layer architecture

In this section we describe a novel layer architecture based on Multiply, Max and Min operations which is naturally prone to aggressive pruning. After a formal definition of the novel DNN layer, we explain the *vanishing contributes* techniques that is used for training.

3.1.1 Layer description

We refer to (2.1) as the standard definition of a feed-forward FC MAC-based layer, where $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ is the matrix of the weights. Additionally, we recall (2.5) and (2.4) which define the multiply map operation and the weighted inputs matrix $\boldsymbol{V}$ and we recall (2.6) which defines the accumulate reduce operation.

Looking at (2.6), when thinking of pruning the DNN layer, we want only a few entries of matrix $\boldsymbol{V}$ to actually influence the prediction, so that we can remove the less significative contributes of $\boldsymbol{V}$, i.e., the unused interconnections. We take to extremes this concept, while also keeping into account a structure that would not require far more complex hardware when

compared to the accumulation, as we have done for the SAM paradigm. We force each neuron of the layer to take into account only two of the most significant contributes, that is to say, the maximum and the minimum values of each row of $\boldsymbol{V}$. This leads to a revision of (2.6), that becomes

$$z_j = \underset{i=1}{\overset{N}{\mathcal{M}}} v_{ji} + b_j \quad \text{with } j \in \{1, .., M\} \tag{3.1}$$

where we define operator $\mathcal{M}$ as

$$\underset{i=1}{\overset{N}{\mathcal{M}}} v_{ji} \triangleq \max_{i \in \{1,..,N\}} v_{ji} + \min_{i \in \{1,..,N\}} v_{ji} \tag{3.2}$$

Compared to SAM, the map operation is the same as for MAC (i.e., multiplication) and a single parameters matrix $\boldsymbol{W}$ is used (contrary to the two potentiation and inhibition matrices of SAM).

We will refer to the map-reduce operations in (2.5) and (3.1) as the the Multiply and Max&Min paradigm (MAM^2, *MAM-squared*). These equations, together with (2.2), describe the novel MAM^2-based layer. In Figure 3.1 a visual representation of (3.1) is shown.

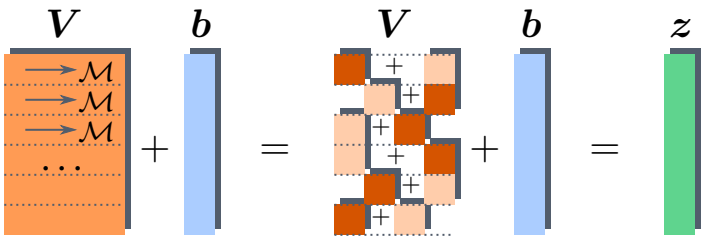


FIGURE 3.1: Max&Min reduce operation. For each row of $\boldsymbol{V}$, the minimum and maximum values are selected, then added together.

3.1.2 A bridge from MAC to MAM²: the vanishing contributes

A potential problem of MAM^2 layers is the propagation of the gradients, as, given a single input instance, for each row of matrix $\boldsymbol{V}$ only a couple of gradients are actually propagated backwards. Because of this, to ease the train process of MAM^2-based layers, we define a layer structure that is the bridge between the MAC and the MAM^2 layers. This can easily be performed by introducing a parameter β, so that (2.6) and (3.1) are merged in

$$z_j = \beta \sum_{j=1}^{N} v_{ij} + (1 - \beta) \underset{j=1}{\overset{N}{\mathcal{M}}} v_{ij} + b_j \quad \text{with } i \in \{1, .., N\} \tag{3.3}$$

where $\beta \in [0, 1]$. With $\beta = 1$ and $\beta = 0$ only the MAC or the MAM^2 contributes remain, respectively, while an intermediate value of β guarantees a transition between the two paradigms.

During training, By using (3.3) it is possible to gradually transition from a MAC-based to a MAM^2-based layer. This enables, in a first phase of the training, the backward propagation of the gradient through all the interconnections. In particular, this is achieved in the following

way. Starting with $\beta = 1$ during the first training iteration, for each training iteration β is lowered linearly until it reaches $\beta = 0$. When $\beta = 0$ (3.3) is reduced to (3.1) and β is kept fixated for the rest of the training iterations, keeping only the MAM^2 contribute. We call this method *vanishing contributes*. This technique allows a better performance of the DNN when compared to the work on SAM, where the paradigm based on max and min is trained as-it-is.

It is important to highlight that (3.3) is used only during the training phase, while pruning of the layers and test inferences are performed strictly with (3.1) (i.e., $\beta = 0$).

3.2 MAM² training method and pruning: a toy case

As a first example on how the MAM^2-based layers can be used to obtain an aggressively pruned network, we employ a simple DNN composed of 3 FC layers as shown in Figure 3.2 paired with MNIST dataset [41]. We define this model as FC-DNN, with FC-DNN-MAC the version with MAC-based hidden layers and FC-DNN-MAM^2 the version with MAM^2-based hidden layers, and we compare them.

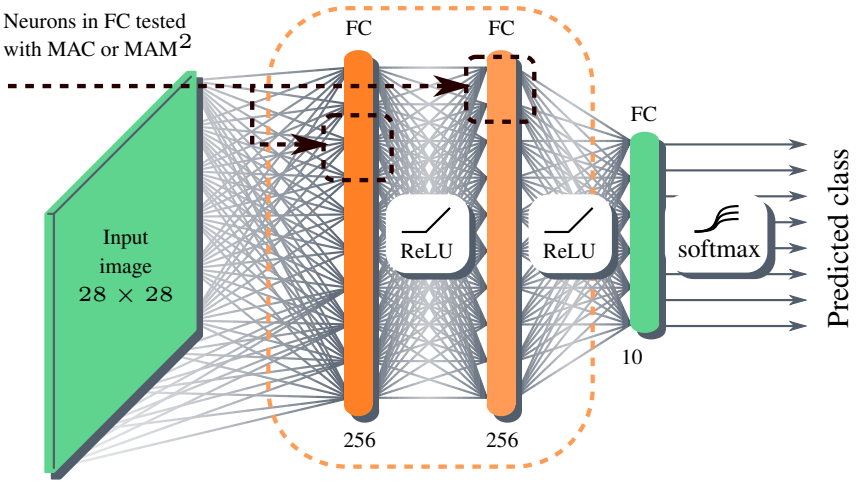


FIGURE 3.2: Structure of the 3 FC layers DNN for MNIST classification (FC-DNN). The two hidden FC layers are implemented either using MAC or MAM^2 map-reduce paradigm.

MNIST dataset task consists in the classification of handwritten digits among 10 different classes. The dataset is composed of a total of 70000 b/w images of size 28×28. The dataset is split in 50 000 training images, 10 000 validation images and 10 000 test images. Each pixel in the image is a value normalized between 0 and 1. In order to improve the generalization capabilities of the trained network, data augmentation is performed. This in particular consists in image rotation, shifting, scaling, horizontal and vertical shrinking and shearing [41].

In order to prune the two structures for the comparison, all the techniques discussed in Section 3.4.4 are employed for both MAC and MAM^2 FC hidden layers. Also, PSP technique is applied to the MAM^2 model and the selection probability of each value in each of the two hidden FC layers is analyzed.

3.2.1 Training method

During training, pairs of images and labels from MNIST are fed to FC-DNN and the parameters are updated using Adam optimizer [40]. Training MAM^2 layers requires starting with standard MAC-based layers and then quickly transition to MAM^2 following (3.3), defined by the vanishing contributes technique of Section 3.1.2. More in detail, we define for each training epoch q a value $\beta[q]$ to be substituted to β in (3.3). We start with $\beta[1] = 1$ at epoch 1 and we decrease it linearly to $\beta[Q] = 0$ at the Q-th epoch. After this, $\beta[q]$ is kept to 0 for $q > Q$. The accuracy trend during training with different values of Q can be seen in Fig. 3.3. As expected, when $\beta[q] > 0$ in the first Q epochs all the contributes to the summation in (3.3) have not completely vanished yet. Nonetheless, when $\beta[q] = 0$ all the contributes but the maximum and the minimum vanish, producing an initial decrease in accuracy that is quickly retrieved during the successive epochs. As a rule of thumb, Q should be chosen small but greater than a few epochs. In fact when Q is too small, the training process fails to train the DNN to its maximum possible accuracy, while on the other hand, high values of Q make the convergence slow.

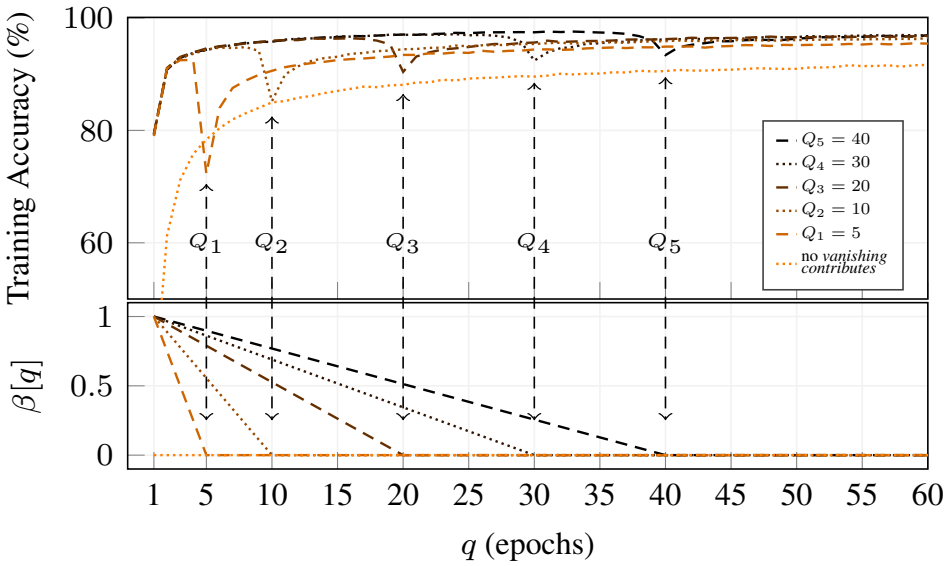


FIGURE 3.3: Top-1 accuracy trend on the training set during the training of the FC-DNN with MNIST and $\beta[q]$ trend. Different values of Q are tested. After an initial accuracy increase, there is an accuracy drop when $\beta[Q] = 0$ but the DNN quickly recovers its classification capability. With Q big enough, the sensibility of the final accuracy with respect to Q is not significant.

In this section, training with the FC-DNN + MNIST is performed for 80 epochs with learning rate set to 0.001, batch size set to 128 and $Q = 30$.

3.2.2 The natural sparsity of MAM² layers

We apply a particular pruning technique in order to analyze the properties of MAM^2, namely Probability of Selection Pruning (PSP) to the trained DNN. This methods works as follows. When performing a single inference through the DNN, only two values for each row of $\boldsymbol{V}$ are

actually propagated and influence the output, as seen in (3.1) and visible in Figure 3.1. In this way, when inferring a single input, we know that most of the values in $\boldsymbol{V}$ (i.e., weights in $\boldsymbol{W}$) are definitely not used. By leveraging on this property, it is possible to define a new score to be assigned to each parameter. By using a dedicated pruning dataset, for each inferred input we keep track of the parameters that are chosen. Over the whole dataset, it is possible to evaluate for each parameter the approximated probability of being selected by the MAM^2 operation, thus defining a set of scores useful to select what interconnections are to be removed. Similarly to the other methods, the scores can be evaluated globally or layer-wise, with the former as the one employed in this work.

The validation set, employed as pruning set, is used as input and the probability of selection for each element of matrix $\boldsymbol{V}$ is evaluated for each layer. In Figure 3.4 the values of $\boldsymbol{V}$ with non-zero probability for the two MAM^2-based layers are highlighted, showing that most values have null probability of being selected. This suggests how MAM^2 results indeed in an intrinsically sparse – and because of this, prunable – structure. Furthermore, in Figure 3.5, the trend of the percentage of interconnections with a varying minimum probability of selection is represented, showing that only a few interconnections have a high probability of being used while most of them are used only a few times.

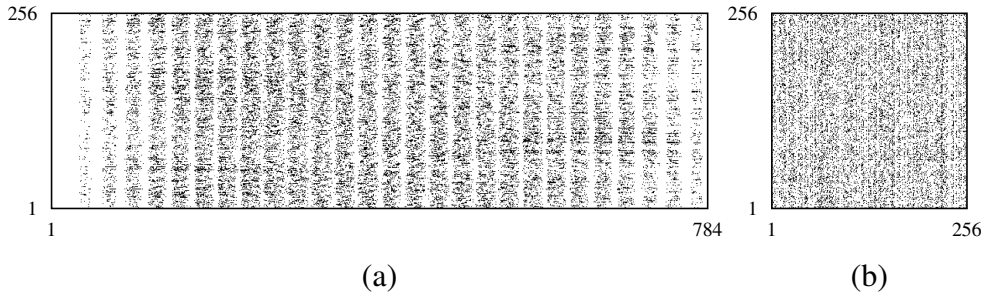


FIGURE 3.4: Maps of the probability of selection for matrices $\boldsymbol{V}$ of the first (a) and second (b) hidden FC layers. Black dots correspond to values of $\boldsymbol{V}$ used at least once (i.e., with non-zero probability), corresponding to 15.7% of the total interconnections for the first layer and 19.0% for the second.

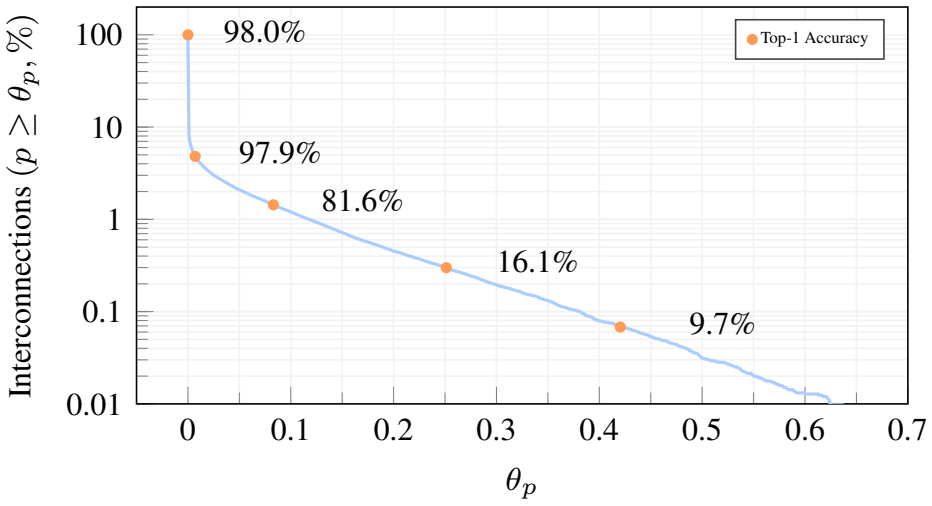


FIGURE 3.5: Percentage of interconnections with a probability of selection p greater than a threshold value θ_p, with a variable value of θ_p. Most of the interconnections (i.e., parameters) have values of p very low if not null. Also, the accuracy on the task while keeping only the interconnections with $p \geq \theta_p$ is shown, demonstrating that the parameters corresponding to the lowest values of p can be safely removed.

These sparsification properties can be measured in practice when pruning the DNN with any of the methods presented in Section 1.2.1. In Figure 3.6, the performance of the pruned FC-DNN + MNIST with either MAC or MAM^2 based FC hidden layers is shown. As expected, the MAM^2 based layers better retain the baseline accuracy when most of the interconnections are removed from the structure. Moreover, values of $\boldsymbol{V}$ which are often selected typically also correspond to high magnitude weights, so pruning methods such as GMP are effective when applied to these layers.

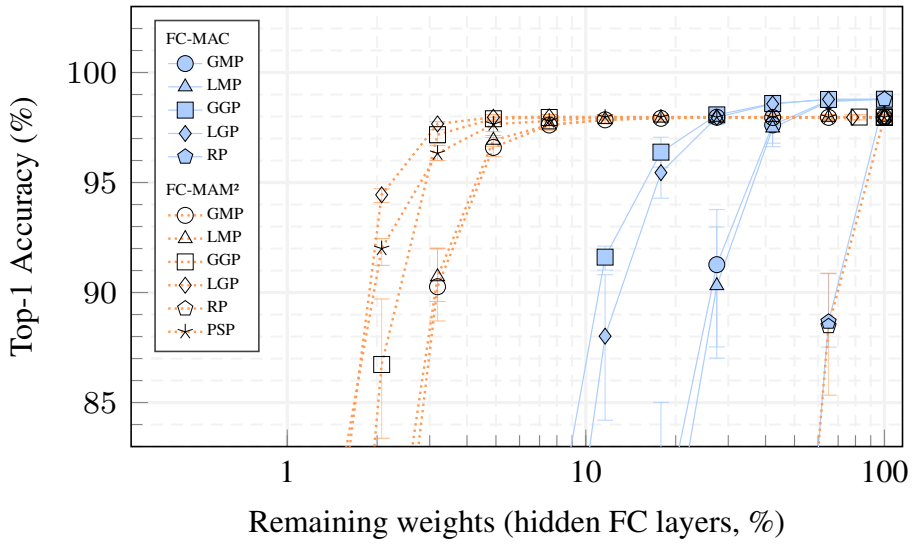


FIGURE 3.6: FC-DNN + MNIST + one-shot pruning: Top-1 accuracy vs percentage of kept weights in the two hidden FC layers. The points show the mean performance over 5 tests, with the maximum and minimum accuracy indicated by the error bars.

3.3 Performance and cost

To analyze the behavior of MAM^2-based layers, we test them with classical computer vision DNNs, i.e., with structures capable of solving image classification tasks. In this section we present the model architectures and the datasets employed for the tests. Training processes and tests are performed within a TensorFlow framework, with MAM^2 operation implemented in C++ as a custom TensorFlow operation.

3.3.1 Model architectures

The two architectures we will consider are classical DNNs used to solve classification tasks, namely AlexNet and VGG-16 models. The former was designed by A. Krizhevsky in collaboration with I. Sutskever and G. Hinton and competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012 [42] while the latter was proposed by K. Simonyan and A. Zisserman in [43] and was submitted to ILSVRC-2014. The versions of these two networks that we use in this work are summarized in Figure 3.7 and in Figure 3.8. In this work, for each of the two architectures we focus only on the two FC hidden layers – highlighted in Figure 3.7 and in Figure 3.8 – since they contain 90.0% and 86.4% of the total amount of weights of AlexNet and VGG-16, respectively. In order to assess the pruning capabilities of the MAM^2 paradigm, we replace them with MAM^2-based FC layers. From now on, we will refer to the two original DNN models as AlexNet-FCMAC and VGG-16-FCMAC and to their modified versions as AlexNet-FCMAM^2 and VGG-16-FCMAM^2.

3.3.2 Datasets

We test AlexNet and VGG-16 models on three different datasets, namely CIFAR-10, CIFAR-100 [44] and ImageNet [45].

CIFAR-10 dataset task consists in the classification of animals and vehicles among 10 different classes. The dataset consists of a total of 60 000 colour images of size 32×32. It is split in 45 000 training images, 5000 validation images and 10 000 test images. Data augmentation is performed and it consists in random image rotation, shifting and change of saturation, contrast and brightness. Each pixel in the images is normalized between 0 and 1 and all samples are resized to be of size 224×224 using bilinear interpolation.

CIFAR-100 dataset task consists in the classification of items belonging to 100 different classes. Similarly to CIFAR-10, it consists of a total of 60 000 colour images of size 32×32. It is split in 45 000 training images, 5000 validation images and 10 000 test images. Each pixel in the image is normalized between 0 and 1 and in and all samples are resized using a bilinear interpolation to be of dimension 224×224. The same data augmentation employed for CIFAR-10 is used.

ImageNet is a large database made available for the first time in 2010 for the ImageNet Large Scale Visual Recognition Challenge. It contains more than 1.2 million RGB images of different sizes representing objects belonging to 1000 different classes. The training set is composed by 1 281 167 images, while the validation and the test set are 50 000 images each. Data augmentation on the training set is performed by means of random horizontal flip,

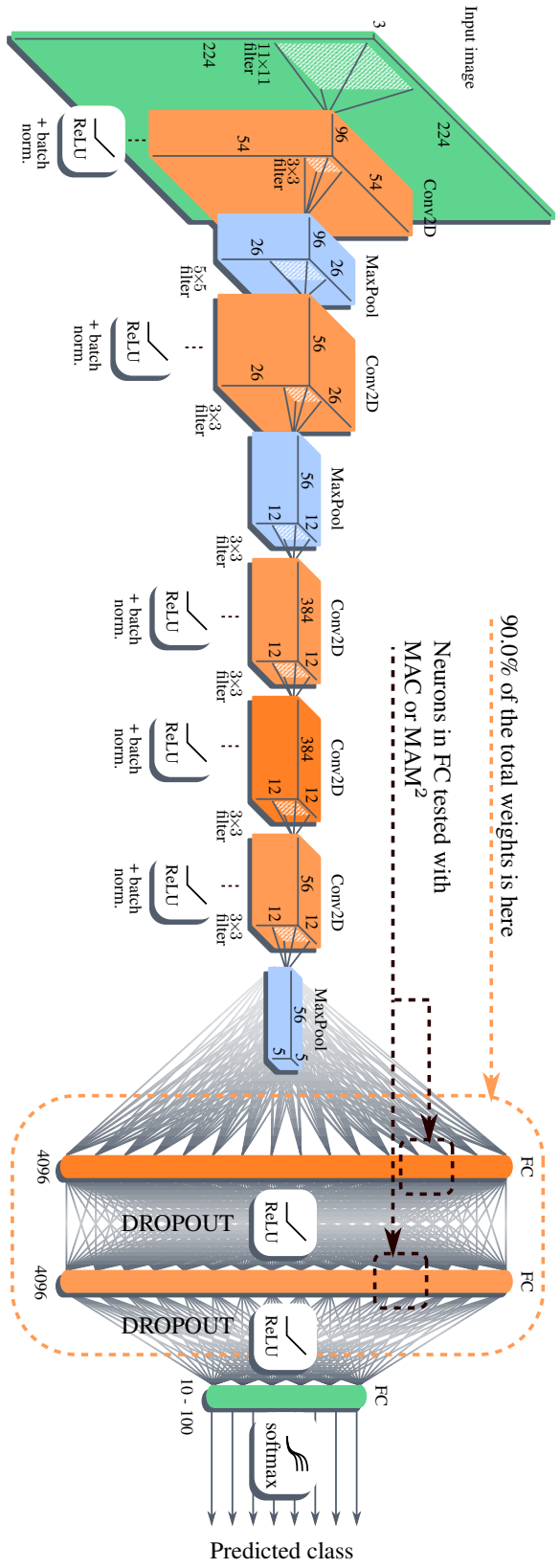


FIGURE 3.7: AlexNet architecture. The two highlighted FC layers employ 91% of the total number of parameters in the whole structures. These two layers are being tested both with classic MAC based neurons and the novel MAM² map-reduce paradigm for a comparison in pruning capabilities and performance.

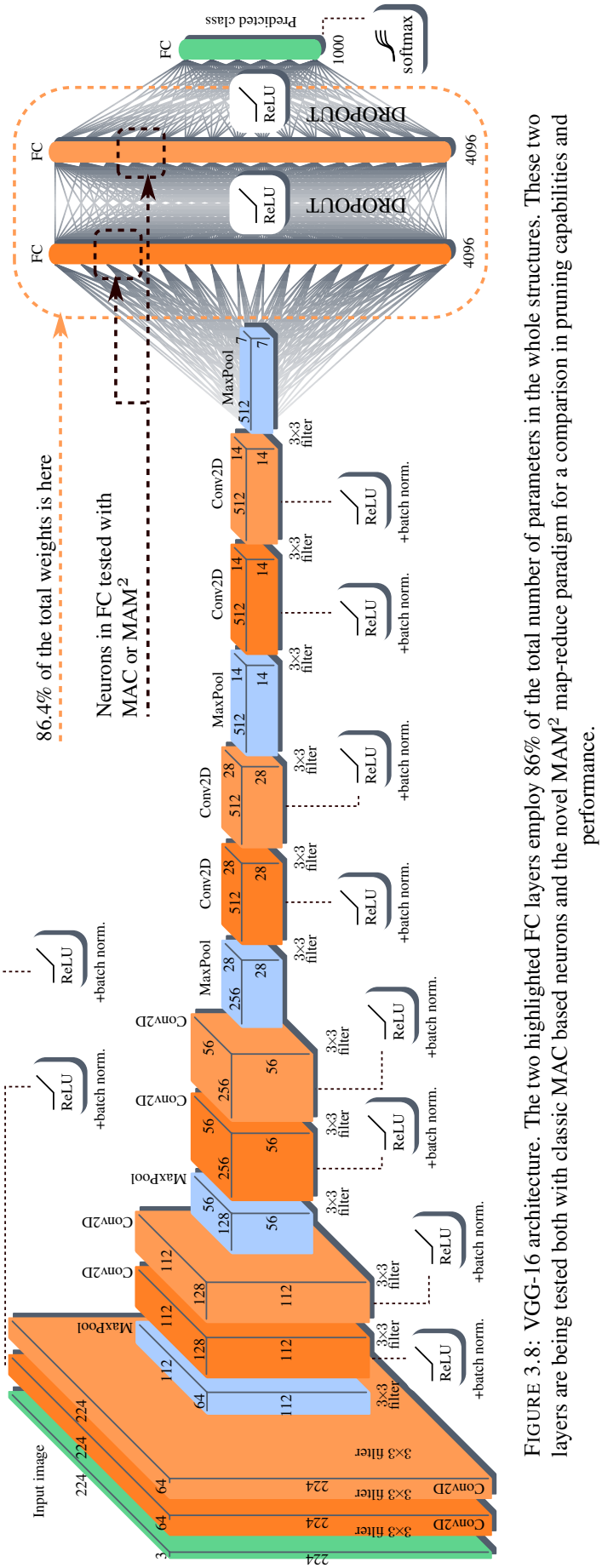


FIGURE 3.8: VGG-16 architecture. The two highlighted FC layers employ 86% of the total number of parameters in the whole structures. These two layers are being tested both with classic MAC based neurons and the novel MAM^2 map-reduce paradigm for a comparison in pruning capabilities and performance.

random saturation and random change in contrast and brightness. All the images are resized to 224×224 pixels using a bilinear interpolation.

3.3.3 MAM² performance: one-shot pruning

In this section we expand the discussion on MAM^2-based DNN layers and we show a comparison between MAC-based and MAM^2-based layers on three different datasets, namely CIFAR-10, CIFAR-100 and ImageNet, as they are described in Section 3.3. AlexNet-FCMAC, AlexNet-FCMAM^2, VGG-16-FCMAC and VGG-16-FCMAM^2 models are pruned using one-shot (i.e., post training pruning with no further fine-tuning) non-structured methods as described in Section 3.4.4 and the improvements introduced by MAM^2 in terms of parameters reduction and computational load reduction are assessed.

Training with AlexNet + CIFAR-10 and AlexNet + CIFAR-100 is performed for 230 epochs with learning rate to 0.001 and batch size to 256. Dropout after the two hidden FC layers is set to 0.5 for AlexNet-FCMAC as indicated in [42] and to 0.2 for AlexNet-FCMAM^2, with $Q = 12$ for the vanishing contributes method as explained in Section 3.1.2 and in Section 3.2.1.

Converesly, VGG-16 + ImageNet is trained starting from a pre-trained version of classic VGG-16-FCMAC model[1]. Training is performed using Stochastic Gradient Descent (SGD) for 45 epochs. All the parameters of the DNN are kept fixed during training for 35 epochs, with the exception of the two hidden FC layers and the output FC layer, whose parameters are re-initialized randomly. On the contrary, the last 10 epochs of training are performed training all the parameters in the DNN in order to fine-tune the whole structure. We set the learning rate to 0.001, the momentum coefficient to 0.9 and batch size to 256. Dropout after the two hidden FC layers is set to 0.5 for AlexNet-FCMAC and to 0.15 for AlexNet-FCMAM^2. For MAM^2-based layers, we employ the vanishing contributes method with $Q = 5$.

AlexNet-FCMAC and AlexNet-FCMAM^2 are trained on CIFAR-10 and CIFAR-100 datasets, while VGG-16-FCMAC and VGG-16-FCMAM^2 are trained on ImageNet. After training, for any structure under test, the two hidden FC layers are pruned, being the ones with the most parameters as explained in Section 3.3. All the pruning techniques introduced in Section 3.4.4 are employed where applicable. Top-1 and Top-5 accuracy for AlexNet-FCMAC and AlexNet-FCMAM^2 are shown in Figure 3.9 for CIFAR-10 and in Figure 3.10 for CIFAR-100, while results for VGG-16-FCMAC and VGG-16-FCMAM^2 are shown in Figure 3.11.

[1] we used tensorflow pre-trained model for this purpose.

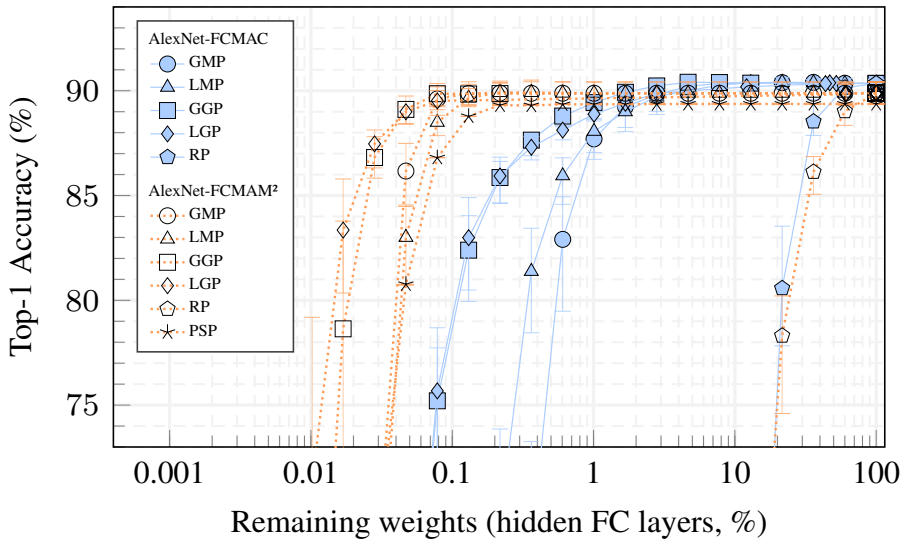
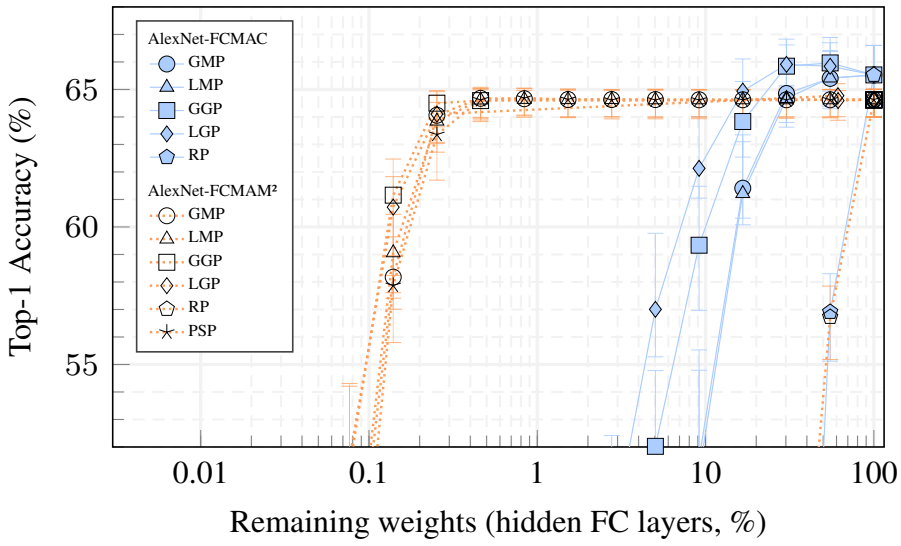
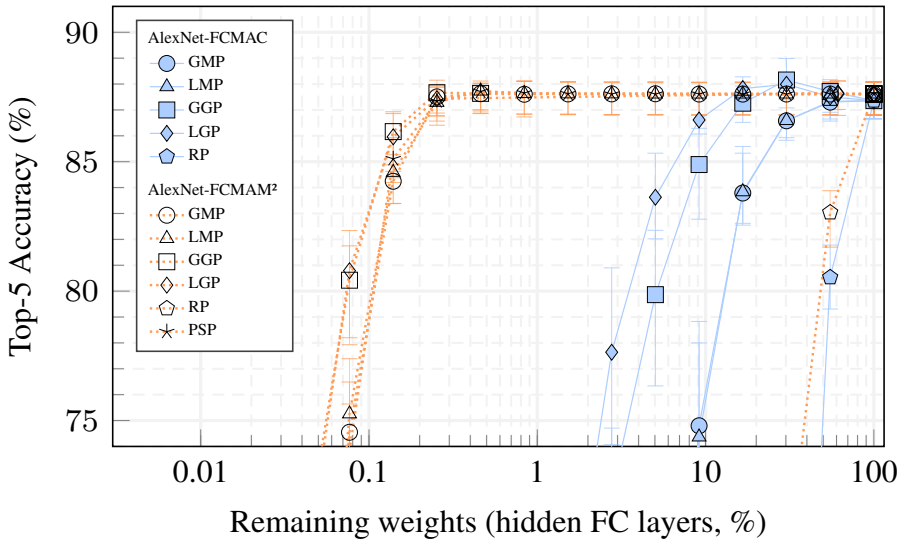


FIGURE 3.9: AlexNet + CIFAR10 + one-shot pruning: Top-1 accuracy vs percentage of kept weights in the two hidden FC layers. The points show the mean performance over 5 tests, with the maximum and minimum accuracy indicated by the error bars.



(a)



(b)

FIGURE 3.10: AlexNet + CIFAR100 + one-shot pruning: Top-1 (a) and Top-5 (b) accuracy vs percentage of kept weights in the two hidden FC layers. The points show the mean performance over 5 tests, with the maximum and minimum accuracy indicated by the error bars.

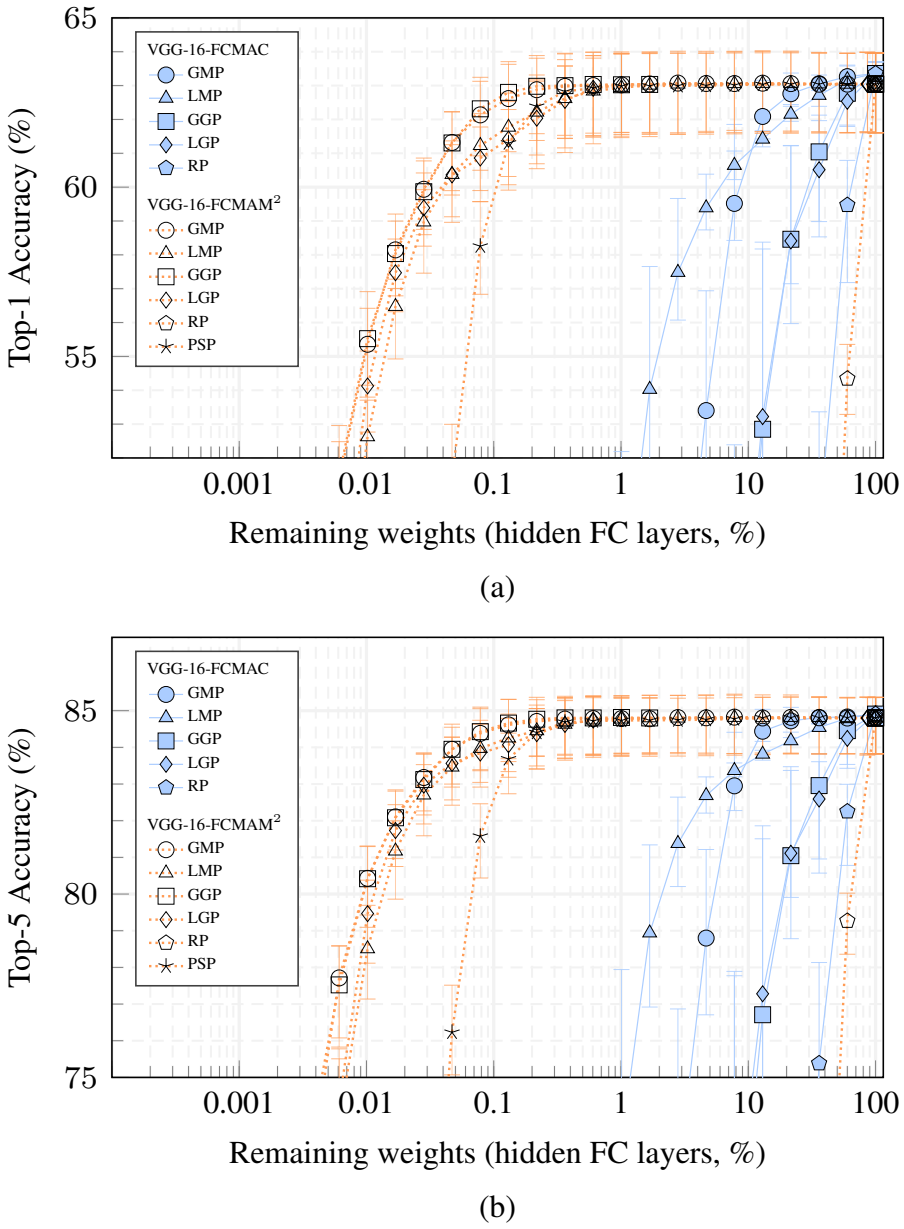


FIGURE 3.11: VGG-16 + ImageNet + one-shot pruning: Top-1 (a) and Top-5 (b) accuracy vs percentage of kept weights in the two hidden FC layers. The points show the mean performance over 5 tests, with the maximum and minimum accuracy indicated by the error bars.

Compared to what happens to MAC-based layers, MAM^2-based ones intrinsically rely on less weights, similarly to what is discussed in Section 3.2.2. In fact, when the DNN is pruned, AlexNet-FCMAM^2 and VGG-16-FCMAM^2 typically retain their baseline accuracy with a number of weights far lower when compared to AlexNet-FCMAC and VGG-16-FCMAC. In Table 3.1 a more detailed comparison is shown for one-shot pruning of all the DNNs and datasets of this paper, showing the number of weights kept for any tested DNN, dataset and pruning method with a loss not greater than 2% on the baseline accuracy (evaluated as the maximum mean accuracy reached on a given DNN model and task).

Regarding the computational complexity of the resulting DNNs, it is proportional to the

number of parameters, with a number of additions and multiplications about equivalent to the number of kept parameters. In fact, the computational cost of MAM^2 is about the same of MAC, as the operator $\mathcal{M}$ from (3.2) can be implemented as successive comparisons (that, at hardware level, are subtractions with a check of the sign of the result). Additionally, we can safely hypothesize that the overhead due to the necessity of manipulating sparse matrices is not comparable to the gain in terms of the reduction of the number of operation, being the proportion of kept parameters very small compared to the size of the full parameter matrices. Nonetheless, a further insight on this is out of the scope of this work.

TABLE 3.1: Remaining parameters in the hidden FC layers with one-shot pruning above 2% loss on baseline accuracy (overall reached maximum mean accuracy)

MAC parameters				MAM^2 parameters			
FC-DNN + MNIST – baseline accuracy 98.8%							
GMP	42.25%	LMP	42.25%	GMP	7.54%	LMP	4.90%
GGP	27.46%	LGP	27.46%	**GGP**	**3.19%**	**LGP**	**3.19%**
RP	100.00%			RP	100.00%	**PSP**	**3.19%**
AlexNet + CIFAR-10 – baseline accuracy 89.9%							
GMP	1.01%	LMP	1.01%	GMP	0.08%	LMP	0.08%
GGP	0.60%	LGP	0.60%	**GGP**	**0.05%**	**LGP**	**0.05%**
RP	36.00%			RP	60.00%	PSP	0.13%
AlexNet + CIFAR-100							
Top-1, baseline accuracy 65.9%							
GMP	30.25%	LMP	30.25%	**GMP**	**0.25%**	LMP	0.46%
GGP	30.25%	LGP	16.64%	**GGP**	**0.25%**	**LGP**	**0.25%**
RP	100.00%			RP	100.00%	PSP	0.46%
Top-5, baseline accuracy 88.2%							
GMP	30.25%	LMP	30.25%	**GMP**	**0.25%**	**LMP**	**0.25%**
GGP	16.63%	LGP	9.15%	**GGP**	**0.25%**	**LGP**	**0.25%**
RP	100.00%			RP	100.00%	**PSP**	**0.25%**
VGG-16 + ImageNet							
Top-1, baseline accuracy 63.4%							
GMP	12.96%	LMP	36.00%	**GMP**	**0.08%**	LMP	0.13%
GGP	60.00%	LGP	60.00%	**GGP**	**0.08%**	LGP	0.13%
RP	100.00%			RP	100.00%	PSP	0.22%
Top-5, baseline accuracy 84.9%							
GMP	7.78%	LMP	7.78%	**GMP**	**0.03%**	LMP	0.05%
GGP	36.00%	LGP	60.00%	**GGP**	**0.03%**	**LGP**	**0.03%**
RP	100.00%			RP	100.00%	PSP	0.13%

3.3.4 MAM² performance: iterative pruning

Performing additional training after one-shot pruning of a DNN can significantly improve its performance. Iterative pruning methods are an extreme application of this concept, as they consists in multiple consecutive pruning and training sessions to reach the desired network size. Nonetheless, these methods require multiple time consuming and resource hungry training processes of large structures, which for many applications might be impractical. The goal of this section is to show that the MAM^2 paradigm is effective also when iterative methods are applied.

In literature, many iterative pruning techniques have been proposed for unstructured pruning of FC layers. Here we use the Lottery Tickets hypothesis [24] as described in Section 1.2.

AlexNet-FCMAC and AlexNet-FCMAM^2 are trained with the same setup as in Section 3.3.3 and pruned using iterative Lottery Ticket method. Different proportions of removed parameters are employed through different tests, namely $\alpha = 0.9$, $\alpha = 0.8$, $\alpha = 0.7$ and $\alpha = 0.6$. Of course, with a lower value of α, more consecutive training processes are required to reach the desired network size, resulting in a higher computational cost.

In Figure 3.12 is shown the accuracy of AlexNet-FCMAC and AlexNet-FCMAM^2 with the application of Lottery Ticket. Overall, the performance of MAM^2 is equivalent to MAC, even tough a far lower number of consecutive training processes is required to reach the same results, possibly greatly improving the deployment time and cost effectiveness of a DNN for a given task.

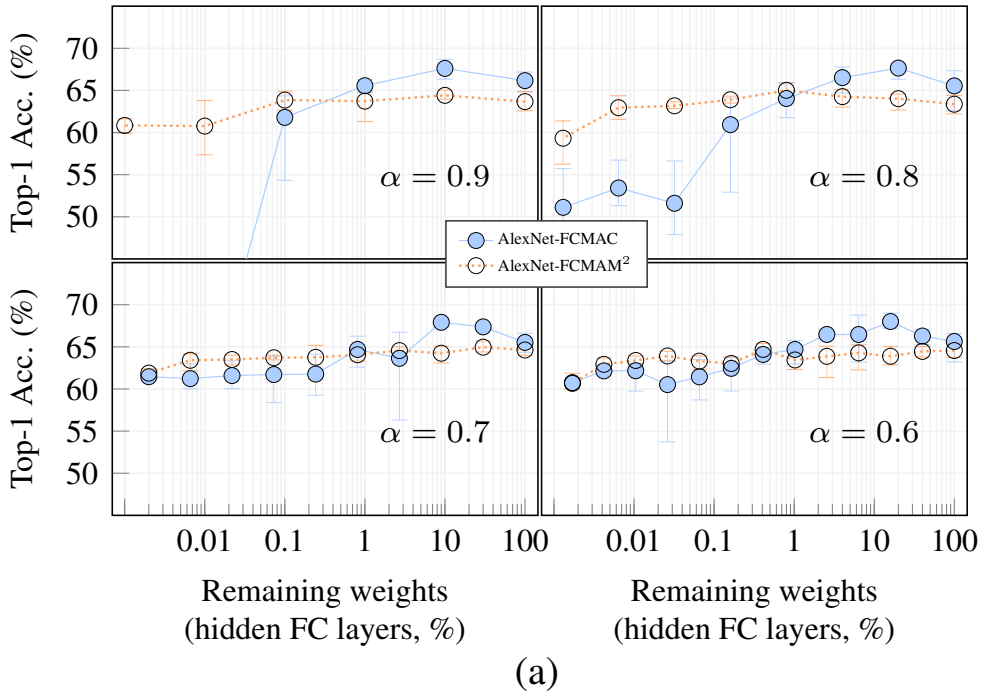


FIGURE 3.12: AlexNet + CIFAR-100 + Lottery Ticket pruning: Top-1 (a) and Top-5 (b) accuracy vs percentage of kept weights for tests with different values of α. The points show the mean performance over 5 tests, with the maximum and minimum accuracy indicated by the error bars.

In Figure 3.13 is shown the accuracy of the DNNs given a certain number of iterations, with less than 0.02% of the parameters in the hidden FC layers. When the number of iterations is low, MAM^2 performs much better than MAC. This is crucial for enabling the use of iterative pruning methods with large DNNs that need long training times and expensive hardware for

training.

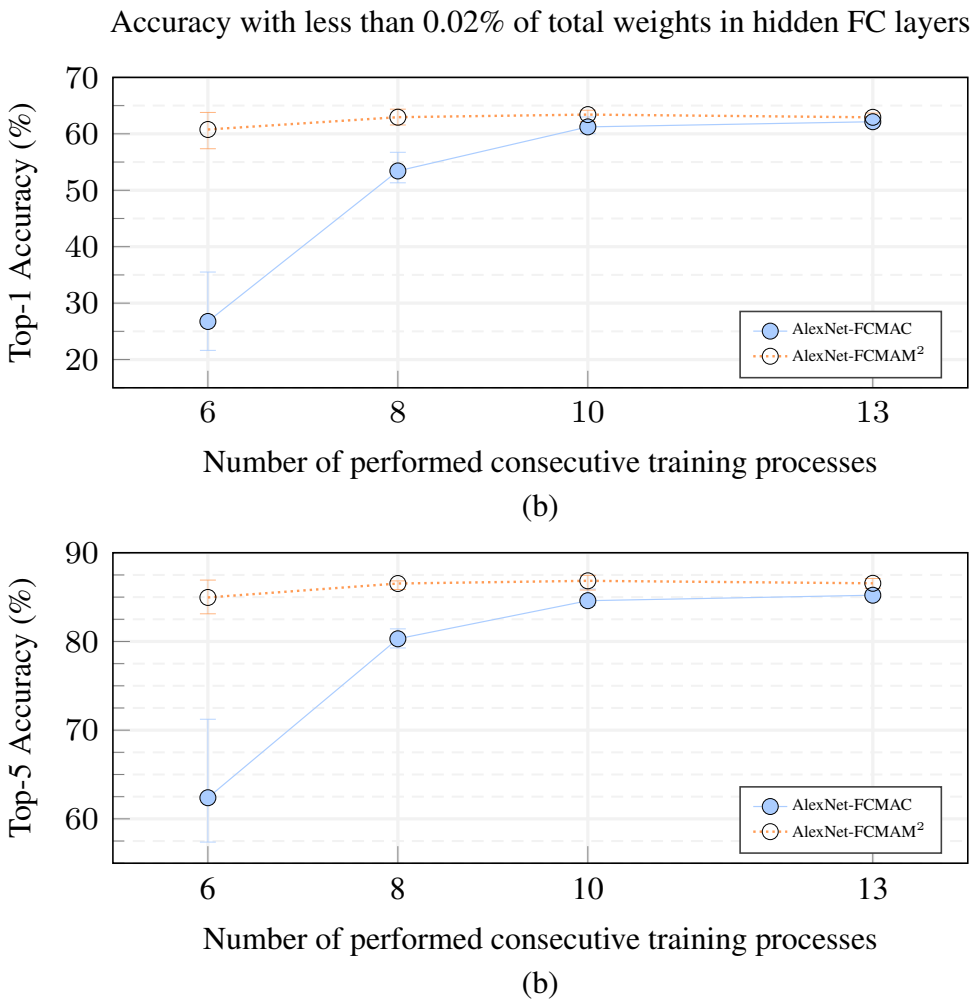


FIGURE 3.13: AlexNet + CIFAR100 + Lottery Ticket pruning: Top-1 (a) and Top-5 (b) accuracy with less than 0.02% of the total weights in the hidden FC layers, reached with different Δw and thus with a different number of consecutive training processes. The points show the mean performance over 5 tests, with the maximum and minimum accuracy indicated by the error bars.

3.4 Case study for implementation: autoencoder for ECG signals

In this section we present the performance of a large autoencoder, pruned by means of a MAM^2-based layer in order to fit it on devices with limited resources.

3.4.1 Autoencoder architecture

The autoencoder we consider in this work is composed of three one-dimensional convolutional layers followed by three FC layers and three one-dimensional transposed convolutional layers. The complete description of the DNN is shown in Figure 3.14.

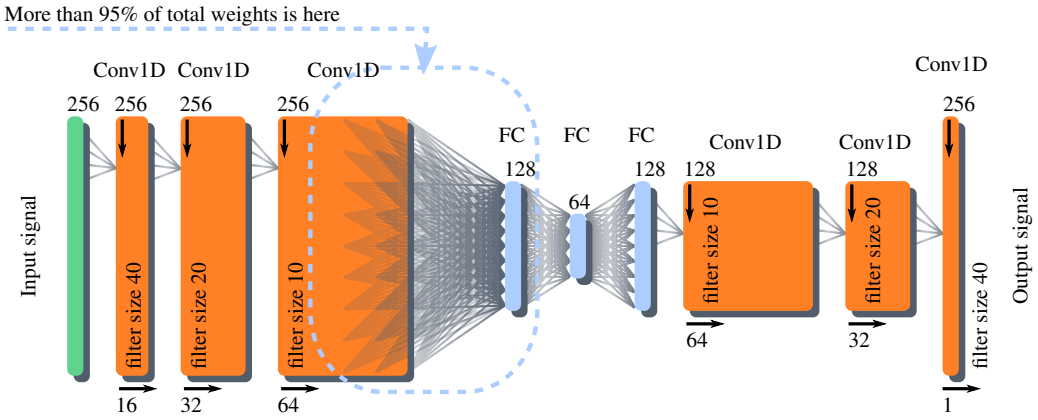


FIGURE 3.14: Structure of the autoencoder. The first FC layer contains more than the 95% of the total number of weights in the whole network and for this reason it is substituted with a MAM²-based layer in order to allow the implementation of the autoencoder on a small device. All the layers use a ReLU activation function.

The autoencoder contains more than 2 millions trainable parameters and almost 96% of them are contained in the first FC layer. We implement this layer both as a MAC layer and as a MAM^2 layer, and then we prune it to reduce the size of the DNN. We refer to the structure containing the MAC layer as AE-MAC, while we refer to the structure containing the MAM^2 layer as AE-MAM^2.

3.4.2 Dataset

The autoencoder is trained on a synthetic ECG dataset, generated as described in [46]. It is composed of 100 000 ECG windows of size $n = 256$, with 80 000 windows used for training, 10 000 for validation and 10 000 for test. The ECG has been resized so that any value is included in the interval between 0 and 1. The error on the decoded output of the autoencoder is evaluated with the Reconstruction Signal to Noise Ratio (RSNR), defined as

$$\text{RSNR} = 20 \log_{10} \frac{\|\boldsymbol{x}\|2}{\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|2} = \left(\frac{\|\boldsymbol{x}\|2}{\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|2} \right)_{\text{dB}} \tag{3.4}$$

where $\boldsymbol{x}$ is the input ECG window and $\hat{\boldsymbol{x}}$ is the output of the autoencoder, i.e., the reconstructed signal. To evaluate the average performance over a batch of inputs, we define the Average RSNR (ARSNR) as

$$\text{ARSNR} = \mathbf{E}\left[\left(\frac{\|\boldsymbol{x}\|2}{\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|2} \right)_{\text{dB}} \right] \tag{3.5}$$

where $\mathbf{E}[\cdot]$ is the expected value.

3.4.3 Quantization and training

We quantize and train the DNN with a standard quantization-aware approach based on the fake-quantization technique. We train both AE-MAC and AE-MAM² models for 200 epochs

with full-precision parameters only. Then we fine tune the models with additional quantization-aware training using uniform quantization for another 10 epochs. Weights are quantized to 8 bits in the range $[-1, +1]$ and inputs and outputs are quantized to 16 bits in the range $[-1, +1]$. Biases are kept as-they-are and converted a-posteriori to 32-bit fixed point. The Mean Squared Error (MSE) is selected as loss function and Adam Optimizer [47] is used with a starting learning rate of 0.001. The batch size is set to 128. Vanishing contributes is employed to train the MAM²-based layer with $Q = 30$.

3.4.4 Pruning the autoencoder

In this work we focus on one-shot unstructured pruning, i.e. the model is trained and then a set of selected weights is removed without taking into account the spatial disposition of the remaining interconnections. We prune the first FC layer both in AE-MAC and AE-MAM², as its impact on the memory footprint is dominant compared to the other layers. We select the Magnitude Pruning technique (MP) as pruning algorithm, consisting in scoring parameters based on their absolute magnitude. The idea behind this approach is the assumption that the smaller a parameter is in magnitude, the less it influences the output of the network. The resulting ARSNR vs the number of remaining parameters is summarized in Fig. 3.15, where the mean performance over 10 trained autoencoders is highlighted.

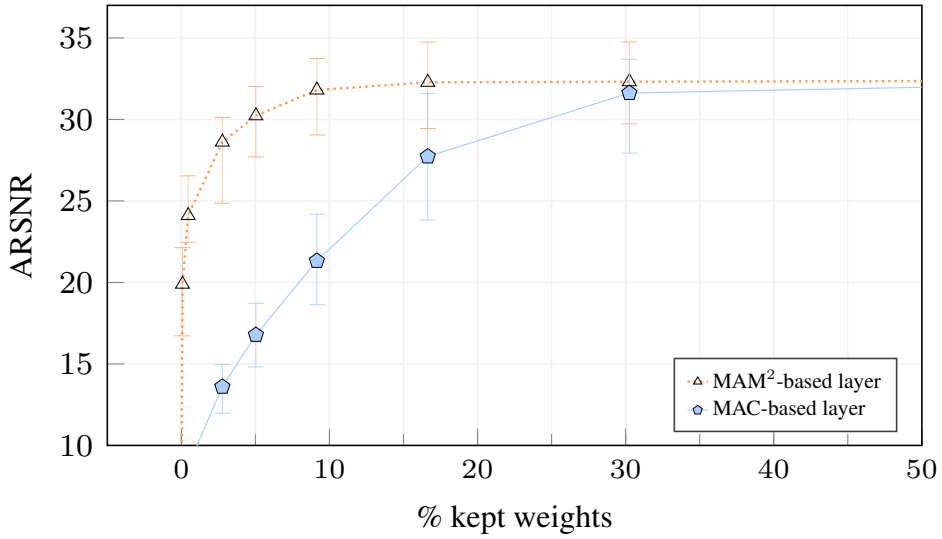


FIGURE 3.15: ARSNR vs the percentage of remaining parameters in the first FC layer of the autoencoder. The mean performance over 10 trainings is shown.

Results show that AE-MAM^2 is more robust to pruning compared to to AE-MAC. With 5% of remaining weights, the mean ARSNR over 10 trained models of AE-MAM^2 is still over 30 dB while it is only about 16.7 dB in the case of AE-MAC.

All the pruned weights are put to zero and since we are using an unstructured pruning technique to remove interconnections, the result is a sparse weight matrix. The sparsity represented in Figure 3.15 does not directly translate into an advantage in terms of memory

footprint since, in order to be exploited, it is necessary to employ a sparse representation of the matrix which inevitably introduces a computational and memory overhead.

In this work we implement AE-MAM² with 95% of its weights pruned[2]. This means that the number of remaining weights in the first FC layer is 106 424. In the MAM^2-based layer, these parameters are distributed over 87 neurons only resulting in 41 empty neurons whose output, consequentely, is only influenced by the bias value.

3.5 Hardware implementation of MAM² based layer

In this section, we test an actual implementation of the MAM^2-based layer on an edge device. Firstly, a representation for the sparse matrix is to be defined, then the memory footprint of the layer is computed and finally the AE-MAC and AE-MAM^2 autoencoders are implemented on an ARM Cortex-M4 based MCU, namely STM32F767ZI, and their performance is compared.

3.5.1 Sparse Matrix Representation

In order to benefit from the sparsity of the pruned weight matrix, while also minimizing the overhead due to the sparse representation, we need to accurately study an encoding method for the weights. In this work we use a sparse representation that consists in three vectors, in particular

- $\boldsymbol{w}$ contains all the weights in the matrix that have not been pruned arranged neuron after neuron;
- $\boldsymbol{o}$ contains the number of pruned interconnections between two kept weights of the same neuron (or the number of pruned interconnections preceding the first kept weight in the neuron);
- $\boldsymbol{s}$ contains the number of weights that are kept for each neuron.

An example of representation of sparse matrices is shown in Fig. 3.16.

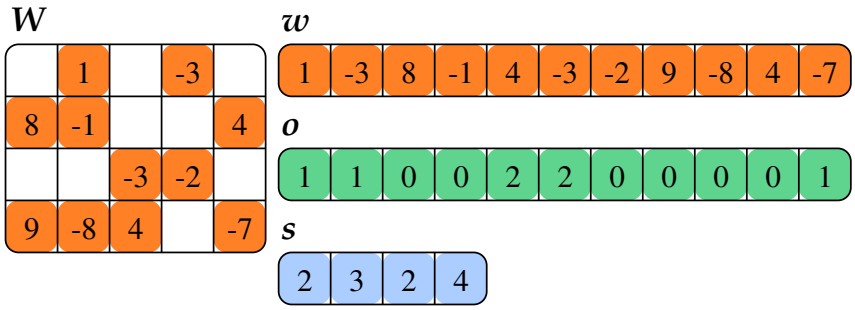


FIGURE 3.16: Example of representation for the description of sparse weight matrices. It consists in vectors $\boldsymbol{w}$, $\boldsymbol{o}$ and $\boldsymbol{s}$, containing the weights values, pruned interconnection between two weights and number of weights per neuron, respectively.

[2] we implement the best model in terms of accuracy obtained over the 10 trainings

In order to minimize the memory footprint, both the values in $\boldsymbol{w}$ and $\boldsymbol{o}$ are encoded with 8 bits. This means that the maximum number of consecutive weights that can be pruned is 255, i.e., the maximum value that can be represented with $\boldsymbol{o}$. If the gap between two non pruned weights is bigger, a zero is introduced in vector $\boldsymbol{w}$ as well as a 255 in $\boldsymbol{o}$ and the number of weights that are kept for that neuron is increased by one in vector $\boldsymbol{s}$. Furthermore, an additional zero is introduced also when a single weight remains in a MAM²-based neuron to have two separate values for the maximum and the minimum operations.

On the contrary, vector $\boldsymbol{s}$ contains 16-bit values. This is typically not a problem since the overhead caused by this vector is normally negligible compared to the size of $\boldsymbol{w}$ and $\boldsymbol{o}$.

The actual memory footprint of the MAC-based and MAM^2-based FC layer vs the ARSNR is shown in Fig. 3.17. Compared to the curves in Fig. 3.16, memory footprint is about twice as expected due to the sparse representation. Being the overhead proportional to the number of weights kept, the advantage of MAM^2 over MAC is accentuated.

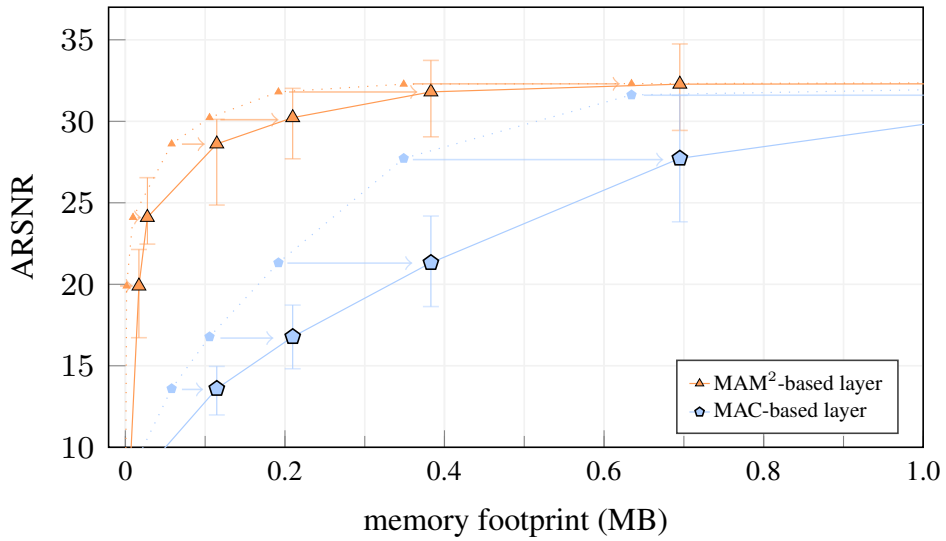


FIGURE 3.17: ARSNR vs the actual memory footprint of the first FC layer for the sparse representation of the weight matrix. Also, the curves showing the memory footprint with no sparse representation overhead are highlighted (i.e., the dotted lines take into account only the size of vector $\boldsymbol{w}$). This shows that the overhead is worse for MAC, being it proportional to the number of kept weights.

3.5.2 On-device implementation

The autoencoder is implemented on an ARM Cortex-M4 based MCU, namely STM32F767ZI. The device is characterized by an SRAM of 512 kB and a maximum operative frequency of 216 MHz. Custom C code is used to implement the different layers of the network, compiled with gcc with optimization options -Ofast and -loop-unroll. AXI interface, instruction cache and memory cache are enabled. The first FC layer, originally containing most of the parameters, is pruned as in Section 3.4 and encoded as in Section 3.5.1.

In Table 3.2 the memory footprint of each layer in the autoencoder is shown.

TABLE 3.3: Comparison between the accuracy of AE-MAC and AE-MAM^2 and time and energy consumption due to the first FC layer.

Model	ARSNR	Execution time	Energy
AE-MAM^2	32.6 dB	6.95 ms	691.5 µJ
AE-MAC	17.6 dB	4.32 ms	429.8 µJ

TABLE 3.2: Number of weights and biases together with the memory footprint of each layer in the pruned AE-MAM².

Layer	n. weights	n. bias	size
Conv1D_1	640	16	0.7 kB
Conv1D_2	10 240	32	10.4 kB
Conv1D_3	20 480	64	20.7 kB
FC_1	106 424	128	213.6 kB
FC_latent	8192	64	8.4 kB
FC_2	8192	128	8.7 kB
Conv1DT_1	640	64	0.9 kB
Conv1DT_2	40 864	32	41.0 kB
Conv1DT_3	1280	1	12.9 kB
Total	196 952	529	305.7 kB

The size of the pruned FC layer is computed taking into account the overhead due to the representation of the sparse matrix, in particular both vectors $\boldsymbol{w}$ and $\boldsymbol{o}$ contain 106 424 values encoded with 8 bits and the $\boldsymbol{s}$ vector contains 128 elements represented with 16 bits. In addition to the size of the parameters of the different layers, it must be guaranteed that at least 34 kB of the SRAM are still free, which is the memory footprint for the output buffer with the highest size (i.e., the input of the first FC layer).

It is important to highlight that the size of the original non-pruned autoencoder (both as AE-MAC and as AE-MAM^2) is almost 2.2 MB which is more than the size of the SRAM of the employed device. The accuracy of both AE-MAC and AE-MAM^2 is indicated in Table 3.3 as well as the execution time and the energy consumption of the first FC layer of the two models. Energy consumption is estimated as the power multiplied by the execution time, where the power is evaluated from typical electrical characteristics of STM32F767ZI indicated on datasheet, with a clock frequency of 180 MHz and a supply voltage of 1.7 V, giving a typical value of 99.5 mW.

Even tough the execution time of the MAM^2-based layer is about 50% higher than the one of the standard MAC-based one, mainly due to the more complex structure that requires the evaluation of both the maximum and the minimum values in a neuron instead of only accumulating all the values, the accuracy of the reconstructed ECG sample by AE-MAM^2 is significantly better than the one reconstructed by AE-MAC as shown in Fig. 3.18.

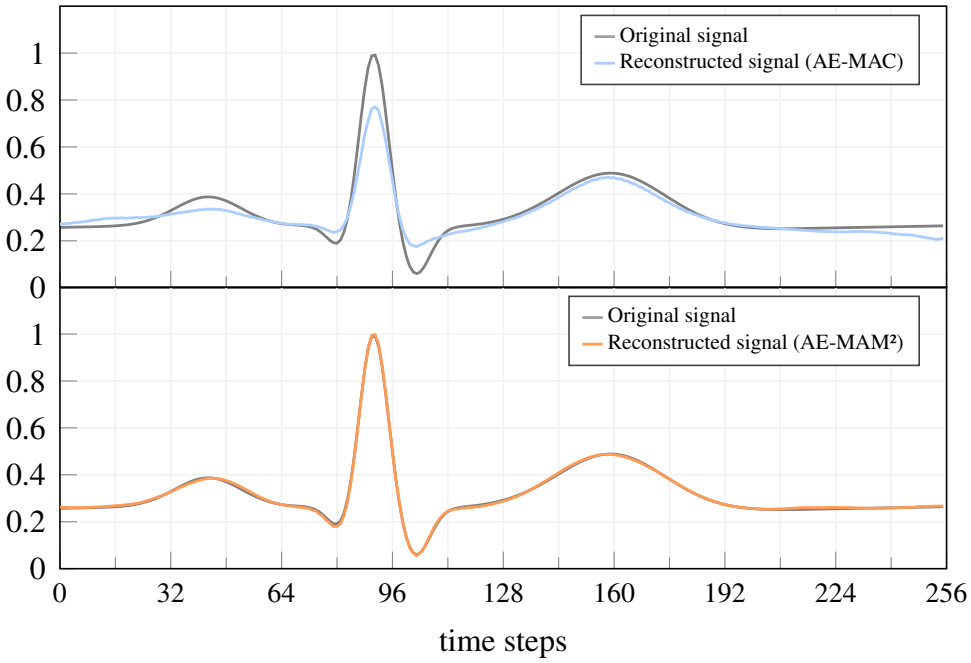


FIGURE 3.18: Example of original ECG signal and reconstructed signal at the output of AE-MAM^2 and AE-MAC implemented on device.

3.6 Chapter conclusion

We have presented and elaborated on a novel MAM^2 map-reduce paradigm which allows to build fully-connected layers naturally prone to aggressive pruning. These new type of layer are trained through vanishing gradient technique, that allows to seamlessly transform a classic MAC-based structure in a MAM^2-based one, resulting in a structure with a comparable performance in terms of accuracy. MAM^2-based layers can be pruned with multiple state-of-the-art non-structured techniques, ranging from one-shot methods to iterative ones. When employed with state-of-the-art DNN structures such as AlexNet and VGG-16 on multiple classification tasks, such as CIFAR-100 and ImageNet, MAM^2-based structures allow a further reduction of the total number of parameters compared to a classic MAC-based DNNs, together with a reduction of the computational cost necessary to obtain such pruned networks in the case of iterative methods.

Moreover, we have presented an implementation on an ARM Cortex-M4 based MCU of an autoencoder for ECG signals whose largest layer has been substituted with a MAM^2-based layer. The network, trained with a quantization-aware approach, has been quantized and the MAM^2-based layer has been pruned. The pruned structure has been encoded by means of a sparse matrix representation and fit on the MCU device, where the memory footprint, the computational time and the energy consumption of the pruned layer have been evaluated. Even tough there is a time overhead of MAM^2 structures over standard MAC with the same number of pruned parameters, MAM^2-based pruned layers, compared to MAC-based layers, guarantee a far better ECG reconstruction quality, while the MAC-based structure with the same reconstruction performance could not be fit on-device.

Chapter 4

A sparse approach to data: implementation of a Spiking Neural Network

Alternatives to classic MAC-based DNNs exist in the form of Spiking Neural Networks (SNNs) [48, 49, 50] that leverage on the possibility of representing data sparsely both in time and space. Still, these structures need for each neuron to integrate the input values coming from all the interconnections, thus still needing to pack a high number of parameters to work.

4.1 Event-based data: an overview

Event-based encoding of data is an interesting approach to image vision and object detection tasks in the field of Edge Computing. This type of data representation is intrinsically sparse both in space and time, thus allowing both a big reduction of the memory footprint necessary to store the information and the minimization of the energy cost due to its acquisition. Datasets encoded with this phylosophy can be recorded for example with Dynamic Vision Sensors (DVS), that are asynchronous cameras capable of acquiring only the changes happening to the scene they are looking at. The first prototype of DVS sensor was developed in 2006 [51] and since then event-based versions of widely used datasets and new collections of samples have been created [52, 53, 54, 55, 56, 57].

When working with event-based data, a promising approach is to use Spiking Neural Networks (SNN) [58, 48, 49]. These networks leverage the possibility of representing data sparsely both in time and space because each spiking neuron is stateful, as it integrates the inputs in its membrane potential variable. In machine vision, SNNs processing event-based data have been shown to solve many tasks reliably and efficiently, ranging from classification [59, 60] or hand-gesture recognition [52, 61] to object recognition [62, 63] and optical flow estimation [64].

Among different SNN models, Recurrent Spiking Neural Networks (RSNN) [50] are of great interest, as their recurrent nature extends the vanilla SNN's ability to process temporal signals to very long time-scales that go beyond the time constants of individual elements in the network [65]. In this work we present an RSNN implemented on a low-end Micro Controller Unit (MCU) with strict constraints in term of memory footprint and power consumption,

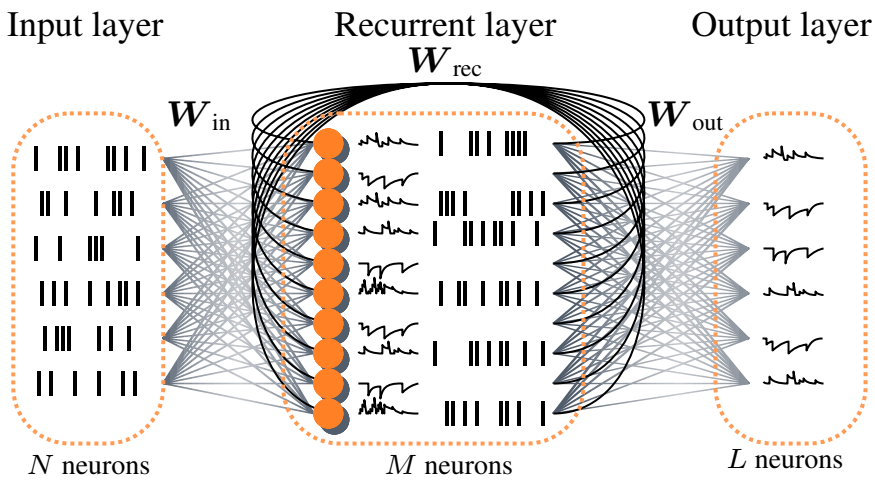


FIGURE 4.1: Scheme of the RSNN model employed in this work, composed of a input layer (for feeding input spikes), a hidden recurrent layer composed of LIF neurons and an output layer composed of leaky integrators.

inspired by recently proposed RSNN hardware-constrained designs [66]. By deploying an RSNN on a low-end MCU, we demonstrate the suitability of these structures on low-cost, low-budget commercial devices, which would allow fast deployment times compared to more complex custom hardware implementations.

4.2 Recurrent SNN architecture

In this section we describe the model of the RSNN that we are using for our implementation on MCU, based on Leaky Integrate and Fire (LIF) neurons and following the time-discrete models presented in [50] and used for a physical implementation in [66].

4.2.1 Model description

The RSNN model we use in this work is composed of an input layer, a hidden recurrent layer and an output layer, schematized as in Figure 4.1.

The input layer is simply defined as a series of external input spikes that are fed to the interconnections leading to the successive recurrent layer. The input spikes are encoded as a series of vectors $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^T$ defined for any discrete time step $t \in \{1, \cdots, T\}$, with $\boldsymbol{x}^t \in \{0,1\}^N$ where N is the number of input neurons. The value 1 represents a spike in input at time t, while 0 indicates the absence of activity.

On the contrary, the LIF neurons in the hidden recurrent layer are entities retaining potentials $\boldsymbol{v}^t \in \mathbb{R}^M$ which generate spiking outputs $\boldsymbol{z}^t \in \{0,1\}^M$, with M as the number of recurrent neurons. The potentials $\boldsymbol{v}^t$ are updated each time step following the rule

$$v_j^{t+1} = \alpha v_j^t + \sum_{i=1}^{N} w_{ji}^{\text{in}} x_i^t + \sum_{i=1, i\neq j}^{M} w_{ji}^{\text{rec}} z_i^t - \theta z_j^t \quad \text{for } j \in 1, \ldots, M \tag{4.1}$$

where v_j^t is the value of the potential of the j-th neuron, $\alpha \in (0,1)$ is a damping/leakage factor, w_{ji}^{in} is the value at row j and column i of the input weights matrix $\boldsymbol{W}^{\text{in}}$, x_i^t is the i-th value of input spikes vector $\boldsymbol{x}^t$, w_{ji}^{rec} is the value at row j and column i of the recurrent weights matrix $\boldsymbol{W}^{\text{rec}}$, z_i^t is the i-th value of output spikes vector $\boldsymbol{z}^t$ and θ is a threshold value.

Moreover, the spiking outputs generated by the hidden recurrent neurons behave accordingly to the following equation

$$z_j^t = \mathcal{H}\left(v_j^t - \theta\right) \quad \text{for } j \in 1, \ldots, M \tag{4.2}$$

where $\mathcal{H}(\cdot)$ is the step Heaviside function defined as

$$\mathcal{H}\left(v_j^t - \theta\right) = \begin{cases} 0 & \text{for } v_j^t \leq \theta \\ 1 & \text{for } v_j^t > \theta \end{cases} \tag{4.3}$$

The first term on the right-hand-side of (4.1) introduces a damping/leakage on the potential of the neuron, meaning that when there are no spikes in input, the absolute value of the potential decreases exponentially until it reaches 0.

Then, the second and third terms on the right-hand-side of (4.1) integrate the input spikes $\boldsymbol{x}^t$ and the hidden recurrent spikes $\boldsymbol{z}^t$, respectively. When a spike is detected, the value of the corresponding weight is added to the current neuron potential. The input spikes are sent to the model from an external source, as described before. The hidden recurrent spikes are generated from (4.2) and then fed back to hidden neurons through recurrent connections, excluding self-recurrence.

Finally, the Heaviside function in (4.2) simply describes the firing mechanism of the neuron: when the neuron potential grows bigger than the threshold value θ, a spike is fired (i.e., $z_j^t = 1$) and the neuron potential is reset (the last term on the right-hand-side of (4.1) serves this purpose).

The last part of the model is composed of the output layer, which can be seen as a set of non-firing leaky integrators. The value of the potential of the output neurons is updated as

$$y_k^{t+1} = \kappa y_k^t + \sum_{i=0}^{L} w_{ji}^{\text{out}} z_i^t \quad \text{for } k \in 1, \ldots, L \tag{4.4}$$

where y_k^{t+1} is the k-th output of the network (encoded as a value of potential), $\kappa \in (0,1)$ is a damping/leakage factor and w_{ji}^{out} is the value at row j and column i of the output weights matrix $\boldsymbol{W}^{\text{out}}$. The output neuron is fed by the output spikes of the hidden recurrent layer $\boldsymbol{z}^t$.

The leakage factors α and κ define the leakage time constants of the recurrent and output layer units, respectively τ_{rec} and τ_{out}, as $\alpha = \exp\left(-\Delta t / \tau_{\text{rec}}\right)$ and $\kappa = \exp\left(-\Delta t / \tau_{\text{out}}\right)$, where Δt is the time interval between two discrete time-steps, i.e. the temporal resolution.

The RSNN is trained with the Back Propagation Through Time (BPTT) algorithm, it is thus important that the gradients are defined everywhere. As (4.2) introduces a discontinuity, a pseudo derivative function is employed as suggested in [50], defined as

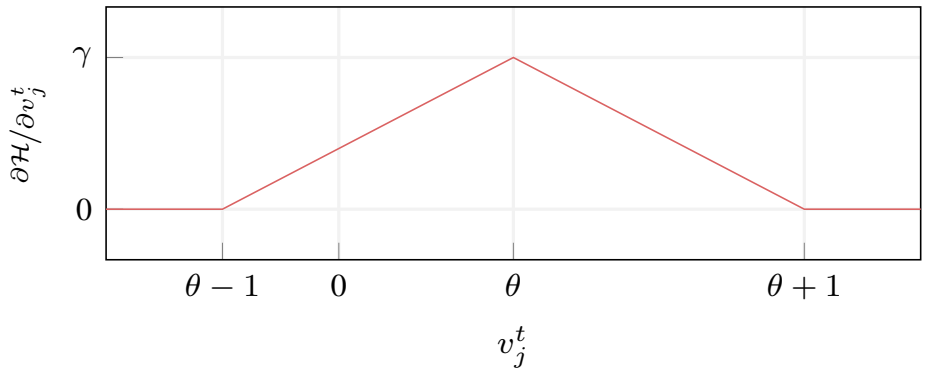


FIGURE 4.2: Plot of the pseudo derivative function in (4.5).

$$\frac{\partial \mathcal{H}}{\partial v_j^t} \triangleq \gamma \max\left(0, 1 - \left|v_j^t - \theta\right|\right) \tag{4.5}$$

where γ is a factor controlling the strength of the pseudo derivative, as illustrated in Figure 4.2.

4.3 Dataset and Performance

We want to employ the RSNN model with an event-based classification task, in order to verify its capabilities for event-based image vision tasks. In this section we present the dataset employed, the training setup and its performance on the test set. Training and preliminary tests are performed within a PyTorch-based framework.

4.3.1 N-MNIST event-based dataset for digit recognition

The N-MNIST dataset [67] is the neuromorphic version of the well-known MNIST dataset [41]. It is composed of a total of 70 000 samples (55 000 for training, 5000 for validation and 10 000 for testing). Each sample is associated with a number from 0 to 9 and is a collection of events with a duration of about 350 ms, a time resolution of about 1 µs and a spatial resolution of 34×34 pixels. Each event is represented as a $+1$ or as a -1, depending on the direction of variation of the associated pixel. The way events are collected takes inspiration from the biological phenomenon known as saccade, defined as a quick and simultaneous movement of both eyes between multiple phases of fixation in the same direction. In particular, each digit image from the MNIST dataset is displayed on a monitor and then recorded by a motor-driven DVS camera, moving in three different directions in a triangular form.

In this work, the resolution of the recordings is down-sampled to 17×17 with a remapping of all the events in order to decrease the number of input neurons required in the RSNN and consequentially to reduce the memory footprint and computational effort on MCU. For the same reason, the number of time-steps T is reduced to 300 by discarding any event eventually exceeding 300 ms and by down-sampling the time resolution to $\Delta t = 1$ ms, i.e., the events contained in each 1 ms time bin are merged together.

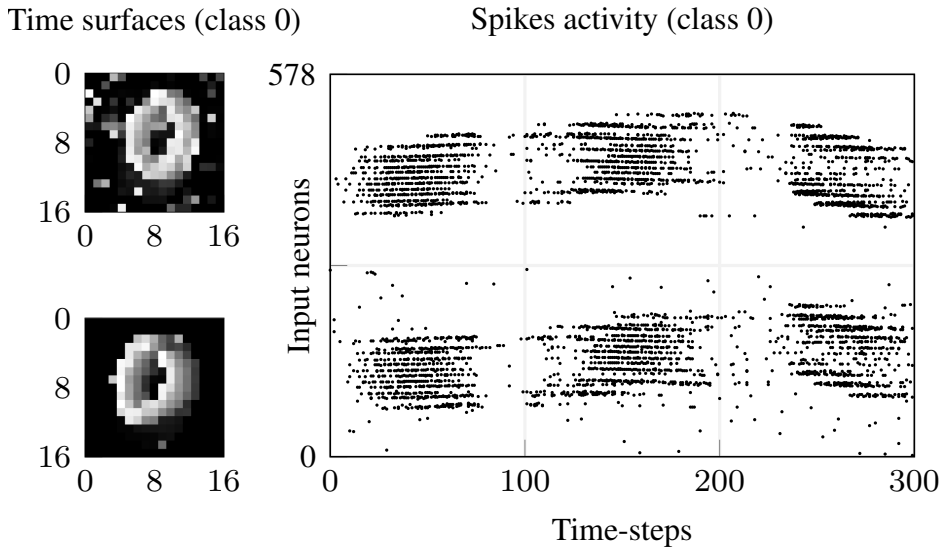


FIGURE 4.3: An example of sample of class 0. On the left, the time surfaces for positive and negative events, obtained by integrating all the events over 300 ms. On the right, the spikes activity of the sample.

4.3.2 Training setup and preliminary performance

We train and test an RSNN with $M = 100$ recurrent neurons and $L = 10$ output neurons, as the number of classes to be recognized. The hyperparameters selected for the network are $\theta = 0.6$, $\tau_{\text{rec}} = 250\,\text{ms}$, $\tau_{\text{out}} = 20\,\text{ms}$ and $\gamma = 0.3$.

The event-driven nature of SNNs requires the use of spike-trains as input. Because of this, a couple of input neurons is associated to each pixel in the N-MNIST dataset, one neuron for the positive events $+1$ and the other for the negative events -1 associated with that position. In fact, we set the number of input neurons of the RSNN to $N = 578$, defined as the number of pixels in the sample (17×17) multiplied by two to include both positive and negative events. An example of input sample is represented qualitatively in Figure 4.3.

The RSNN is trained using a standard BPTT approach [68] with Adam optimizer [40], with the minimization of a cross-entropy (CE) loss function between the output potentials $\boldsymbol{y}^t$ and the target $\boldsymbol{y}^t_{\text{true}}$ label averaged for all the time-steps $t \in \{1, \dots, T\}$. For validation/test, the selected class is considered as the one corresponding to the output neuron with the highest average output potential.

In order to minimize the spiking activity of the recurrent neurons, a regularization contribute is added to the loss function computed as the L^2 norm of the spike-trains (i.e., $\boldsymbol{z}^t$) generated by the recurrent neurons, averaged for all the time steps. By introducing this contribution, the number of spikes generated while inferring the complete test set is reduced by about 90%. This introduce a significant increase of the sparsity of the hidden layer activity while also reaching the same classification accuracy.

Additionally, we want the resulting network to use quantized parameters, meaning that we need to employ quantization-aware training techniques. Because of this, during training we perform fake quantization, i.e., forward-pass phase is performed using dynamically quantized 8-bit weights.

With a batch size of 5 and a learning rate of 1×10^{-4}, after 50 epochs the accuracy we achieve on the test set is 97.2 %, which is near state-of-the-art performance on the N-MNIST dataset with more complex structures [69, 70, 71, 72].

4.4 Implementation and performance

4.4.1 Implementation on MCU

The RSNN model as described in Section 4.2.1 is implemented by software on an ARM Cortex-M4 based MCU, namely the STM32F767ZI. This device features an SRAM of $512\,\mathrm{kB}$ and a maximum operative frequency of $216\,\mathrm{MHz}$. Spikes updates are applied column-first, i.e., for each spike the potential update is applied sequentially to all the neuron potentials, and the leakage is applied to all the neurons potentials at each time-step. C code is compiled with gcc and optimized by means of -Cfast and -loop-unroll options. STM32 AXI interface is enabled along with data and instruction caches.

Each input spike which is sent to the MCU is encoded in 32 bits, with 16 bits used for the input neuron address and 16 bits that indicate the number of time steps from the previous spike. Conversely, recurrent spikes are generated internally each time step and are encoded as an array of 8 bit values indicating the recurrent neuron address.

Internally, the weights are encoded with 8 bits (sign bit with 7 fractionary bits) while the neuron potentials are encoded with 32 bits (sign bit, 16 integer bits and 15 fractionary bits). Given the bits alignment used for neuron potentials, no overflow detection mechanism is required as the numerical range is far greater than what can be reached in practice.

4.4.2 Performance

In order to assess the performance of the RSNN implementation on MCU, the test set composed of $10\,000$ samples of 300 time-steps each is inferred, with an accuracy in line with section 3.4 and the computational times required by each operation is measured with an on-chip counter.

In Figure 4.4 the computational time with an operative frequency $f_{\mathrm{CLK}} = 144\,\mathrm{MHz}$ per time-step for different samples is shown, along with the spiking activity of the model. Computational time is split among its different contributes, i.e., application of leakages to recurrent and output neurons, application of input spikes and recurrent spikes to the recurrent neurons and to the output neurons and recurrent spikes generation. Most of the computational time is due to the application of the spikes to the neuron potentials, which is of course proportional to the number of spikes to be integrated. On the other hand, other contributes are almost negligible.

In Table 4.1 average and maximum power consumption of the MCU, computational time for a single time-step and energy absorbed by the device for the update of a time-step are shown. Power consumption is calculated with the values declared on datasheet with a supply voltage of $1.7\,\mathrm{V}$, STM32 AXI interface and cache enabled, all peripherals disabled. It should be noted that most of the power contribution comes from an internal voltage supply of about

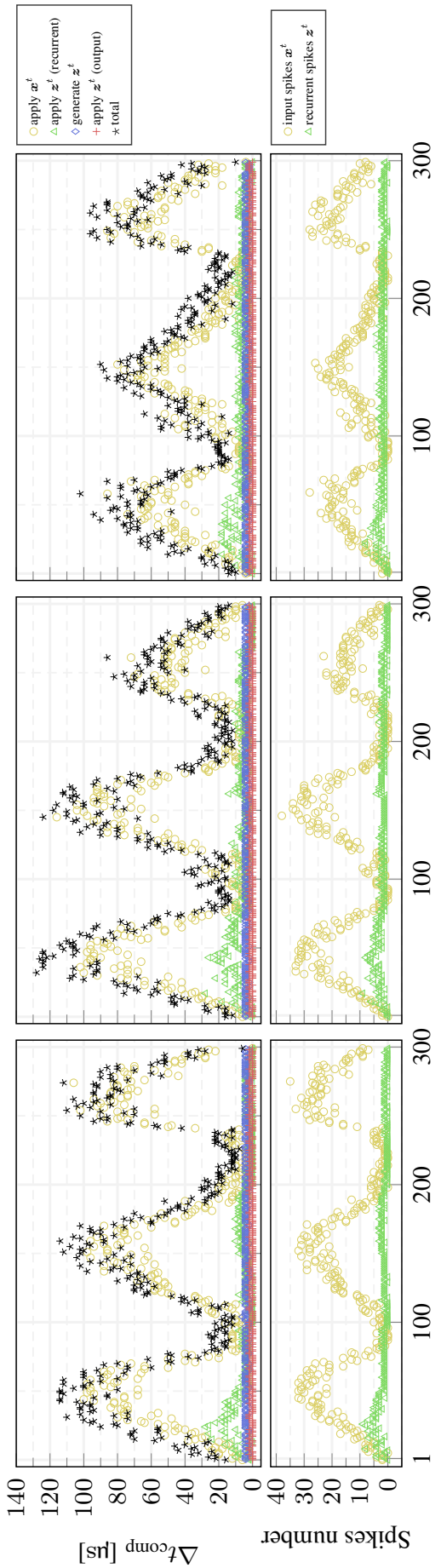


FIGURE 4.4: Computational time per time step for some inferred samples with $f_{\text{CLK}} = 144\,\text{MHz}$, compared with the input and recurrent spiking activity. The computational time is split in different contributes, and the contributes due to the application of the leakage are not shown as they are below 1 µs. The greatest contributes are due to the input spikes whose number is high compared to the recurrent spikes. Their trend is coherent to the three movements performed by the DVS camera while recording N-MNIST.

TABLE 4.1: Power consumption of STM32F767ZI, with the computational time and energy consumption for a single time-step update of the RSNN

f_{CLK}	P_{typ}	$\Delta t_{\text{comp, avg}}$	$E_{\text{comp, avg}}$
25 MHz	13.1 mW	292.5 µs	3.8 µJ
60 MHz	26.8 mW	121.5 µs	3.3 µJ
144 MHz	58.7 mW	49.4 µs	3.1 µJ
169 MHz	83.6 mW	43.5 µs	3.7 µJ
180 MHz	99.5 mW	40.6 µs	4.1 µJ
f_{CLK}	P_{max}	$\Delta t_{\text{comp, max}}$	$E_{\text{comp, max}}$
25 MHz	18.5 mW	1082.4 µs	21.9 µJ
60 MHz	34.1 mW	444.7 µs	15.9 µJ
144 MHz	70.1 mW	207.0 µs	14.8 µJ
169 MHz	98.1 mW	161.3 µs	16.1 µJ
180 MHz	116.5 mW	150.4 µs	17.8 µJ

1.2 V, as indicated on datasheet. The average and maximum computational times are evaluated over all the time-steps for all the 150 samples, while the energy consumption is simply the product of power consumption and time.

Maximum computational times are aboiut $1\,\text{ms}$ for the lowest operational frequency tested, while it can reach very low values for an higher f_{CLK}, enabling the use of this type of neural network models on low-cost, low-resource devices with fast deployment times when compared to custom hardware implementations.

Finally, the memory footprint of the RSNN model on MCU is mostly due to its parameters, which, in this work, use $68.8\,\text{kB}$ of space.

4.5 Chapter conclusion

We trained and tested an RSNN model to solve an event-based image vision task, using the N-MNIST dataset for digit recognition. The model is implemented on a low-cost, low-power MCU device with strong resource constraints, in order to validate the benefits of this approach using standard commercial devices and fast deployment times. With an accuracy of 97.2%, the maximum computational time per time-step can be as low as $150.4\,\mu\text{s}$ with an average energy consumption per time-step of $4.1\,\mu\text{J}$ at $f_{\text{CLK}} = 180\,\text{MHz}$, showing that it is possible to employ software-based RSNNs for inference in real-time image vision applications.

Part II

An Edge Computing Application for DNNs: Decoder for Compressed Sensing

Chapter 5

Compressed Sensing Decoder based on Deep Neural Oracles

Nowadays an increasing number of applications require tight constraints in terms of energy consumption and computational cost. An important example is given by medical systems designed to continuously monitor a patient condition without running out of energy [73] as in Figure 5.1. Due to this, it is essential to find a way to acquire and transmit important information in a way capable of reducing as much as possible the required energy.

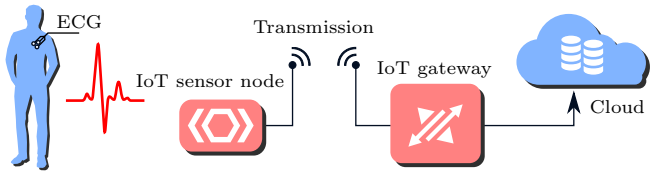


FIGURE 5.1: Example of medical application of IoT: a sensor node continuously acquire ECG data from a patient and transmit it so that it can be analyzed. The analysis can happen on an IoT device on the edge or directly in the cloud on servers. (© 2022 IEEE)

The use of the Compressed Sensing (CS) approach [6, 7, 8] allows for such a desired energy reduction. CS is a relatively new paradigm for the acquisition/sampling of signals that violates the intuition behind the theorem of Shannon. In fact, CS theory states that, under surprisingly broad conditions, it is possible to reconstruct certain signals or images using far fewer samples or measurements than they are used with traditional methods.

To enable this, CS is based on two concepts: *sparsity*, which is related to the signals of interest, and *incoherence*, which relates to the methods of measurement/acquisition/sampling. Sparsity expresses the idea many natural signals have a very parsimonious representation when expressed in an appropriate *sparsity basis* and is in fact commonly verified in many hysical phenomena. Incoherence expresses the idea that a reduced number of acquisitions of a waveform that have a sparse representation in an appropriate basis, which is made in a domain that is incoherent with it, allows always to capture the entire signal information.

With these premises, CS can simultaneously acquire and compress an input signal with a simple linear projection on a set of sensing waveforms. In this way, values called measurements are obtained, and even if their number can be much smaller than the number of samples

required according to the Nyquist-Shannon theorem, they are enough for a correct signal reconstruction.

With CS, it is has been possible to devise protocols for sampling or measurement [74, 75] which capture the information content but require a number of measurements comparable to the number of non-zero coefficients in the expression of the signal of interest with respect to its appropriate sparsity base. Consequently, the most significant feature of these sampling procedures is that they allow a sensor to capture the information content of a signal without going through the acquisition of its entire profile, thus performing acquisition and compression at the same time. In other words, CS is a very simple and efficient procedure to sample sparse signals at a reduced rate, using much fewer resources compared to standard sampling required for A/D conversion and ensuring a large energy saving at acquisition time.

This is particularly interesting in many applications including electromagnetic inverse scattering [76], structural health monitoring [77] and image processing [78]. Additionally, of particular interest is that many signals of interest in biomedical applications enjoy the sparsity property and can, therefore, be efficiently acquired using CS, i.e., by using less energy, in less time and/or with fewer sample. In fact, an area for which CS is particularly useful is the design of Body Area Network (BAN) nodes, which aim to efficiently acquire biosignals. This has already been demonstrated for Electrocardiographic (ECG) [79, 20], Electromyographic (EMG) [80] and Electroencephalography (EEG) [81] signals. CS has also been shown useful for magnetic resonance imaging (MRI) waveforms acquisition [82], where the main advantage stems from reducing the overall acquisition time [83].

All these advantages in the acquisition phase are balanced by the increase in complexity necessary for the signal reconstruction compared to the simple low-pass filtering needed in a standard D/A conversion. In fact, reconstruction in a CS frameworks boils down to solving the problem (which is also fundamental in a number of heterogeneous applications) of recovering an n-dimensional sparse signal $\boldsymbol{x}$ from a set of m measurements $\boldsymbol{y}$ that represent the output of the CS under-sampling encoding, i.e., with $m < n$. More specifically, one needs to find the sparsest n-dimensional vector $\boldsymbol{x}$ among the infinite solution of the ill-defined system $y = \mathcal{L}(\boldsymbol{x})$, where $\mathcal{L} : \mathbb{R}^n \mapsto \mathbb{R}^m$ is a linear dimensionality-reduction operator, which, regrettably, is an NP-hard problem. However, thanks to [84], the solution can be obtained by solving a minimization problem, called *Basis Pursuit* (BP)[1], using linear programming. In other words, the result in [84] is fundamental since it allows to obtain a solution for the BP problem in polynomial time, thus making the use of CS practical. Yet, the computational resources needed by the numerical algorithm solving BP may be so demanding to make its solution practically unfeasible in low-complexity nodes, like a typical BAN gateway. To cope with this, several dedicated BP/BPDN solvers have been proposed, such as the Spectral Projected Gradient for L1 Minimization (SPGL1) [85], and the Generalized Approximate Message Passing (GAMP) [86]. Alternative solutions rely on the observation that the main issue in the computation of $\boldsymbol{x}$ is not finding a generic solution to $y = \mathcal{L}(\boldsymbol{x})$, but to find the sparsest one. Starting from this, further computational cost reduction can be achieved by generating solutions which iteratively adjusts their sparsity at each step. Different heuristics

[1]The problem is called *Basis Pursuit with DeNoising* (BPDN) if noise is also considered

may be used to promote sparsity and give raise to different methods, such as the Orthogonal Matching Pursuit (OMP) [87] and the Compressive Sampling Matching Pursuit (CoSaMP) [88], often used for their low computational complexity [89]. Further to those methods, schemes have been proposed in which the recovery algorithm is adapted to the class of signals to acquire (see, e.g., [90, 91, 92] where the decoder stages are tuned on the reconstruction of ECGs). These schemes exploit statistical priors on the signal to favor reconstructions close to what is typical in the class of acquired signals.

More recently it has been demonstrated that additional advantages in terms of a smaller computational complexity or improvement in the quality of the reconstructed signal can be obtained by adopting Deep Neural Networks (DNNs) for reconstruction [93, 94, 95, 96, 97, 98, 99, 100]. More specifically, in [97], authors have shown a probabilistic relation between CS and a stacked denoising autoencoder (SDA) implemented as 3-layer neural network. Once adequately trained, the SDA can directly recover a sparse image from its linear (or mildly non-linear) measurements and has offered, in some cases, advantages in terms of quality of the reconstructed images compared to the most common greedy reconstruction algorithms. A similar approach that employs fully-connected DNNs can be found in [98], where CS has been applied to videos, and the proposed approach enables fast recovery of video frames at a significantly improved reconstruction quality. In [99], authors have proposed a DNN called ISTA-Net and inspired by the Iterative Shrinkage-Thresholding Algorithm (ISTA) [101], which has been designed to optimize the solution of BP to reconstruct compressed images. Another deep learning model (BW-NQ-DNN) applied to CS acquisition/reconstruction of neural recording has been presented in [100]. Here, three networks have been jointly optimized to perform a binary measurement matrix multiplication, a non-uniform quantization, and reconstruction, respectively. Despite the advantage shown in terms of quality of reconstruction, this approach has a few drawbacks: *i*) it requires a pre-processing stage detecting signal peaks, which adds complexity to the encoder and specializes it to spiky signals; *ii*) it quantizes CS measurements after a programmable non-linearity, which adds further complexity.

In this part of the work, we describe an innovative use of DNNs in a CS-based acquisition and reconstruction framework we worked on [102, 103, 104, 21, 105]. Unlike all the cases mentioned above that use DNNs to reconstruct the input signal directly, our model only provides a *divination* of the support of the input signal, i.e., the positions of the non-null components of the signal expressed along the sparsity basis. Our approach not only improves reconstruction quality compared to standard techniques but also introduces a *self-assessment capability* that allows estimating *on the fly* the quality of reconstruction. Furthermore, with our method, signals can be successfully reconstructed even when they refer to very short acquisition windows, a crucial feature that a further reduction of the complexity and a mixed-signal implementation of the acquisition stage.

To the best of our knowledge, this framework is the first to use a DNN for support identification, and one of the few actually using a DNN to improve the reconstruction of signals sampled using CS, which are not images.

5.1 Compressed Sensing basics

Let us refer to the scheme in Figure 5.2 and assume to work by chopping input waveforms into subsequent windows, each of which is represented by a set of its samples $\boldsymbol{x} = (x_0, \dots, x_{n-1})$ collected at Nyquist rate that we see as a vector $\boldsymbol{x} \in \mathbb{R}^n$. CS hinges on the assumption that $\boldsymbol{x}$ is κ-*sparse*, i.e., in the simplest possible setting, that an orthonormal matrix $\boldsymbol{D}$ exists (whose columns are the vectors of the sparsity basis) such that when we express $\boldsymbol{x} = \boldsymbol{D}\boldsymbol{\xi}$, then the vector $\boldsymbol{\xi} = (\xi_0, \dots, \xi_{n-1})$ does not contain more than $\kappa < n$ non-zero entries.

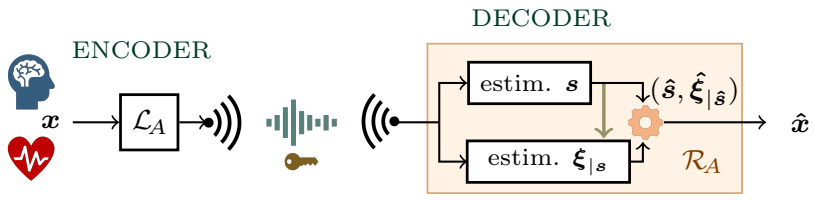


FIGURE 5.2: General scheme of an encoder-decoder pair based on CS. In the decoder, we distinguish the estimation of the signal support $\boldsymbol{s}$, and the estimation of the non-zero coefficients $\boldsymbol{\xi}_{|\boldsymbol{s}}$. Classical decoders perform both estimations simultaneously. The approach we present in this chapter first estimates $\boldsymbol{s}$ and then $\boldsymbol{\xi}_{|\boldsymbol{s}}$.

The fact that $\boldsymbol{x}$ depends only on a number of scalars that are less than its sheer dimensionality hints at the possibility of compressing it. CS does this by applying a linear operator $\mathcal{L}_{\boldsymbol{A}} : \mathbb{R}^n \mapsto \mathbb{R}^m$ depending on the *sensing* (or acquisition, or encoding) matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with $m < n$ and defined in such a way that $\boldsymbol{x} \in \mathbb{R}^n$ can be retrieved from $\boldsymbol{y} = \mathcal{L}_{\boldsymbol{A}}(\boldsymbol{x}) \in \mathbb{R}^m$. The ratio n/m is the *compression ratio* and will be indicated by CR. In particular, the linear operator (encoder) output is a m-dimensional measurement vector $\boldsymbol{y}$ obtained by projecting $\boldsymbol{x}$ over the rows of $\boldsymbol{A}$ as

$$\boldsymbol{y} = \boldsymbol{A}(\boldsymbol{x} + \boldsymbol{\nu}) \tag{5.1}$$

where the vector $\boldsymbol{\nu}$ includes noise and non-idealities in the system implementation.

It can be intuitively accepted that the larger the κ, the larger the m is needed to guarantee that $\boldsymbol{x}$ can be retrieved from $\boldsymbol{y}$, and thus the lower the achievable CR. This relationship is asymptotically identified by CS theory as $m = \mathcal{O}\left(\kappa \log\left(n/\kappa\right)\right)$ [8]. In finite and practical cases, one may often aim at using an m value proportional to κ. Nevertheless, the worst-case theoretical guarantees fail for $m < 2\kappa$. In fact, despite the infinite number of counterimages of $\boldsymbol{y} = \mathcal{L}_{\boldsymbol{A}}\left(\boldsymbol{\xi}\right)$, the first prerequisite for the recoverability is that when we add the κ-sparsity prior only one of them survives. Hence, given any two κ-sparse vectors $\boldsymbol{\xi}'$ an $\boldsymbol{\xi}''$ it cannot be $\boldsymbol{y} = \mathcal{L}_{\boldsymbol{A}}\left(\boldsymbol{\xi}'\right)$ and $\boldsymbol{y} = \mathcal{L}_{\boldsymbol{A}}\left(\boldsymbol{\xi}''\right)$, i.e., $\mathcal{L}_{\boldsymbol{A}}\left(\boldsymbol{\xi}' - \boldsymbol{\xi}''\right)$ must be non-zero. Hence, $\boldsymbol{\xi}' - \boldsymbol{\xi}''$ cannot be in the kernel of $\mathcal{L}_{\boldsymbol{A}}$. Since, in the worst-case, $\boldsymbol{\xi}' - \boldsymbol{\xi}''$ is 2κ-sparse, the only way of guaranteeing this is that $\mathcal{L}_{\boldsymbol{A}}$ when restricted to any 2κ-dimensional coordinate subspace of $\mathbb{R}^n$ is a maximum rank operator. Clearly, if $\mathcal{L}_{\boldsymbol{A}} : \mathbb{R}^n \mapsto \mathbb{R}^m$ with $m < 2\kappa$, this is not possible and, whenever the worst-case scenario is hit, the sparsity prior is no longer able to guarantee signal recovery. In practice, though worst-case scenarios seldom appear, classical reconstruction algorithms fail before the limit $m = 2\kappa$ is reached.

Clearly, compression by $\mathcal{L}_{\boldsymbol{A}}$ must be coupled with a signal reconstruction stage[2] $\mathcal{R}_{\boldsymbol{A}} : \mathbb{R}^m \mapsto \mathbb{R}^n$ such that ideally $\boldsymbol{x} = \mathcal{R}_{\boldsymbol{A}}(\mathcal{L}_{\boldsymbol{A}}(\boldsymbol{x}))$. In practice the chain of the encoding and decoding step is a lossy process and $\hat{\boldsymbol{x}} = \mathcal{R}_{\boldsymbol{A}}(\mathcal{L}_{\boldsymbol{A}}(\boldsymbol{x}))$ is only an approximation of $\boldsymbol{x}$.

5.1.1 Rakeness CS

The class of linear operators $\mathcal{L}_{\boldsymbol{A}}$ that can be effectively paired with a decoder $\mathcal{R}_{\boldsymbol{A}}$ is extremely large. Most notably, if $\boldsymbol{A}$ is an instance of a matrix whose entries are independent zero-average and unit-variance Gaussian random variables, then $\mathcal{L}_{\boldsymbol{A}}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ is known to work [7, 8, 39] with very high probability.

The requirements of the CS framework are not limited to a minimum number of measurements, but also include the proper design of the rows of $\boldsymbol{A}$, that we define as *sensing sequences*. Most notably, if the elements of a generic sensing matrix row $\boldsymbol{a}$ are drawn as instances of independent and identical distributed (i.i.d.) random variables with zero-mean and unit-variance Gaussian distribution, then $\boldsymbol{\xi}$ can be recovered from $\boldsymbol{y}$ [7, 8, 39] with very high probability. Actually, literature shows that there is plenty of room for optimizing $\boldsymbol{A}$ [106, 107, 108, 109], and suitably designed matrices are able to increase compression considerably compared to naive random instances. For example, if the matrix $\boldsymbol{A}^{\pm}$ is defined as $a^{\pm}_{jk} = \mathrm{sign}(a_{jk})$, with a_{jk} the element of $\boldsymbol{A}$ at row j and column k, then $\mathcal{L}_{\boldsymbol{A}}(\boldsymbol{x}) = \boldsymbol{A}^{\pm}\boldsymbol{x}$ is also known to work with very high probability [110]. This makes the computation of $\mathcal{L}_{\boldsymbol{A}}(\boldsymbol{x})$ multiplier-less and is thus one of the best options for very low resources implementations of the encoder stage.

This agnostic and general approach can then be specialized to a suitable class of signals in many ways by adopting an adapted CS approach [111]. Among the several approaches that, thanks to adaptation, guarantee better performance with respect to agnostic CS, the state-of-the-art for the design of sensing matrices is the rakeness-based CS (Rak-CS) [108, 109, 111]

The Rak-CS approach models the sensing sequences not as instances of i.i.d. variables but as a stochastic process whose correlation matrix $\boldsymbol{\mathcal{A}} = \mathbf{E}[\boldsymbol{a}\boldsymbol{a}^\top]$ is obtained as:

$$\boldsymbol{\mathcal{A}} = \frac{1}{2}\frac{n\boldsymbol{\mathcal{X}}}{\mathrm{tr}(\boldsymbol{\mathcal{X}})} + \frac{1}{2}\boldsymbol{I}_n \tag{5.2}$$

where $\boldsymbol{I}_n$ is an $n \times n$ identity matrix, and $\boldsymbol{\mathcal{X}} = \mathbf{E}[\boldsymbol{x}\boldsymbol{x}^\top]$ is the correlation matrix of the stochastic process generating input instances.

Roughly speaking, the statistical adaptation of the sensing matrix proposed by Rak-CS is a middle ground between standard CS theory (that suggests i.i.d. based sensing, and for which it is $\boldsymbol{\mathcal{A}} = I_n$) and an over-adapted setting where $\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{X}}$. Such an approach has been proved to guarantee good performance also in case of uncommon instances [20].

[2] Terms like *decoding* or *recovery* are also used to described this stage.

5.2 Bio-signal datasets and performance indexes

All the tests we perform in this dissertation on CS encoder-decoder pairs are based on bio-signals datasets, specifically ECG and EEG signals, for the the assessment of the reconstruction performance. Due to the large number of signal instances needed, in general, to train a DNN, both in the ECG and in the EEG cases, we used a MATLAB code to generate synthetic instances of the two classes of signals.

For both classes of bio-signals, the considered sparsity basis $\boldsymbol{D}$ is a family of the orthogonal Wavelet functions [112], for which these signals exhibit sparsity. In more detail, we select the Symmlet-6 family as sparsity basis for ECG signals [20], while our choice for the EEG case is the Daubechies-4 family [113]. Here, κ is set on 16 for $n = 64$ and 24 for $n = 128$. For the EEG signals the sparse vectors $\boldsymbol{\xi}$ are with respect the basis representing the Daubechies-4 wavelet transformation where $k = \{16, 26\}$ matches $n = \{64, 128\}$.

5.2.1 Synthetic ECG

The synthetic ECG dataset is generated as described in [46] and used as in [102, 21]. The dataset employed for the performance assessment is composed of $800\,000$ windows of size $n = 256$, of which $720\,000$ are used to train the TCSSO encoder/decoder, $40\,000$ compose the validation set while the remaining $40\,000$ are used to test the performance.

White noise has been added to the signals, so that the Intrinsic Signal to Noise Ratio (ISNR) is $60\,\mathrm{dB}$.

The synthetic generator[3] of ECGs is thoroughly discussed in [46]. Signals are generated as noiseless waveforms. The noisy cases are obtained by superimposing additive white Gaussian noise whose power is such that the intrinsic SNR (ISNR) is $60\,\mathrm{dB}$.

The setup is the same detailed in [108]. The heart-beat rate is randomly set using an uniform distribution between $60\,\mathrm{beat/min}$ and $100\,\mathrm{beat/min}$. We generate chunks of $2\,\mathrm{s}$ with a $256\,\mathrm{sample/s}$ sampling frequency, that are split into windows of n subsequent samples. For both $n = 64$ and $n = 128$ cases we generate 8×10^5 input vectors $\boldsymbol{x}$ such that the corresponding total number of signal chunks are 10^5 and 2×10^5. These input vectors are randomly split between a training set and a test set where the latter contains the 20% of the total amount of vectors $\boldsymbol{x}$.

5.2.2 Synthetic EEG

The detailed description of the code to generate the synthetic EEG signal[4] can be found in [114]. The generator emulates event-related brain potentials, modeling an evoked potential as the series of a positive and a negative peak occurring at a fixed time relative to the event. The peaks are added to the uncorrelated background noise, whose power is set to a level such that the resulting signal is very similar to an EEG signal measured by a real scalp electrode.

[3] The MATLAB code is freely available for downloaded from the Physionet website at `http://physionet.org/content/ecgsyn/`

[4] The MATLAB code is freely available for download from the Medical Research Council Brain Network Dynamics Unit at the University of Oxford website at `http://data.mrc.ox.ac.uk/data-set/simulated-eeg-data-generator`

Though the software can generate all channels in a multi-electrode EEG according to the standard 10-20 system, we focus on the "Fz" electrode, since it is in proximity (but not exactly on the top) of the simulated source of the stimulus. The sampling rate is set to $1024\,\mathrm{sample/s}$ with a stimulus frequency of $1\,\mathrm{Hz}$.

We generate tracks corresponding to 50 different patients by starting from the parameters used in [114] and adding a random uniformly distributed offset to each of them. The ranges of the offsets for the positive peaks are ± 16 samples for the position of the peak, $\pm 0.05\,\mathrm{Hz}$ for the peak frequency, and ± 1 for the peak amplitude. Ranges for the random offsets for negative peaks are ± 26 samples for the position of the peak, $\pm 1\,\mathrm{Hz}$ for the peak frequency, and ± 4 for the peak amplitude.

The signal length for each patient is such that the total number of n-sample windows is 8×10^5. After that, 20% of signal instances for each patient are randomly select to contribute at the test set while the remaining 80% is for the training phase.

5.2.3 Performance indexes

The encoder-decoder chain may simultaneously perform more than one useful operation on the signal (see, e.g., [115, 116, 117, 118] for its use as an encryption stage) of which compression is surely the most obvious as $m < n$. The compression performance of the encoder-decoder chain is easily assessed by the compression ratio n/m.

To evaluate the achieved quality of service, one defines the Reconstructed Signal to Noise Ratio (also indicated simply as SNR in [79, 81]), measured in dB, as

$$\mathrm{RSNR} = 20\log_{10}\frac{\|\boldsymbol{x}\|_2}{\|\boldsymbol{x}-\hat{\boldsymbol{x}}\|_2} = \left(\frac{\|\boldsymbol{x}\|_2}{\|\boldsymbol{x}-\hat{\boldsymbol{x}}\|_2}\right)_{\mathrm{dB}} \tag{5.3}$$

where $\hat{\boldsymbol{x}}$ is the reconstructed signal.

RSNR can be used to define two ensemble-level performance figures, computed starting from a set $\boldsymbol{x}^{(t)}$ (for $t = 0, \ldots, T-1$) of signal instances recovered as $\hat{\boldsymbol{x}}^{(t)}$. The first is the Average RSNR (ARSNR)

$$\mathrm{ARSNR} = \frac{1}{T}\sum_{t=0}^{T-1}\mathrm{RSNR}^{(t)} \tag{5.4}$$

while the second is the Probability of Correct Reconstruction (PCR) that, given a $\mathrm{RSNR}_{\mathrm{min}}$ value, is defined as

$$\mathrm{PCR} = \frac{1}{T}\#\left\{t \,\middle|\, \mathrm{RSNR}^{(t)} \geq \mathrm{RSNR}_{\mathrm{min}}\right\} \tag{5.5}$$

where $\#$ counts the number of elements in the set. The value of $\mathrm{RSNR}_{\mathrm{min}}$ has to be set accordingly to the minimum RSNR level that is considered *sufficient* for a correct reconstruction.

5.3 Signal encoding and pros/cons of short windows

The possibility of optimizing and changing the size of the sensing matrix $\boldsymbol{A}$ paves the way to applications in all those settings in which the computational complexity of compression must be kept at bay, e.g., in BANs for which reduced computation and compression before

transmission are essential to fit within a tight resource budget. Given this premise, it is worth stressing that, to best express its potential in reducing computational complexity at the encoder, CS should consider the shortest possible acquisition windows. To understand why, consider the processing of N given samples. They may be partitioned into N/n contiguous and non-overlapping time windows, each with n samples. Operator $\mathcal{L}_{\boldsymbol{A}}$ can be applied to each window, entailing a number of operations $\mathcal{O}(n \cdot m)$. The total number of operations to process the N samples is $\mathcal{O}(n \cdot m \cdot N/n) = \mathcal{O}(n \cdot N/\mathrm{CR})$. However, CR is fixed to a sufficient level to reconstruct the original n-dimensional signal $\boldsymbol{x}$ from the m measurement $\boldsymbol{y}$ with a quality that is deemed acceptable. Hence, at given CR and N, the computational complexity is linearly increasing with n, i.e., with the length of individual time windows.

Another aspect that has to be considered is the signal reconstruction latency. Even considering that $\mathcal{R}_{\boldsymbol{A}}(\boldsymbol{y})$ is an instantaneous operation, the reconstructed signal is recovered with a delay of up to n time steps, since $\boldsymbol{y}$ is available with a delay of up to n time steps. Of course, the smaller the n, the lower the reconstruction latency.

Beyond these high-level reasons, short windows may benefit the implementation of the encoder also at a more physical level. In purely digital realizations [119, 120, 121], the samples come from a conventional Analog-to-Digital converter and the encoder is implemented as a sequence of sums and subtractions depending on the entries of, for example, an antipodal sensing matrix $\boldsymbol{A}^{\pm}$. In this case, not only the computation time but also the memory needed to store $\boldsymbol{A}$ reduces when n (and m) gets smaller. In mixed-mode realizations (i.e., in the design of Analog-to-Information converter based on CS) [122, 123, 81, 79, 20], $\boldsymbol{y} = \boldsymbol{A}^{\pm}\boldsymbol{x}$ is computed component-wise as $y_j = \sum_{k=0}^{n-1} a_{jk}^{\pm} x_k$, i.e., accumulating the signal samples in the analog domain. This operation implies an analog storage to hold the intermediate sum value. However, independently of the actual implementation and technology, the approach is doomed to suffer from leakage and disturbance [20, 124]. These phenomena degrade the stored value along time, and their effect increases with the hold time and the number of sums. Hence, the lower the n, the shorter the time and the smaller number of operations needed to compute y_j, and therefore the smaller the degradation incurred before conversion into digital words occurs.

Regrettably, gaining all the advantages connected with the reduction of n is not straightforward. In fact, real-world signals are such that, when n shrinks, the ratio κ/n is expected to increase. Since κ affects m, any reduction of n tends to impair the compression ratio. As a remark, the trend with which κ/n increases when n decreases is a feature of the class of signals considered.

To get a quantitative feeling of these trends, we show in Figure 5.3 the normalized sparsity κ/n for different values of n observed in the classes of ECG and EEG signals. Instances are obtained according to the synthetic generators described in Section 5.2.

The value of κ is seen as a system parameter estimated at design-time so that the representations along the sparsity basis of most of the signal instances, feature a number of non-negligible elements not larger than κ. In Figure 5.3, κ is estimated as the least number of entries of the sparse representation that includes 99.5% of the energy in the 99% of the

ECG instances, and 95% of the energy in the 99% of the EEG instances[5]. From the figure it is clear that the smaller the n, the larger the (normalized) sparsity, and therefore the lower the attainable CR that ensures a target reconstruction quality.

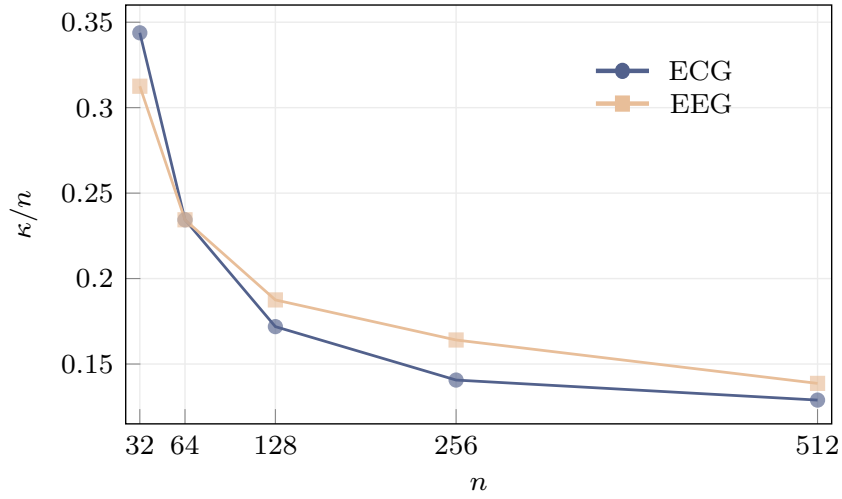


FIGURE 5.3: The effect of reducing n on the normalized sparsity κ/n in the two examples of a synthetic ECG and a synthetic EEG signal. (© 2020 IEEE)

The above considerations reveal that there is a multi-faceted trade-off linking computational/implementation complexity, reconstruction quality, and compression.

5.4 A two-step trained structure for signal reconstruction: TCSSO decoder

Given the premises of the previous sections of this chapter, we introduce now a novel DNN-based method for the reconstruction of compressed bio-signals, that we first introduced in [102].

As a starting point, to better formalize sparsity and its consequences, recall $\boldsymbol{x} = \boldsymbol{D}\boldsymbol{\xi}$ and that not more than κ entries of $\boldsymbol{\xi}$ are non-null. The positions of the non-zero entries of $\boldsymbol{\xi}$ identify the so-called *support* $\operatorname{supp}\boldsymbol{\xi}$ that we will represent by means of the binary vector $\boldsymbol{s} \in \{0,1\}^n$ such that $s_j = 1$ if $\xi_j \neq 0$ and $s_j = 0$ otherwise. Binary, n-dimensional vectors can be used to *index* a generic n-dimensional vector $\boldsymbol{v}$ so that $\boldsymbol{v}_{|\boldsymbol{s}}$ is the sub-vector of $\boldsymbol{v}$ collecting only the entries $\boldsymbol{v}_j$ such that $\boldsymbol{s}_j = 1$. We will use binary n-dimensional vectors also to *index* matrices $\boldsymbol{M}$ with n columns so that $\boldsymbol{M}_{|\boldsymbol{s}}$ is the sub-matrix of $\boldsymbol{M}$ that contains only the columns whose index j is such that $s_j = 1$. With this notation, κ-sparsity is equivalent to say that $\boldsymbol{x}$ is efficiently represented by two pieces of information, namely the n-dimensional binary vector $\boldsymbol{s}$ and the real vector $\boldsymbol{\xi}_{|\boldsymbol{s}}$ whose dimensionality does not exceed κ.

Sparsity is fundamental in the decoding process going from $\boldsymbol{y}$ back to $\boldsymbol{x}$. In fact, since $m < n$, the mapping $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{D}\boldsymbol{\xi}$ from $\boldsymbol{\xi}$ to $\boldsymbol{y}$ is non-injective. Hence, any given measurement

[5]For the class of EEG signals we refer to a synthetic signal that emulates event-related brain potentials, where readings in each lead contain information on the external stimulus as well as a part on other neurons activity. This is why we assume that 95% of the total energy is enough to identify the meaningful components of the signal.

vector $\boldsymbol{y}$ corresponds to an infinite number of possible $\boldsymbol{\xi}$. However, if $\boldsymbol{A}$ is properly designed, only one of the counter-images of $\boldsymbol{y}$ is κ-sparse and can be found by relatively simple algorithmic means.

As shown in Figure 5.2, a decoder recovers both $\boldsymbol{s}$ and $\boldsymbol{\xi}_s$. Among the many methods proposed in the literature, the most classical approach is BPDN which recovers both pieces of information simultaneously by solving the optimization problem

$$\hat{\boldsymbol{\xi}} = \arg\min_{\boldsymbol{\xi}\in\mathbb{R}^n} \|\boldsymbol{\xi}\|_1 \quad \text{s.t.} \quad \|\boldsymbol{y} - \boldsymbol{AD}\boldsymbol{\xi}\|_2 \leq \tau \tag{5.6}$$

where $\hat{\boldsymbol{x}} = \boldsymbol{D}\hat{\xi}$ is the reconstructed signal, $\|\boldsymbol{v}\|_p$ indicates the p-norm of the generic vector $\boldsymbol{v}$, and $\tau \geq 0$ accounts for the possible presence of disturbances in the computation of $\boldsymbol{y}$ by relaxing the constraint $\boldsymbol{y} = \boldsymbol{AD}\boldsymbol{\xi}$ that would hold in the noiseless case. The noiseless case itself corresponding to solve the simpler BP problem can, of course, be tackled by setting $\tau = 0$.

Though implicitly performed, support identification is an essential ingredient in BP and BPDN and is embedded in the 1-norm used in the objective function. The reason behind the use of the 1-norm minimization is to replace the minimization of the cardinality of the support of $\boldsymbol{\xi}$ that would yield a combinatorial problem. In fact, 1-norm minimization tends to select the ξ with the least number of non-zero entries among all the possible $\boldsymbol{\xi}$ satisfying the constraint [7]. This property is so critical that changing the 1-norm in the merit function would completely spoil reconstruction while changing the 2-norm in the constraint usually still gives sensible results. Note that, despite its fundamental merit, the 1-norm minimization is only a *proxy* of support identification, which works under suitable assumptions that are not necessarily satisfied in practice, especially for large κ/n values [7].

Since we enlarge the application of the CS framework to the cases where κ/n is quite large, we here consider a different approach composed of two consecutive blocks as we proposed in [102]. The first block, indicated in [102] as *support oracle* (SO), is devoted to identify the support and is capable to divine $\boldsymbol{s}$ by looking at the vector $\boldsymbol{y}$. Then, assuming $\boldsymbol{s}$ is known, by defining $\boldsymbol{B} = \boldsymbol{AD}$, we can observe that

$$\boldsymbol{y} = \boldsymbol{A}(\boldsymbol{D}\boldsymbol{\xi} + \boldsymbol{\nu}) = \boldsymbol{B}\boldsymbol{\xi} + \boldsymbol{A}\boldsymbol{\nu} = \boldsymbol{B}_{|\boldsymbol{s}}\boldsymbol{\xi}_{|\boldsymbol{s}} + \boldsymbol{A}\boldsymbol{\nu}. \tag{5.7}$$

The fact that $\kappa < m$ makes $\boldsymbol{B}_{|\boldsymbol{s}}$ a *tall* matrix (the number of rows exceeds the number of columns) such that each measurement vector $\boldsymbol{y}$, ignoring the noise $\boldsymbol{\nu}$, possesses a unique counterimage given by $\boldsymbol{\xi}_{|\boldsymbol{s}} = \boldsymbol{B}^{\dagger}_{|\boldsymbol{s}}\boldsymbol{y}$, where $\cdot^{\dagger}$ indicates the Moore-Penrose pseudoinverse operation.

In other words, given (5.7), to recover the input signal, it is enough a second stage computing

$$\hat{\boldsymbol{\xi}}_{|\boldsymbol{s}} = \boldsymbol{B}^{\dagger}_{|\boldsymbol{s}}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\nu}) = \boldsymbol{B}^{\dagger}_{|\boldsymbol{s}}\boldsymbol{y} - \boldsymbol{B}^{\dagger}_{|\boldsymbol{s}}\boldsymbol{A}\boldsymbol{\nu} \tag{5.8}$$

that is a much simpler operation with respect to any CS recovery algorithm. The term $\boldsymbol{e} = \boldsymbol{B}^{\dagger}_{|\boldsymbol{s}}\boldsymbol{A}\boldsymbol{\nu}$ defines the signal recovery error in the sparse representation.

The advantages introduced by the knowledge of the support vector $\boldsymbol{s}$ in the solving the recovery problem are graphically schematized in Figure 5.4.

Since the aim is always the computation of the non-null entries of $\boldsymbol{\xi}$, if $\boldsymbol{s}$ is unknown, the signal recovery is performed by inverting a wide matrix (which is an ill-defined problem) thus obtaining both null and non-null entries of $\boldsymbol{\xi}$. Otherwise, assuming that an oracle divining $\boldsymbol{s}$ exists, the recovery problem only focuses on the computation of the non-null entries of $\boldsymbol{\xi}$ such that the recovery stage only performs the (pseudo-)inversion of a tail matrix.

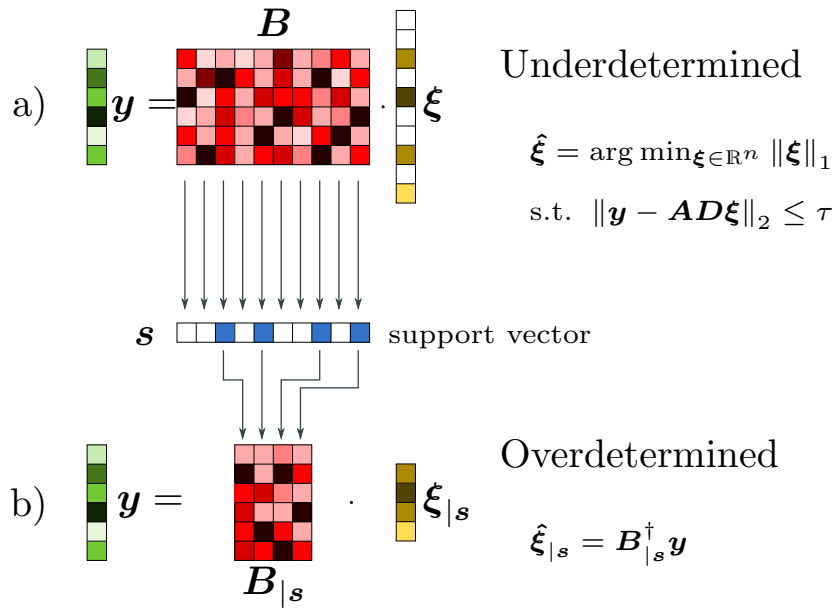


FIGURE 5.4: The original ill-defined underdetermined problem (a) can be transformed in a overdetermined problem (b) by using the support vector $\boldsymbol{s}$. (© 2020 IEEE)

5.4.1 Oracle training and decoder operations

The support oracle we propose is based on a DNN trained on signals with the same statistical features of the one to be acquired. The DNN $\mathcal{N}_C : \mathbb{R}^m \mapsto [0,1]^n$ is defined by the connection parameters in C, with m inputs that correspond to the m entries of the measurement vector y and n outputs.

The neural network has 3 intermediate fully connected layers of cardinality $2n$, $2n$, and n, all with a ReLu activation function. The output layer is also fully connected with n units and sigmoidal activation function that map any scalar a into $(1+e^{-a})^{-1}$. Training also adapts the matrix $\boldsymbol{A}$, so that encoder and decoder are jointly optimized to improve support identification and thus to improve reconstruction performance.

Both the connection parameters C and the matrix $\boldsymbol{A}$ are initialized as instances of independent zero-average unit-variance Gaussian random variable and adjusted by training the compound system $\mathcal{N}_C \circ \mathcal{L}_A : \mathbb{R}^n \mapsto [0,1]^n$. The training set is made of a sequence of κ-sparse signals $\boldsymbol{x}^{(t)} = \boldsymbol{D}\boldsymbol{\xi}^{(t)}$ (for $t = 0, \ldots, T-1$) and of corresponding binary vectors $\boldsymbol{s}^{(t)}$. The true support of $\boldsymbol{\xi}^{(t)}$ encoded in $\boldsymbol{s}^{(t)}$ and the output $\boldsymbol{o}^{(t)} = \mathcal{N}_C\left(\mathcal{L}_{\boldsymbol{A}}\left(\boldsymbol{x}^{(t)}\right)\right)$ of the DNN are

compared by means of a loss function, which is a total component-wise clipped cross-entropy between $\boldsymbol{s}$ and $\boldsymbol{o}$ defined by

$$X^{(t)} = -\sum_{j|\boldsymbol{s}_j^{(t)}=1} \mathrm{L}_\epsilon\left(\boldsymbol{o}_j^{(t)}\right) - \sum_{j|\boldsymbol{s}_j^{(t)}=0} \mathrm{L}_\epsilon\left(1-\boldsymbol{o}_j^{(t)}\right) \tag{5.9}$$

where $\mathrm{L}_\epsilon(\cdot) = \min\{\log_2(1-\epsilon), \max\{\log_2(\epsilon), \log_2(\cdot)\}\}$ for a small ϵ. Though $\mathcal{L}_{\boldsymbol{A}}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ in the forward pass, to prevent the sign function from interrupting error backpropagation, in the backward pass we assume $\nabla_{\boldsymbol{A}}\mathcal{L}_{\boldsymbol{A}}(\boldsymbol{x}) = \nabla_{\boldsymbol{A}}(\boldsymbol{A}\boldsymbol{x})$. With this, since $\boldsymbol{A}_{jk} = \mathrm{sign}(\boldsymbol{A}_{jk})$ for every j and k, the training acts on the continuous-valued parameters whose sign is used in feedforward computation.

As defined in Section 5.2, we generate a dataset composed of 8×10^5 signal instances for both the ECG and the EEG case. Each dataset is split in 80% for training (training set) and 20% for performance assessment (validation set).

Each window has been pre-sparsified, meaning that κ-sparsity has been enforced for each window by removing the $n-\kappa$ smallest entries of $\boldsymbol{\xi}$. This has been done to ensure that the conditions for CS are met, while a further insight on the signal sparsity problem and how to treat realistic signals can be found in the next sections as in [21]. For each signal, the true support vector $\boldsymbol{s}$ is retrieved based on the non-zero elements of $\boldsymbol{\xi}$ and used as training label.

All models proposed in this paper are implemented and trained using the TensorFlow framework [125] with the help of the high-level API provided by Keras [126]. Training is performed with stochastic gradient descent, where each gradient step is computed with a mini-batch comprising of 30 signal instances and an initial learning rate value of 0.1.

To appreciate the complexity of the networks we propose, the one for $n = 64$ and with m ranging in $[16, 40]$ contains from 32 128 to 36 736 parameters and in our examples is trained for 500 epochs[6]. For $n = 128$ and with m ranging in $[24, 64]$, the model counts from 124 672 to 140 032 parameters and in our examples is trained for 1000 epochs. Even assuming that each parameter is encoded in 4B, the total memory footprint is limited below 150KiB for $n = 64$ and below 550KiB for $n = 128$. Such requirements may easily fit within the memory budget of commercially available devices used for small scale computation and gateway tasks.

The trained oracle can be exploited in the definition of the decoder reported in Figure 5.5. We compute $\boldsymbol{o} = \mathcal{N}_C(\boldsymbol{y})$ and, given a certain threshold $\boldsymbol{o}_{\min} \in [0, 1]$, we estimate $\boldsymbol{s}$ with the binary vector $\hat{\boldsymbol{s}} \in \{0, 1\}^n$ such that $\hat{s}_j = 1$ if $\boldsymbol{o}_j \geq \boldsymbol{o}_{\min}$ and $\hat{\boldsymbol{s}}_j = 0$ otherwise. Starting from $\hat{\boldsymbol{s}}$ we finally estimate as in (5.8), neglecting the noise term, the reconstructed signal

$$\hat{\boldsymbol{\xi}}_{|\hat{\boldsymbol{s}}} = \left(\boldsymbol{A}\boldsymbol{D}_{|\hat{\boldsymbol{s}}}\right)^\dagger \boldsymbol{y} \tag{5.10}$$

The two estimations $\hat{\boldsymbol{s}}$ and $\hat{\boldsymbol{\xi}}_{|\hat{\boldsymbol{s}}}$ define the recovered signal $\hat{\boldsymbol{x}}$. Decoder operations depend on the value of $\boldsymbol{o}_{\min}$ that is set by a further training phase in which each vector in the training set is encoded and decoded for different values of $\boldsymbol{o}_{\min}$. The $\boldsymbol{o}_{\min}$ yielding the highest ARSNR is selected. We name our approach Trained CS with Support Oracle (TCSSO) to summarize its main features.

[6] In each epoch the training algorithm walks through the entire training set.

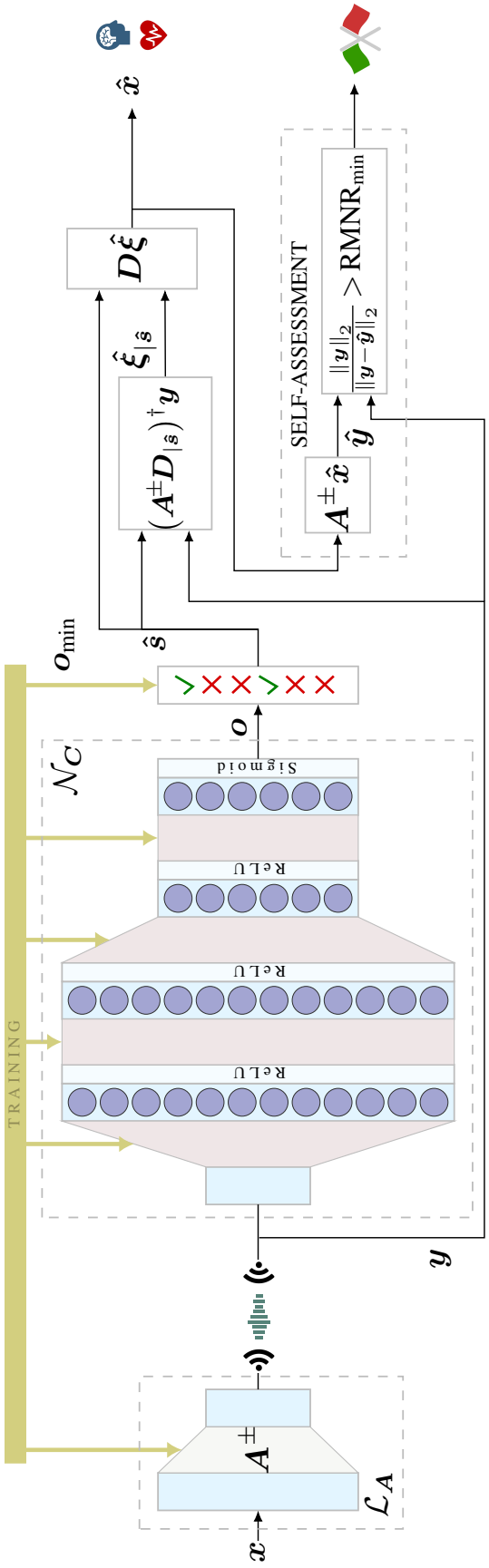


FIGURE 5.5: Trained CS with support oracle block scheme including self-assessment capability. The fully connected DNN in the left is $\mathcal{N}_C \circ \mathcal{L}_A$ with an output layer $\boldsymbol{o} \in [0,1]^n$. Estimated support $\hat{s}$ is such that $\hat{\boldsymbol{s}}_j = 1$ if $\boldsymbol{o}_j \geq \boldsymbol{o}_{\min}$ and $\hat{\boldsymbol{s}}_j = 0$ otherwise. The estimated support is employed in the signal reconstruction where the reconstructed signal $\hat{\boldsymbol{x}}$ is also the input of the self-assessment block. (© 2020 IEEE)

5.4.2 Reconstruction performance

We compare the performance of TCSSO with that of some well-known methods. Both for TCSSO and the other methods, we use antipodal sensing matrices $\boldsymbol{A}^{\pm} \in \{-1, 1\}^{m \times n}$. Since TCSSO simultaneously adapts encoder and decoder, we pair some classical signal recovery algorithms with rakeness technique for the optimization of the matrix $\boldsymbol{A}^{\pm}$ that is able to cope with the antipodality constraint on the entries.

As decoders, we consider BP and BPDN as presented in (5.6) along with Orthogonal Matching Pursuit (OMP) [87] and Generalized Approximate Message Passing (GAMP) [86]. OMP is a lightweight greedy approach that iteratively estimates the signal support while GAMP is often better than BP and BPDN as it exploits Gaussian approximation of BP that usually holds for large n values. When dealing with ECGs, we also test the performance of the Weighted ℓ_1 minimization (WL1) [91] as a representative of decoders that exploit statistical priors on the signal support. In all the tested cases, BP outperforms BPDN such that, in the rest of the paper, we consider BP as a reference for standard CS decoder.

We evaluate ARSNR and PCR by Montecarlo simulations using the samples of the validation set for both ECG and EEG cases with a superimposed noise resulting in an Intrinsic Signal-to-Noise Ratio (ISNR) equal to $60\,\mathrm{dB}$. The achieved performances are reported in Figure 5.6 for the $n = 64$ and $\kappa = 16$ case. In all plots, the number of measurements sweeps from $m = 40$ down to $m = 16$ thus focusing on compression ratios from CR $= 1.6$ up to CR $= 4$.

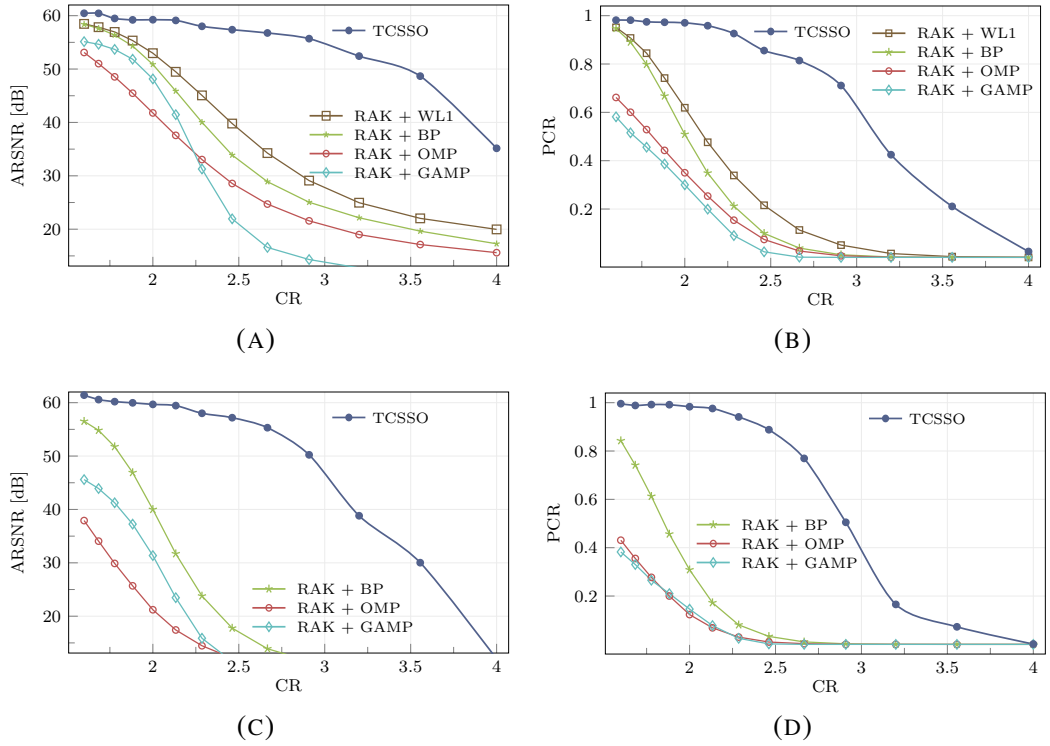


FIGURE 5.6: Reconstruction performance for ECG (a)-(b) and EEG signals (c)-(d) in terms of both ARSNR (a)-(c) and PCR (b)-(d) with $\text{RSNR}_{\text{min}} = 55\,\text{dB}$. The support oracle decoder with trained sensing matrices, TCSSO, is compared against OMP, BP, and GAMP with adapted sensing matrices (rakeness-based CS, RAK). For the ECG, we also consider the adapted decoder WL1 in [91].

TCSSO outperforms all other techniques and allows us to work at compression ratios much larger than those commonly achievable, while still requiring a limited computational effort since $n = 64$. For example, to guarantee ARSNR $= 50\,\text{dB}$, results in Figure 5.6(a),(c) show that by using TCSSO one may get CR ≈ 3.5 for ECGs and ≈ 2.9 for EEGs. In the same setting, RAK+WL1 is the best performing competitor for ECGs with CR ≈ 2.2 while RAK+BP is to be considered a benchmark for the EEGs with CR ≈ 1.8.

Figure 5.7 shows how the situation changes when n increases from 64 to 128. Performance is reported only in terms of ARSNR and for TCSSO along with its best competitor. The increase of n positively impacts performances in general since κ/n decreases. Nevertheless, TCSSO still outperforms the best of the traditional CS frameworks. Considering ARSNR $=$ $50\,\text{dB}$ as the desired quality of service, TCSSO works with CR ≈ 4.4 and CR ≈ 2.9 while the competitors give at most CR ≈ 2.7 and CR ≈ 2.2 in case of ECGs and EEGs respectively.

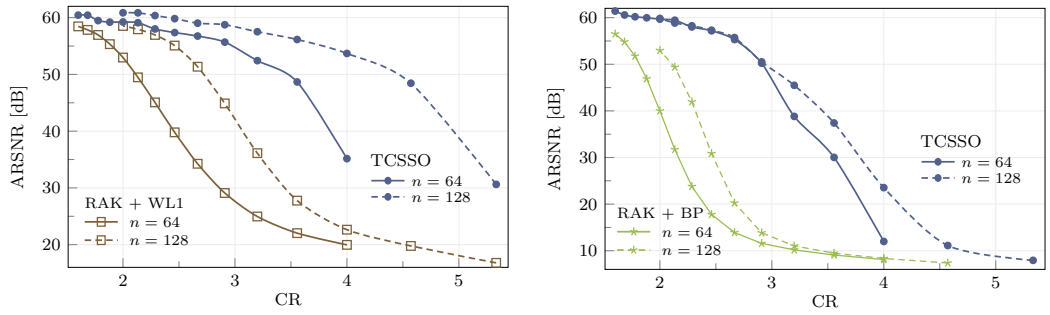


FIGURE 5.7: Performance in terms of ARSNR ECG (a) and EEG signals (b) as a function of the compression ratio for both TCSSO and the best observed traditional approach (RAK + WL1 for ECG while the EEG case refers to RAK + BP). Results are for $n = 64$ as well as for $n = 128$. (© 2020 IEEE)

5.4.3 Preliminary evidence on real signals

The ideal path for applying our method in real-world cases is to collect enough acquisitions to substantiate both a training and a validation set. The acquisitions to which we have access do not currently allow such a thorough assessment. However, some evidence can be given by using synthetic data for the learning phase, and real-world signals for a preliminary assessment. Such an approach is suboptimal since there is no guaranteed coherence between the training set used for learning and the validation set used for assessment. The results are still encouraging.

In particular, we may consider the waveforms contained in the MIT database for testing compression of ECG signals [127, 128] and the pool of acquisitions used in [113] for EEG. A sample comparison between original and reconstructed waveforms for $n = 64$ and $m = 32$ (which is equal to 2κ) is reported in Figure 5.8. Despite the appearance of some artifacts introduced by the decoder, the plots show that, even with the suboptimal setting, our method is able to yield acceptable reconstructions with extremely small n and with m below the classical threshold 2κ.

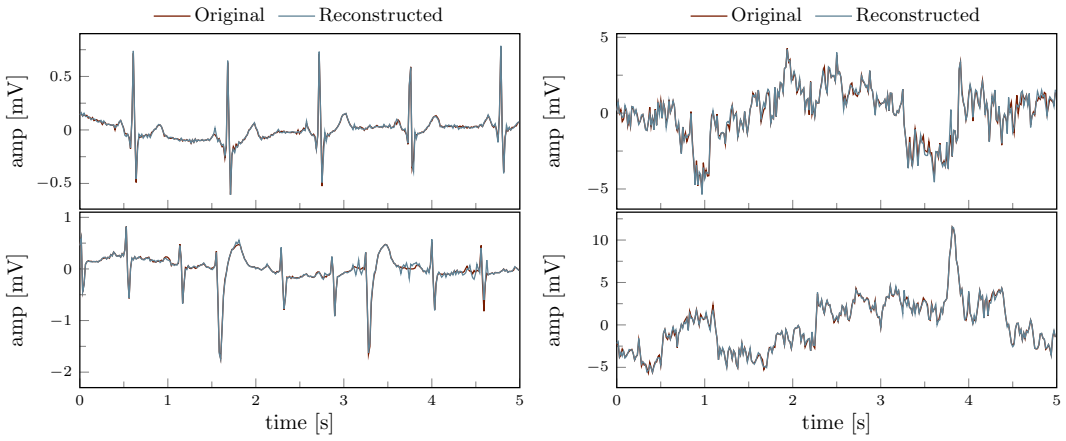


FIGURE 5.8: Comparisons between real-world signals and their corresponding waveforms reconstructed by TCSSO for ECGs (a) and EEGs (b). The top plot (a) refers to records 11950_03 and 12531_03 taken from the online repository MIT-BIH ECG compression test database [129]. The bottom plot (b) contains EEG signal chunks of the "Fz" electrode from the pool of acquisition described in [113]. For both cases we adopt $n = 64$ and $m = 32$. (© 2020 IEEE)

5.5 Decoder self-assessment

The TCSSO architecture described in the previous section can be extended by exploiting a property that stems from the fact that $\boldsymbol{s}$ is estimated separately from $\boldsymbol{\xi}_{|\boldsymbol{s}}$.

In fact, assume that no noise is present and that the size and content of $\boldsymbol{A}^{\pm}$ are such that $\boldsymbol{y} = \boldsymbol{A}^{\pm}\boldsymbol{D}\xi$ is satisfied by one and only one κ-sparse $\boldsymbol{\xi}$, i.e., that recovery of the true signal is theoretically possible. If the oracle is successful in divining the support, then $\hat{\boldsymbol{s}} = \boldsymbol{s}$ and $\boldsymbol{y} = \boldsymbol{A}^{\pm}\boldsymbol{D}_{|\boldsymbol{s}}\boldsymbol{\xi}_{|\boldsymbol{s}}$ implies that $\boldsymbol{y} \in \text{span}\left(\boldsymbol{A}^{\pm}\boldsymbol{D}_{|\hat{\boldsymbol{s}}}\right)$, where, for any matrix $\boldsymbol{M}$, $\text{span}\left(\boldsymbol{M}\right)$ is the subspace generated by the linear combination of its columns. This has a twofold consequence: *i*) (5.10) computes $\hat{\boldsymbol{\xi}}_{|\hat{\boldsymbol{s}}} = \boldsymbol{\xi}_{|\boldsymbol{s}}$, *ii*) if $\hat{\boldsymbol{\xi}}$ is mapped back we have $\boldsymbol{A}^{\pm}\boldsymbol{D}\hat{\boldsymbol{\xi}} = \boldsymbol{y}$.

However, if the oracle fails, then $\hat{\boldsymbol{s}} \neq \boldsymbol{s}$ and since $\boldsymbol{\xi}$ is the unique κ-sparse solution of $\boldsymbol{y} = \boldsymbol{A}^{\pm}\boldsymbol{D}\boldsymbol{\xi}$ then $\boldsymbol{y} \notin \text{span}\left(\boldsymbol{A}^{\pm}\boldsymbol{D}_{|\hat{\boldsymbol{s}}}\right)$. This has a twofold consequence: i) (5.10) computes $\hat{\boldsymbol{\xi}}_{|\hat{\boldsymbol{s}}} \neq \boldsymbol{\xi}_{|\boldsymbol{s}}$, ii) if $\hat{\boldsymbol{\xi}}$ is mapped back we have $\boldsymbol{A}^{\pm}\boldsymbol{D}\hat{\boldsymbol{\xi}} \neq \boldsymbol{y}$.

Clearly, the decoder cannot check the correctness of $\hat{\boldsymbol{\xi}}$ as the true $\boldsymbol{\xi}$ is unknown. Nevertheless, it may map $\hat{\boldsymbol{\xi}}$ back to measurement obtaining $\hat{\boldsymbol{y}} = \boldsymbol{A}^{\pm}\boldsymbol{D}_{|\hat{\boldsymbol{s}}}\left(\boldsymbol{A}^{\pm}\boldsymbol{D}_{|\hat{\boldsymbol{s}}}\right)^{\dagger}\boldsymbol{y} = \boldsymbol{A}^{\pm}\hat{\boldsymbol{x}}$ that could be different from $\boldsymbol{y}$. As a result, $\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2$ is most naturally linked to the decoder failure and grants a useful self-assessment capability. In particular, one may monitor the quantity

$$\text{RMNR} = \frac{\|\boldsymbol{y}\|_2}{\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2} \tag{5.11}$$

that is the Reconstruction Measurements-to-Noise Ratio, and declare that the oracle, and thus the TCSSO decoder, has succeeded when $\text{RMNR} \geq \text{RMNR}_{\text{min}}$ for a certain threshold.

This situation can be exemplified in the small-dimensional case $n = 4$, $\kappa = 2$ and $m = 3$ with

$$\boldsymbol{A}^{\pm} = \begin{pmatrix} +1 & +1 & +1 & +1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & +1 & -1 \end{pmatrix}$$

Since $\kappa = 2$, the instances of the original signal $\boldsymbol{\xi} \in \mathbb{R}^4$ may have at most two non-null components and thus lay on the union of all the possible coordinate planes in $\mathbb{R}^4$. We may indicate one of those planes as c_{jk} where j and k are the indexes of the non-null coordinates of its points. The matrix $\boldsymbol{A}^{\pm}$ is such that $\boldsymbol{A}^{\pm}$ maps each of those 6 coordinate planes into a plane in $\mathbb{R}^3$ that can be distinguished from the others. This is exemplified in Figure 5.9 on the left of which we draw the 6 planes $\iota_{jk} \subset \mathbb{R}^3$ that are the images through $\boldsymbol{A}^{\pm}\boldsymbol{D}$ of the coordinate planes $c_{jk} \subset \mathbb{R}^4$. Note that, due to dimensionality reduction, images are not pairwise orthogonal. However, recovery is theoretically possible as no two images ι_{jk} and $\iota_{j'k'}$ are the same. Therefore a sufficiently clever algorithm can establish the support by looking at the measurement vector $\boldsymbol{y}$.

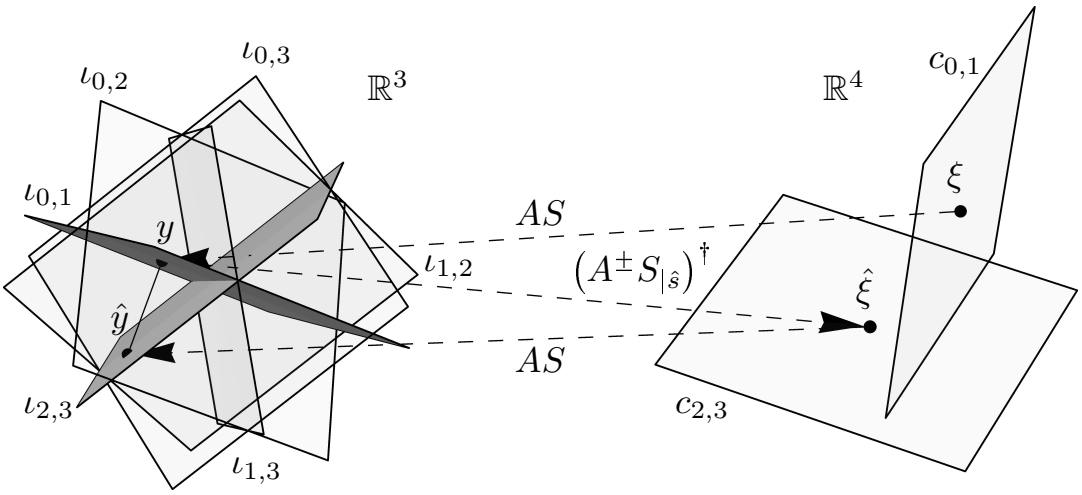


FIGURE 5.9: The mechanism granting self-assessment capabilities to decoders based on a support oracle. (© 2020 IEEE)

Assume now that $s = (1, 1, 0, 0)$, i.e., that the true signal $\boldsymbol{\xi} \in c_{0,1}$ is mapped by $\boldsymbol{A}^{\pm}\boldsymbol{D}$ into a measurement vector $\boldsymbol{y} \in \iota_{0,1}$. Assume also that the oracle mistakes the support and estimates $\hat{\boldsymbol{s}} = (0, 0, 1, 1)$, implying $\hat{\boldsymbol{\xi}} \in c_{2,3}$. By computing (5.10), the vector $\boldsymbol{y}$ is mapped back to $\hat{\boldsymbol{\xi}}$ on that plane, which is therefore different from $\boldsymbol{\xi}$. Though, only approximately, the same holds in the noisy case and give an idea why the difference between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ assesses the correctness of the divined $\hat{\boldsymbol{s}}$, i.e., the quality of the reconstruction $\hat{\boldsymbol{x}}$.

As an example of the underlying mechanism, Figure 5.10 reports some Montecarlo evidence on the relationship between RMNR and RSNR for the ECG signals and in three different configurations. In Figure 5.10a no noise is present and $m = 32 = 2\kappa$; in Figure 5.10b ISNR $= 60\,\mathrm{dB}$ and $m = 32 = 2\kappa$, whereas in Figure 5.10c no noise is present, but $m = 24 < 2\kappa$.

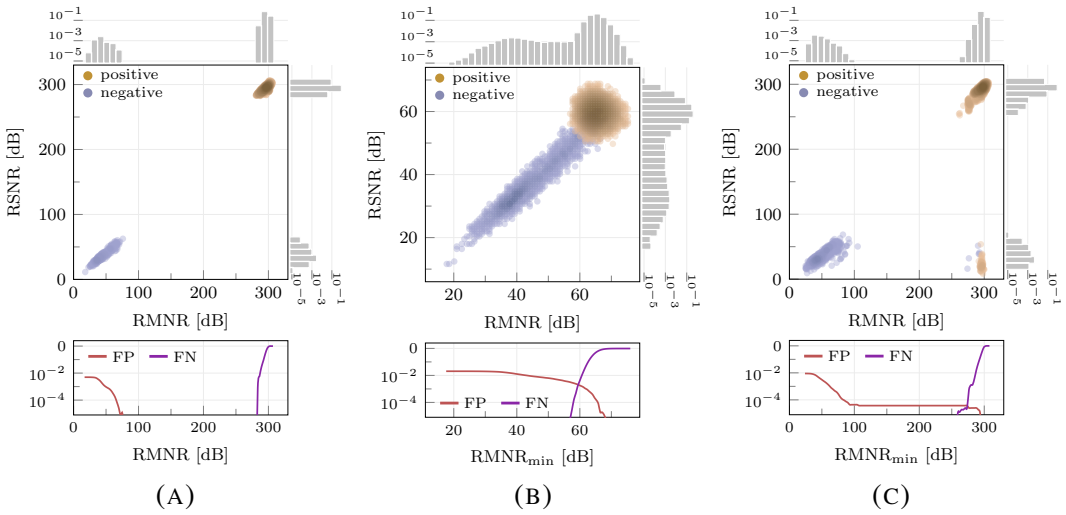


FIGURE 5.10: The relationship between RSNR and RMNR for ECG signals. In all cases $n = 64$ and $\kappa = 16$. Orange dots correspond to cases in which $\hat{\boldsymbol{s}}$ includes all the components of $\boldsymbol{s}$, while blue dots correspond to $\hat{\boldsymbol{s}}$ failing to identify some components in $\boldsymbol{s}$. Above and to the right of the scatter plots are logarithmic histograms estimating the probability density of RMNR and RSNR. In (a) $m = 32 = 2\kappa$ and ISNR $= \infty$. In (b) $m = 32 = 2\kappa$ and ISNR $= 60\,\text{dB}$. In (c), $m = 24 < 2\kappa$ and ISNR $= \infty$. (© 2020 IEEE)

The two-dimensional plots show an estimation of the joint-probability, conditioned to the *positive* events, i.e., the support has been correctly identified ($\hat{s}_j \geq s_j$ for all $j = 0, \ldots, n-1$, orange points) or to the *negative* events, i.e., at least one entry in the support is neglected ($\hat{s}_j < s_j$ for at least one $j = 0, \ldots, n-1$, blue points). Darker colors stand for higher densities.

The one-dimensional plots at the bottom of the figure report the error probabilities of a self-assessment procedure that calls for a positive event whenever $\text{RMNR} \geq \text{RMNR}_{\min}$ and for a negative event otherwise. As the threshold $\text{RMNR}_{\min}$ increases, the probability of a false positive decreases since only very high RMNR reconstructions are declared correct. On the contrary, the probability of a false negative increases since for larger $\text{RMNR}_{\min}$ even good reconstructions can be declared incorrect.

The ideal conditions in Figure 5.10a result in perfect self-assessment capabilities. When noise is added as in Figure 5.10b, positive and negative cases get mixed but remain identifiable by looking at RMNR.

Though no noise is present in Figure 5.10c, the fact that $m < 2\kappa$ makes the number of measurements insufficient for signal reconstruction, as there is no guarantee that only one κ-sparse signal $\boldsymbol{\xi}$ corresponds to the given $\boldsymbol{y}$ through $\boldsymbol{A}^{\pm}\boldsymbol{D}$. Hence, more than one support corresponding to the measurement exists. In these conditions, it may happen that the oracle divines a support that includes the true one (more than κ outputs of the network are larger than $\boldsymbol{o}_{\min}$) as well as components of other possible supports. In this case, the oracle is not missing the support (orange point in the lower-right cluster in the scatter plot of Figure 5.10c). However, pseudo-inversion spreads the reconstruction over all the available components, thus

failing to reconstruct the signal. It may also happen that the oracle divines a support different from the true one. In this case, the oracle is wrong (blue points in the lower-right cluster in the scatter plot of Figure 5.10c), and pseudo inversion identifies a sparse signal that is not the true one. Both cases give rise to points for which RMNR is very high, but the RSNR is very low, and no matter how high the $\mathrm{RMNR}_{\min}$, the probability of a false positive is not vanishing.

Luckily enough, the above cases are the ones breaking *worst-case* guarantees and happen quite rarely: in our 1.6×10^5 validation set, for $n = 32$, $\kappa = 16$ and $m = 24$, the oracle divines a support in excess of the true one only 109 times, and a support different from the true one only 6 times. The statistics commonly used to assess performance remain substantially unaltered by these failures that are undetectable by looking at the RMNR.

In general, the value of $\mathrm{RMNR}_{\min}$ can be decided once that $\boldsymbol{o}_{\min}$ is set, by a further pass over the training set. This allows us to estimate false positive and false negative curves as in Figure 5.10, and use them as criteria. In the following, we will set $\mathrm{RMNR}_{\min}$ as the largest value for which false negative probability is negligible. Whenever a failure is detected, the decoder may take different actions whose effectiveness depends on the final applications.

The exploration of all the possibilities of the resulting two-level decoder is out of the scope of this paper. However, it can be easily recognized that quite a few options are available, such that:

i) raising a warning and mark the current window as potentially incorrect;

ii) feeding the warning back to the encoder and require further information to correct the reconstruction (thus lowering the CR for this instance);

iii) triggering another decoder on the same measurement vector hoping that this will improve reconstruction;

iv) any combination of the above.

As a partial and non-optimized example whose only aim is to show that some information can still be extracted from the measurements when first-attempt TCSSO decoder fails, we trigger GAMP[7] as a second-wind decoder.

Figure 5.11 plots the probability that GAMP yields a RSNR larger than what is given by TCSSO when applied to the instances that the latter marks as incorrectly recovered as $\mathrm{RMNR} < \mathrm{RMNR}_{\min}$, as a function of CR for the $n = 64$, $\kappa = 16$ case. A second-wind decoding is useful when such a probability is larger than 50%, i.e., approximately for $\mathrm{CR} \leq 2$.

[7] GAMP has achieved better results compared to the other classical reconstruction algorithms in this setting, i.e., when the sensing matrix is not designed according to the rakeness-based CS.

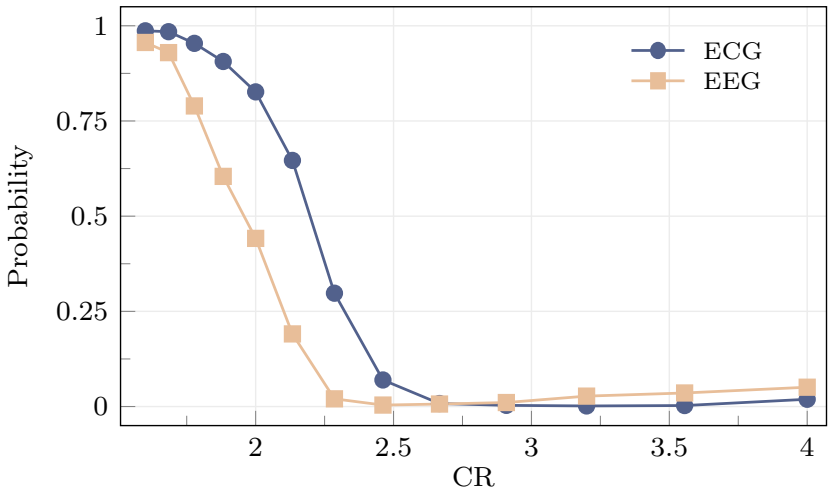


FIGURE 5.11: Probability for GAMP decoder to reconstruct both ECG and EEG signals with RSNR higher than TCSSO in case of TCSSO failure. $\mathrm{Prob}\left(\mathrm{RSNR}_{\mathrm{GAMP}} > \mathrm{RSNR}_{\mathrm{TCSSO}} \,|\, \mathrm{RMNR} < \mathrm{RMNR}_{\mathrm{min}}\right)$. (© 2020 IEEE)

5.6 Computational requirements

As noted previously, CS-based lossy compression methods result in a multi-faceted trade-off between compression ratio, reconstruction quality, and computational complexity. In this section, we give further detail on the last aspect, distinguishing what is required at the encoder (that we want to minimize) and at the decoder (that we want to be not worse than the needs of classical recovery methods). In all cases, we refer to the computational burden *per processed sample*, i.e., we divide the number of operations by the number of samples n contained in the processed window.

5.6.1 Encoder

The complexity of the encoder is briefly introduced in Section 5.3 as one of the leading design criteria. The number of signed accumulations (AC) is $nm = n^2\mathrm{CR}^{-1}$ thus yielding $n\mathrm{CR}^{-1}$ $\mathrm{AC/sample}$. Further to time-complexity, the memory footprint is dominated by the storage of the matrix $\boldsymbol{A}^{\pm}$, and requires a number of entries equal to $nm = n^2\mathrm{CR}^{-1}$. In principle, matrix entries are bits. However, microcontroller-based implementations may favor 1-byte-per-entry or even 4-bytes-per-entry solutions. In fact, in some architectures, the alignment of entries at word boundaries ensures better performance both in terms of speed and energy (see, e.g., [92]), this is why we express the memory footprint as the number of entries in $\boldsymbol{A}^{\pm}$.

From the blue curves in Figure 5.7, one gets that a higher n results in better reconstruction performance for the same CR, and thus there is a trade-off between encoder complexity and window length.

Table 5.1 reports the comparison between the increase of reconstruction quality, complexity, and memory footprint for ECG signals when n goes from 64 to 128, with CR ranging from 2 to 4. At high CR levels, an increase in terms of ARSNR (e.g., with $\mathrm{CR} = 4$, +18.0 dB for ECG and +11.5 dB for EEG) may be worth the $\times 2$ in terms of computational effort and

TABLE 5.1: Performance improvement in terms of ARSNR, computational overhead in terms of $\mathrm{AC/sample}$ and increase in memory footprint for the sensing matrix (# entries of $A^{\pm}$) with n going from 64 to 128. Results are for TCSSO in ECG and EEG cases with compression ratio (CR) ranging from 2 to 4.

CR	ARSNR [dB]				$\mathrm{AC/sample}$		# $\boldsymbol{A}^{\pm}_{jk}$	
	ECG		EEG					
n	64	128	64	128	64	128	64	128
4.00	35.2	+18.0	12.0	+11.5	16	×2	1.0 Ki	×4
3.20	52.4	+5.1	38.8	+6.7	20	×2	1.3 Ki	×4
2.67	56.8	+2.2	55.3	+0.4	24	×2	1.5 Ki	×4
2.29	58.0	+2.4	58.0	+0.2	28	×2	1.8 Ki	×4
2.00	59.2	+1.7	59.7	+0.1	32	×2	2.0 Ki	×4

the $\times 4$ in terms of memory footprint. However, for lower compression ratios, the increase in resource needs is not justified by the limited increase in performance: for $\mathrm{CR} = 2$, memory footprint and complexity increase as before but one only gains +1.7 dB in the ECG case and +0.1 dB in the EEG case.

5.6.2 Decoder

In CS-based schemes, decoding is computationally more intensive than encoding. We may evaluate the complexity of the TCSSO decoder by counting the number of Multiply and Accumulate (MAC) operations needed to compute $\hat{x}$, disregarding the training phase, starting from the fact that the number of MAC operations required in a fully connected layer with n nodes, each with i inputs, is ni.

Neglecting the input layer, that has m nodes and that requires no operations, the oracle $\mathcal{N}_C$ is composed by 3 fully connected hidden layers with $2n$, $2n$ and n nodes, and a final fully connected output layer with n nodes. The number of inputs of these layers is therefore m, $2n$, $2n$ and n, respectively. The layer-by-layer number of MACs required for the forward pass is $2nm$, $4n^2$, $2n^2$ and n^2, giving rise to a total of $(2m + 7n)n = (2/\mathrm{CR} + 7)n^2$ MACs for each window thus yielding $(2/\mathrm{CR} + 7)n\ \mathrm{MAC/sample}$.

After support estimation, additional MACs are needed to compute $\hat{\boldsymbol{x}}$. In particular, we focus on the computational cost of $\hat{\boldsymbol{\xi}} = \boldsymbol{B}^{\dagger}\boldsymbol{y}$, with $\boldsymbol{B}^{\dagger}$ the Moore-Penrose pseudoinverse of $\boldsymbol{B} = \boldsymbol{A}^{\pm}\boldsymbol{D}_{|\hat{\boldsymbol{s}}}$, i.e., of a matrix with m rows and a number c of columns $\kappa \leq c \leq n$, with $c \simeq \kappa$ being the most frequent case. The computational complexity of pseudo-inversion reflects its analytical formula such that $\hat{\boldsymbol{\xi}} = \boldsymbol{B}^{\top}\left(\boldsymbol{B}\boldsymbol{B}^{\top}\right)^{-1}\boldsymbol{y}$ must be computed. Since $\boldsymbol{B}$ is a $m \times c$ matrix, $\boldsymbol{B}\boldsymbol{B}^{\top}$ requires m^2c MACs, and the inversion entails $2m^3$ MACs. Now, the right-multiplication $\left(\boldsymbol{B}\boldsymbol{B}^{\top}\right)^{-1}$ by $\boldsymbol{y}$ costs m^2 MACs and the final left-multiplication results in mc MACs. Considering all contributions, we arrive at estimating a total of $m(2m^2 + m\kappa + m + \kappa)$ for the typical $c = \kappa$ case. The complexity is then equal to $(2n\mathrm{CR}^{-2} + n\kappa/n\mathrm{CR}^{-1} + \kappa/n + \mathrm{CR}^{-1})n\mathrm{CR}^{-1}\ \mathrm{MAC/sample}$.

We may compare the complexity of TCSSO decoding with that of OMP, which is known to be one of the most light-weighted approaches. We consider the standard implementation of OMP as described in [20]. A modified version (bWOMP) of this algorithm has been proposed in [92] to exploit the same statistical prior described in [91] and improve reconstruction performance with no significant increase of computational complexity. The detailed description of OMP is out of the scope of this paper, and we refer to [20] for details. Knowing that OMP is an iterative algorithm that estimates the signal support in at least κ iteration, we limit ourselves to provide the complexity in terms of the number of MAC for the j-th iteration that is $nm + 2m(j-1) + 2m + 2jm$. This yields a total of at least $2\kappa m + 2\kappa^2 m + \kappa nm$ MACs. After that, OMP computes the pseudo-inverse of a matrix of the same size as the $\boldsymbol{B} = \boldsymbol{A}^{\pm}\boldsymbol{D}_{|\hat{\boldsymbol{s}}}$ in TCSSO. The total complexity of the iterative part is therefore given by $(2 + 2n\,\kappa/n + n)n\kappa/n\mathrm{CR}^{-1}$ $\mathrm{MAC/sample}$ and must be compared with the computational effort required by the oracle that is $(2/\mathrm{CR} + 7)n$ $\mathrm{MAC/sample}$.

Though the contributions to the computational complexities computed above have different asymptotic behaviors, their magnitude in the small-n cases can be appreciated only by numerical evaluation. As an example, for $n = 64$, $\kappa = 16$ and $\mathrm{CR} = 2$ (one of our ECG cases) the first part of OMP entails some $784\,\mathrm{MAC/sample}$, the oracle in TCSSO requires some $512\,\mathrm{MAC/sample}$, while the common pseudo-inversion amounts to $1304\,\mathrm{MAC/sample}$. As a further, somehow opposite, example, for $n = 128$, $\kappa = 26$ and $\mathrm{CR} = 4$ (one of our EEG cases) the first part of OMP entails some $1183\,\mathrm{MAC/sample}$, the oracle in TCSSO requires some $960\,\mathrm{MAC/sample}$, while the common pseudo-inversion amounts to $735\,\mathrm{MAC/sample}$.

In both cases, the complexity of TCSSO and that of OMP are analogous, showing that, though TCSSO allows implementing extremely lightweight encoders, the decoder does not have to compensate by increasing its computational requirements compared to conventional decoders. As a consequence, since bWOMP has complexity similar to OMP, at least in the settings we analyzed, the complexity of TCCSO is comparable to the one of decoders that use a statistical prior on the signal support.

5.7 Dealing with the flawed sparsity of realistic signals: TSOC decoder

Until this section, we have considered perfectly sparse bio-signals, with some preliminary results on applying the DNN-based two-step reconstruction proposed in [102] to realistic biomedical signals. In this section, we introduce further improvements to the CS encoding-decoding framework. In particular the support identification is generalized in case of realistic (and therefore non-perfectly sparse) signals. In fact, compared to [102, 103] and the previous sections of this work, the proposed support identification procedure takes into account not only the signal representation in the sparsity basis but also the encoder mechanism and possible sources of noise, as we presented in [21].

5.7.1 Support Oracle for Compressed Signals

When working with bio-signals, one can note that almost all classes of real signals are only *approximately sparse* since the vector $\boldsymbol{\xi} = \boldsymbol{D}^\top \boldsymbol{x}$ is composed of few entries with magnitude significantly greater than zero while the remaining entries are close to zero. In these cases, signals are not sparse but *compressible*.

As a result, it is not possible to define a support for $\boldsymbol{x}$ by looking at the vector $\boldsymbol{\xi}$, since any possible definition would cause rejection of a part of the signal information content. Let us therefore indicate with $\boldsymbol{z}$ a n-size binary vector such that $\boldsymbol{x} = \boldsymbol{D}_{|\boldsymbol{z}}\boldsymbol{\xi}_{|\boldsymbol{z}} + \boldsymbol{x}_d$, where $\boldsymbol{x}_d$ contains signal details of minor interest. With this notation, (5.7) can be reformulated as

$$\hat{\boldsymbol{\xi}}_{|\boldsymbol{z}} = \boldsymbol{B}^\dagger_{|\boldsymbol{z}}[\boldsymbol{y} - \boldsymbol{A}(\boldsymbol{x}_d + \boldsymbol{\nu})] = \boldsymbol{B}^\dagger_{|\boldsymbol{z}}\boldsymbol{y} - \boldsymbol{B}^\dagger_{|\boldsymbol{z}}\boldsymbol{A}(\boldsymbol{x}_d + \boldsymbol{\nu}) \tag{5.12}$$

where the reconstructed signal is now $\hat{\boldsymbol{x}} = \boldsymbol{D}_{|\boldsymbol{z}}\hat{\boldsymbol{\xi}}_{|\boldsymbol{z}}$ and the reconstruction error in the sparse representation is $\boldsymbol{e} = \boldsymbol{B}^\dagger_{|\boldsymbol{z}}\boldsymbol{A}(\boldsymbol{x}_d + \boldsymbol{\nu})$.

The choice of the support $\boldsymbol{z}$ is fundamental in limiting the error $\boldsymbol{e}$ since, along with $\boldsymbol{A}$ and $\boldsymbol{\nu}$, it defines the maximum achievable performance in terms of RSNR. Increasing the number of ones in $\boldsymbol{z}$ reduces the error contribution due to $\boldsymbol{x}_d$. On the other hand, reducing the number of ones in $\boldsymbol{z}$ also reduces the reconstruction error due to the relative decrease of the effect due to $\boldsymbol{\nu}$. The optimal support $\boldsymbol{z}^*$ that balances this trade-off can be obtained, in principle, by computing the RSNR for all possible $\boldsymbol{z}$. Unfortunately, finding $\boldsymbol{z}^*$ implies the solution of a combinatorial optimization problem that requires an exhaustive search over all possible 2^n binary $\boldsymbol{z}$ vectors.

$$\boldsymbol{z}^* = \arg \max_{\boldsymbol{z}\in\{0,1\}^n} \frac{\|\boldsymbol{x}\|_2}{\left\|\boldsymbol{x} - \boldsymbol{D}_{|\boldsymbol{z}}\hat{\boldsymbol{\xi}}_{|\boldsymbol{z}}\right\|_2} \tag{5.13}$$

To overcome the impasse, we propose a greedy approach capable of achieving a good approximation of $\boldsymbol{z}^*$ with complexity as low as n steps. First, the entries of $\boldsymbol{\xi}$ are sorted in decreasing order according to their magnitude. Following this order, the $\boldsymbol{z}$ vector, initialized with all zeros, is built iteratively by adding, step by step, a new non-zero element.

At each step, the RSNR is computed. The support $\boldsymbol{z}$ is identified as the vector that achieves the highest RSNR value. Note that the support $\boldsymbol{z}$ that we estimate is actually the extension of the support applied to a compressible signal once it has been compressed by the CS encoder, and depends both on the sensing matrix $\boldsymbol{A}$ and the noise $\boldsymbol{\nu}$. We refer to this as the *Support of Compressed signal* and to the CS decoder based on the pre-computation of $\boldsymbol{z}$ as the *Support Oracle for Compressed signals* (SOC).

To validate this approach, a dataset of synthetic ECSs generated according to [46] as in [103, 102] has been considered. Differently from the approaches in the latter contributions, we consider here the generated ECG as a compressible signal, i.e., we do not impose $\boldsymbol{x}_d$ to be null. Synthetic ECG instances are generated with sample rate $256\,\mathrm{sps}$ and time windows composed of $n = 128$ successive samples. The signal is then corrupted by additive Gaussian noise so that the intrinsic signal-to-noise ratio (ISNR) is set to $34\,\mathrm{dB}$ [130].

In the example of Figure 5.12 we have plotted the RSNR as a function of the number of ones in $\boldsymbol{z}$ for an instance of $\boldsymbol{x}$ and $\boldsymbol{\nu}$. The figure shows how both the maximum RSNR and the corresponding $\boldsymbol{z}$ cardinality depend on the number m of rows of $\boldsymbol{A}^{\pm}$, which is still antipodal. The figure also highlights the already observed trade-off.

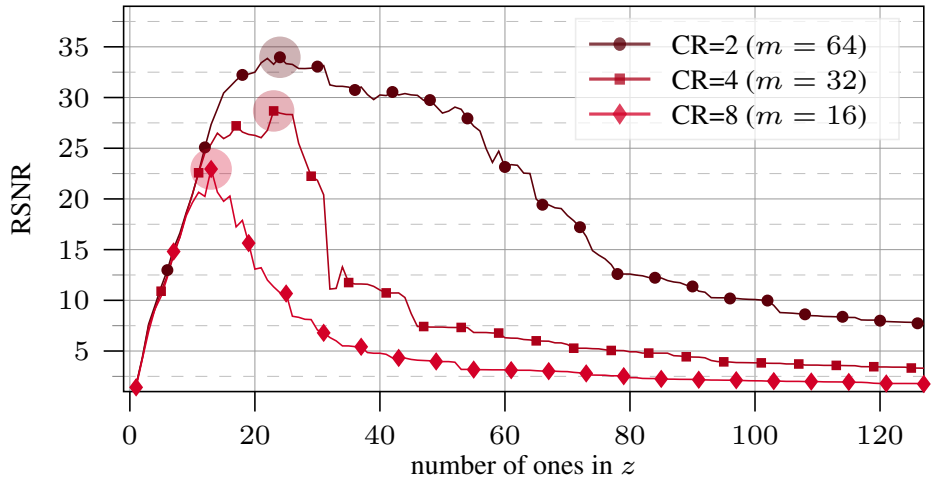


FIGURE 5.12: RSNR as a function of the number of ones in $\boldsymbol{z}$ where the position of the ones in $\boldsymbol{z}$ follows the position of $\boldsymbol{\xi}$ entries with the highest magnitudes. Different profiles refer to $m \times n$ Rak-CS antipodal sensing matrices with $n = 128$ and where the input signal is corrupted by additive Gaussian noise (ISNR=34 dB). (© 2020 IEEE)

When the cardinality of $\boldsymbol{z}$ is low, each new element inserted in the support is associated with a signal component with a large magnitude. This increases the RSNR since the projection of $\boldsymbol{x}$ along the corresponding column of $\boldsymbol{D}$ certainly exceeds the magnitude of the projection of $\boldsymbol{\nu}$ on the same column of $\boldsymbol{D}$. Conversely, when the cardinality of $\boldsymbol{z}$ is high, the additional signal information content brought by the new column of $\boldsymbol{D}$ could be lower than the corresponding noise contribution. Moreover, the RSNR values also depend on the adopted sensing matrix since the higher the compression ratio, the harder the reconstruction.

For each profile in Figure 5.12, the highlighted point represents the number of ones in $\boldsymbol{z}$ that maximizes the RSNR. Hence, according to the definition of $\boldsymbol{z}$ and the greedy method we propose, this point corresponds to the support $\boldsymbol{z}$ for a specific compressible signal instance $\boldsymbol{x}$, a noise vector $\boldsymbol{\nu}$, and a sensing matrix $\boldsymbol{A}^{\pm}$.

5.7.2 Training the support oracle

All the above considerations about the support oracle decoder for compressed signals are based on the possibility to retrieve the signal support $\boldsymbol{z}$ from the observation of the measurements vector $\boldsymbol{y}$. To perform this task, we adopt the same solution as in [102] described in Section 5.4. More specifically, the task performed by the DNN is equivalent to n parallel binary classification tasks that estimate the n entries of $\boldsymbol{z}$.

The structure of the network is the same as $\mathcal{N}_C$ in Figure 5.5. Neglecting the encoding stage, its input layer receives the m-dimensional measurement vector. As before, there are three fully connected hidden layers with $2n$, $2n$ an n neurons respectively and Rectified

TABLE 5.2: Number of DNN parameters used for the TSOC

m	16	20	24	28	32
# param	119 552	120 576	121 600	122 624	123 648
m	36	40	44	48	
# param	124 672	125 696	126 720	127 744	

Linear activation functions (ReLu). Finally, a fully connected output layer with n neurons and sigmoid activation functions produces a vector $\boldsymbol{o}$ with entries in $[0, 1]$. The final estimated support $\hat{\boldsymbol{z}}$ is obtained by applying a threshold $\theta \in [0, 1]$ to $\boldsymbol{o}$ such that $\boldsymbol{z}_j = 1$ if $o_j \geq \theta$, and $\hat{z}_j = 0$ otherwise. The CS decoder that adopts this DNN to divine $\boldsymbol{z}$ is named *Trained Support Oracle for Compressed signal* (TSOC).

Compared to the DNN proposed in [102] and used in the previous sections, the matrix $\boldsymbol{A}^{\pm}$ characterizing the encoder stage is not trained along with the rest of the network since the labels used during the training, i.e., the supports $\boldsymbol{z}$, depend also on the the sensing matrix. In light of that, the training exploits measurement vectors $\boldsymbol{y}$ computed with a fixed sensing matrix $\boldsymbol{A}^{\pm}$, which is still an antipodal matrix generated according to the Rak-CS approach for the considered class of signals. Note that, $\boldsymbol{A}^{\pm}$ plays the role of a set of hyperparameters that therefore cannot be trained with the parameters characterizing the neural network. Note also that $\boldsymbol{A}^{\pm}$ influences the network architecture since a different number of rows m of the sensing matrix corresponds to a different number of neurons in the input layer.

The DNN parameters set C (including weights and biases for each layer) are trained with a dataset of 2×10^6 signal instances $\boldsymbol{x}$ split in 95% for the actual training and 5% as a test set for performance assessment. For each different matrix $\boldsymbol{A}$, DNN input-output pairs $(\boldsymbol{y}, \boldsymbol{z})$ are obtained from both vectors $\boldsymbol{x}$ and randomly drawn noise contributions $\boldsymbol{\nu}$ such that $\boldsymbol{y} = \boldsymbol{A}^{\pm}(\boldsymbol{x} + \boldsymbol{\nu})$. For each value of m, we generate 100 different random candidates $\boldsymbol{A}^{\pm}$ according to the Rak-CS framework. Rak-CS (5.2) requires the signal correlation matrix $\boldsymbol{\mathcal{X}}$ which is estimated from 5000 signal instances generated especially for this purpose. Among these candidates $\boldsymbol{A}^{\pm}$, the matrix chosen for the encoder stage of the TSOC is the one obtaining the best ARSNR over 1000 signal reconstructions when a BPDN decoder is used[8].

Since m corresponds to both the number of rows in $A^{\pm}$ and the number of neurons in the input layer, the parameters characterizing the first hidden layer are $2nm$ weigths and $2n$ biases. For the case $n = 128$, Table 5.2 reports the total number of parameters for the adopted DNN settings where m ranges from 16 to 48.

Finally, 5000 new instances are used to tune the threshold θ applied to the DNN output $\boldsymbol{o}$. Resulting values of θ in all the considered settings are close to the middle-range value 0.5, i.e., values in $\boldsymbol{o}$ vectors concentrate close to the two boundary values zero and one.

[8]The BPDN is implementes with the Spectral Projected Gradient for L1 minimization (SPGL1) toolbox [131].

TABLE 5.3: Performance of the Oracle in terms of P, TP, TPR, TNR and ACC where $\mu(\cdot)$ means average over the test set.

m	μ(P)	μ(TP)	μ(TPR)	μ(TNR)	μ(ACC)
16	14.4	12.5	0.885	0.997	0.983
20	17.1	15.4	0.908	0.996	0.983
24	20.3	18.6	0.926	0.994	0.982
28	23.2	21.6	0.933	0.992	0.981
32	25.3	24.0	0.949	0.990	0.981
36	27.0	25.4	0.945	0.989	0.979
40	28.5	26.9	0.947	0.988	0.978
44	29.6	28.0	0.951	0.985	0.976
48	30.3	28.5	0.945	0.985	0.974

5.7.3 Results for ECGs

To assess the performance of the neural network architecture, we take into account different CS settings, each of which considers a matrix $\boldsymbol{A}$ with a different value for m that ranges from $m = 16$ (CR $= 4$) to $m = 48$ (CR $= 2.7$). Therefore, for each setting, we train a different set of parameters. As anticipated in the previous section, the task of the DNN is equivalent to a multi-label classification [132] that considers each label as an independent binary classification problem. More precisely, for each input $\boldsymbol{y}$, the DNN produces n outputs that are independently either positive or negative. If the i-th output is positive then $\hat{z}_i = 1$ while a negative outcome for the i-th output implies $\hat{z}_i = 0$.

To assess the capability of correctly estimating $\boldsymbol{z}$, metrics related to the difference between $\boldsymbol{z}$ and $\hat{\boldsymbol{z}}$ need to be adopted. Let first introduce the metrics for a single binary classification problem before generalize them for the case of n binary classifications performed by the DNN.

If both entries z_i and $\hat{z}_i$ are equal to 1 we mark this classification as a single true positive, while $z_i = \hat{z}_i = 0$ is a single true negative. In case of single miss-classifications we have either a single false positive ($\hat{z}_i = 1$ and $z_i = 0$) or a single false negative ($\hat{z}_i = 0$ and $z_i = 1$). Hence, the overall performance of a DNN prediction can be expressed in terms of the following metrics:

- Positive (P) and Negative (N):

$$\mathrm{P} = \sum_{i=0}^{n-1} z_i, \quad \mathrm{N} = n - \mathrm{P} \tag{5.14}$$

- True Positive (TP) and True Negative (TN)

$$\mathrm{TP} = \sum_{i=0}^{n-1} z_i \hat{z}_i, \quad \mathrm{TN} = \sum_{i=0}^{n-1} (1 - z_i)(1 - \hat{z}_i) \tag{5.15}$$

- TP Rate (TPR), TN Rate (TNR) and Accuracy (ACC)

$$\begin{aligned} \mathrm{TPR} &= \mathrm{TP}/\mathrm{P}, \quad \mathrm{TNR} = \mathrm{TN}/\mathrm{N}, \\ \mathrm{ACC} &= (\mathrm{TP} + \mathrm{TN})/n \end{aligned} \tag{5.16}$$

A summary of the average values for P, TP, TPR, TNR and ACC over the whole test set can be found in Table 5.3 with the aim of showing the ability of the network to correctly detects the ones in $\boldsymbol{z}$.

The results in Table 5.3 show that the number of ones in $\hat{\boldsymbol{z}}$ increases with m, confirming the behaviors of a single instance reported in Figure 5.12. The difference between P and TP, that is the number of ones wrongly estimated, is roughly constant and less than 2. As a consequence, the TPR tends to increase with m while the TNR slightly decreases. Since in general $\hat{z}$ contains more zeros than ones, the accuracy is dominated by TN so it slightly decreases with m, starting from 0.983 for $m = 16$ to 0.974 for $m = 48$.

Accuracy values very close to one ensure a high overlap between $\boldsymbol{z}$ and $\hat{\boldsymbol{z}}$. Nevertheless, even when $\boldsymbol{z} = \hat{\boldsymbol{z}}$, the decoder still commits an error in the reconstruction as modeled in (5.12). Thus, the overall performance of TSOC must be evaluated in terms of either ARSNR or RSNR distribution.

Figure 5.13 compares the performance in terms of the achieved ARSNR of the proposed TSOC approach with that of SOC (ideal oracle) and BPDN (SPGL1 decoder). All these approaches share the same tuned Rak-CS antipodal sensing matrices $\boldsymbol{A}$. Further comparison shows BPDN approach coupled with matrices $\boldsymbol{A}$ following the standard CS theory (Std) where -1 and +1 occur with the same probability. Rak + TSOC outperforms Rak + BPDN with a gap of at least $5\,\mathrm{dB}$, while the loss with respect to the ideal oracle (Rak + SOC) never exceeds $2.5\,\mathrm{dB}$. Std + BPDN performance is not even comparable with the ones of the other frameworks.

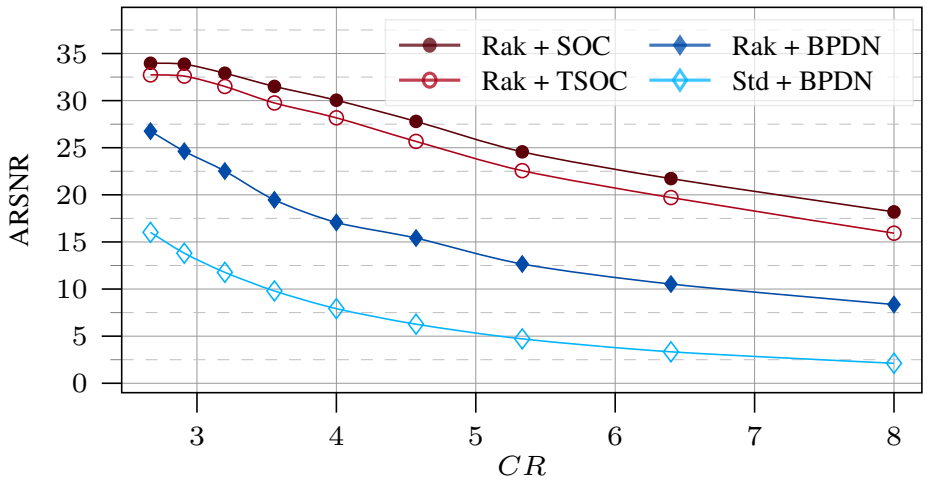


FIGURE 5.13: ARSNR as a function of CR for the proposed TOSC along with the ideal oracle (SOC) compared with the standard decoder (BPDN). The encoder stage follows Rak-CS and the standard CS encoder coupled with BPDN is provided as a reference. (© 2020 IEEE)

To provide a further comparison between these approaches, Figure 5.14 shows the RSNR distributions in case of CR $= 4$ ($m = 32$). The proposed TSOC, along with the ideal SOC, shows an RSNR variance that does not increase compared to both the already presented Std + BPDN and Rak + BPDB.

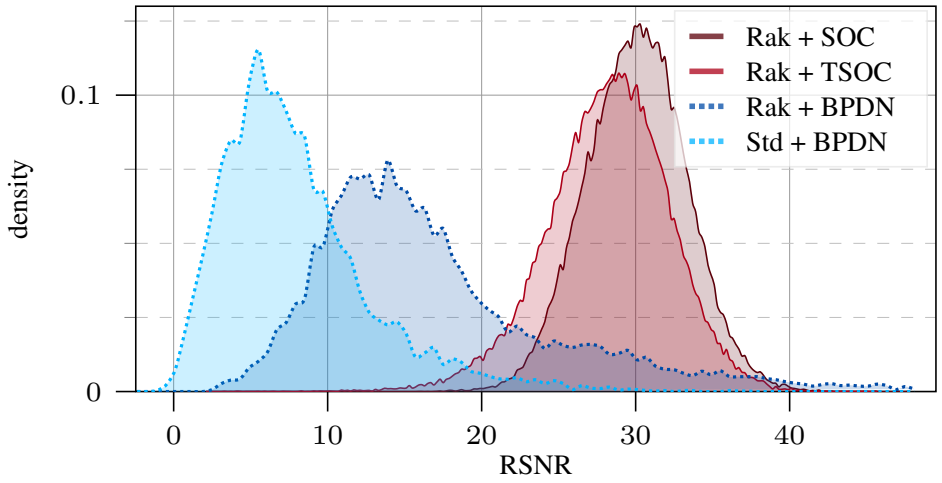


FIGURE 5.14: Probability density functions of RSNR values with CR$= 4$ ($m = 32$) for the considered system configurations.

5.8 Chapter conclusion

We propose a CS decoder that, starting from the compressed measurements, first guesses which components are non-zero in the sparse signal to recover, and then computes their magnitudes. Support guessing is provided by a suitable DNN-based oracle that reveals extremely accurate, especially when trained together with the encoding matrix.

The resulting decoder largely outperforms classical approaches, even when they are paired with one of the most effective adaptation policies for the encoding matrix, and allows the application of CS to signal windows containing a limited number of samples. The adoption of short windows is extremely beneficial along many directions; one of the most remarkable is the computational complexity of the encoder. Yet, short windows are usually out of the reach of classical CS mechanisms as the sparsity assumption on which they hinge tends to fail when the dimensionality of the waveform to compress decreases. Hence, our proposal allows the implementation of extremely low complexity encoders that still feature remarkable compression capabilities.

Furthermore, the separation between support guessing and magnitude calculation allows our decoder to detect cases in which the reconstruction may be affected by significant errors, thus paving the way, for example, to additional processing that further increases the reconstruction performance.

We demonstrated the effectiveness of this novel approach addressing ECG and EEG signals for which compression ratios above 2 can be reached with a computational burden not exceeding 32 signed sums per sample.

Additionally, we have take into account the flawed sparsity of realistic signals, proposing an improved version of the two-step CS decoder that searches for the best support for biological signals as a compromise between the loss of information due to the elimination of the smallest contributes in the sparsity domain and the insertion of noise due to the selection of unwanted contributes.

Chapter 6

Reducing the Complexity of the Deep Neural Oracles

In order to reduce the complexity of the TCSSO and TSOC decoders to minimize latency and power consumption, we propose different approaches based mainly on what has been discussed in Part I. So, in this chapter, different low-resource implementations of the DNN-based two-step decoder are presented with emphasis on: *i*) limiting the performance degradation when pruning the support oracle, i.e., when removing interconnections in the DNN; *ii*) limiting the performance degradation in case of fixed point precision arithmetic (for both DNN and pseudoinverse operation); *iii*) optimizing the memory footprint required by both DNN and pseudoinversion. To reach these goals, we prune and quantize the network and measure the impact of this compression operations on the performance of the system. First, we test pruning with MAM^2 on the layers with most weights in the support oracle. Then, we test the quantization of the parameters of the decoder. We perform a preliminary test with post-training quantization and then we employ quantization-aware techniques in the DNN training. Additionally, Least Mean Square (LMS) filter is employed instead of direct Moore-Penrose pseudoinverse implementation to reduce the computational complexity of the reconstruction phase.

6.1 Pruning the oracle: a MAM² application

We assess here the performance of a TCSSO decoder implementation that employs the MAM^2-based layer described in Section 3.1. For this purpose, we use a pre-sparsified synthetic ECG dataset to train and test the decoder, already introduced in Section 5.2. The performance is compared to a size-equivalent TCSSO decoder built only with classic fully-connected MAC-based layers. Pruning is performed on both MAM^2-based and MAC-based structures in order to minimize their complexity. Finally, the accuracy vs the number of parameters kept for the two structures is evaluated and their performance is compared.

6.1.1 Oracle DNN configuration

The structure of the DNN used in the oracle is the same as described in Section 5.4.1 and it is illustrated more in detail in Figure 6.1. The first layer is used as an encoder to optimize the sensing matrix $\boldsymbol{A}$ (that is not antipodal in this case), whose scalar values correspond to the

weights of the layer. It has n inputs and m outputs, uses a linear activation function and has no bias. It accepts as input the uncompressed signal $\boldsymbol{x}$ and generates the measurements $\boldsymbol{y}$. After training, this layer is detached and used for encoding/compression.

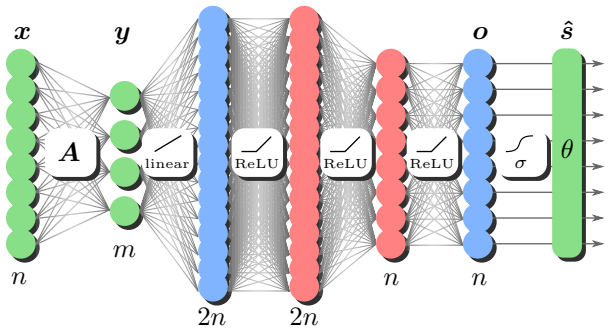


FIGURE 6.1: Structure of the DNN-based oracle. The first layer is used to optimize the sensing matrix $\boldsymbol{A}$ and has no bias nor activation function. In the TCSSO, the compressed signal $\boldsymbol{y}$ is fed directly to the second layer. The third and the fourth hidden layers are the ones that need most of the parameters and are the ones that are being pruned. (© 2022 IEEE)

As before, the following four fully-connected layers accept the measurements $\boldsymbol{y}$ as input and are used for support retrieval and have $2n$-$2n$-n-n neurons, respectively. All the layers but the last use a ReLU activation function, while the last uses a sigmoid function. The outputs, $\boldsymbol{o}$, are limited in $[0, 1]$ and then thresholded to obtain the support vector $\boldsymbol{s}$, which contains only zeros and ones. For this experiment, we use $n = 256$, $m = 64$ and a threshold value of 0.1.

In order to assess the performance of the layer previously described in Section 3.1, we use it to reduce the parameters needed by the DNN. We substitute the second and third and layers in the decoder with two MAM^2-based layers keeping the same number of neurons, as they contain about 80% of the total parameters (see Figure 6.1).

We compare the DNN that contains the MAM^2-layers with one completely built over classic MAC-based layers.

6.1.2 Oracle DNN training

Training is performed similarly to Section 5.4.1 with the vanishing contributes technique introduced in Section 3.1.2. Adam optimizer [40] is employed with learning rate = 0.001 and batch size = 256 for 350 epochs. Following the vanishing contributes technique and referring to (3.3), while training the MAM^2-based layers, we start with $\beta[1] = 1$ at epoch 1 and decrease it linearly to $\beta[15] = 0$ until epoch 15. After this, $\beta[p]$ is kept to 0 for $p > 15$.

Accuracy trend during training can be seen in Figure 6.2. As expected, when $\beta[p] > 0$ in the first 15 epochs all the contributes to the summation have not completely vanished yet. When $\beta[p] = 0$ all the contributes but the maximum and the minimum vanish, producing an initial decrease in accuracy that is quickly retrieved during the successive epochs.

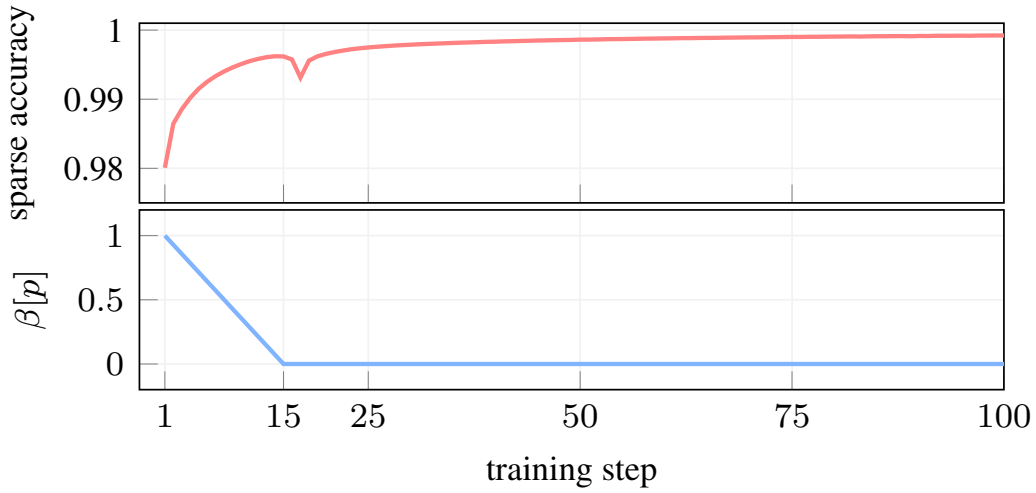


FIGURE 6.2: Sparse accuracy trend on the training set during the training of the DNN-based oracle, along with $\beta[p]$ trend, where p is the epoch number. After an initial accuracy increase, there is an accuracy drop when $\beta[p] = 0$ but the DNN quickly recovers its classification capability. (© 2022 IEEE)

6.1.3 Performance and size

The DNN-based oracle is trained and then the third and the fourth hidden layers are pruned, as they contain most of the parameters. The TCSSO decoder that employs the MAM^2-based layers is pruned using the activation rate method while the size-equivalent TCSSO that use only MAC-based layers is pruned using a magnitude-based approach, i.e., the parameters are scored according to their absolute value and the ones with the lowest scores are pruned. This is a typical approach for pruning MAC-based layers without retraining. The scores used for pruning are evaluated globally on the two layers.

Results can be found in Figure 6.3, where the ARSNR of the reconstructed signal is shown against the percentage of remaining parameters in the weight matrices of the third and of the fourth hidden layers. For each of the two TCSSO configurations, 5 different training and pruning processes have been run.

The MAM^2-based implementation presents the same baseline accuracy of the MAC-based structure. When both the solutions are pruned, MAM^2 layers perform better, allowing a further and significant reduction of the size of the DNN. As an example, in Figure 6.3 it is shown that the MAM^2-based layers can still retain the baseline accuracy with over 94% of their weights pruned while the MAC-based layers do the same with not more than 25% of pruned parameters. In order to further verify this result, we trained five more TCSSO configurations reducing the number of parameters without pruning, i.e., we select a lower number of neurons for the third and fourth hidden layers in the model[1]. As shown in Figure 6.3, none of this models achieve the same ARSNR of the model trained and pruned with the MAM^2-based layers.

[1]We keep the ratio 2:1 used in the original TCSSO which contains 512 and 256 neurons in the third and fourth hidden layers, respectively. The number of neurons for these two layers are 112-56, 96-48, 80-40, 64-32 and 48-24, defining each of the 5 extra configurations.

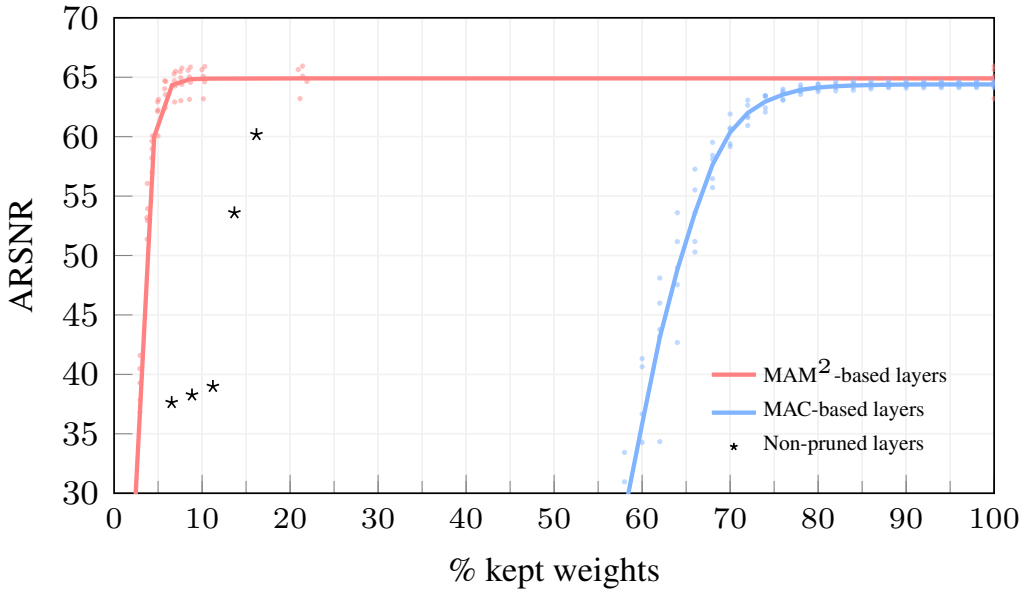


FIGURE 6.3: ARSNR vs the percentage of kept weights in the third and fourth hidden layers of the TCSSO ($m = 64$) after pruning. The mean performance over 5 tests is highlighted. Points marked with a star indicate non-pruned models trained directly with a smaller number of neurons. (© 2022 IEEE)

6.2 Post-training quantized TCSSO

We want to study how TCSSO behaves when we work on low-precision fixed-point values, as presented in [104]. Training of the DNN and a first evaluation of the performance are done with infinite precision (i.e., using a high-precision floating point representation). After this, the framework is quantized and the performance degradation is measured.

As in our framework we are assuming that measurement evaluation is done in analog domain, allowing a theoretical infinite precision, as for [20], in practice the measurements have a finite resolution dependent on the implementation. Given a normalized measurement vector $\boldsymbol{y} \in [-1, 1)$ with infinite resolution, we refer to (1.1) and we define the fixed-point finite-resolution equivalent as

$$y_{\text{quant},i} = \lfloor y \cdot 2^{f_y} \rceil \cdot 2^{-f_y} \quad \text{for } i = 1, \dots, m \tag{6.1}$$

where $\lfloor \cdot \rceil$ is the round to nearest operation, y_{quant_i} is the finite resolution version of the i-th entry of $\boldsymbol{y}$ and f_y is the number of fractional bits we want to employ so that the numeric precision we obtain is $\Delta_y = 2^{-f_y}$. The representation of y_{quant} includes also 1 sign bit. We also recall the set C of the parameters of the DNN $\mathcal{N}_C$ used for support divination with infinite precision. We want to test how the support oracle would perform with a reduced resolution of the parameters by employing a fixed-point set

$$c_{\text{quant},i} = \lfloor c_i \cdot 2^{f_w} \rceil \cdot 2^{-f_w} \quad \text{with } c_i \in C \tag{6.2}$$

where f_w is the number of fractional bits employed for the representation along with 1 sign bit and 1 integer bit and $\Delta_c = 2^{-f_y}$.

Finally, matrix $\boldsymbol{B}$ and matrix $\boldsymbol{D}$ are used in the reconstruction phase of the decoder. $\boldsymbol{B}$ can be expressed in its quantized form $\boldsymbol{B}_{\text{quant}}$ as

$$b_{\text{quant},ij} = \lfloor b_{ij} \cdot 2^{f_p} \rceil \cdot 2^{-f_p} \quad \text{for } i = 1, \ldots, m \text{ and } j = 1, \ldots, n \tag{6.3}$$

where b_{ij} and $b_{\text{quant},ij}$ are the values at row i and column j of matrices $\boldsymbol{B}$ and $\boldsymbol{B}_{\text{quant}}$, respectively. In the same way, $\boldsymbol{D}$ can be expressed in its quantized form $\boldsymbol{D}_{\text{quant}}$ as

$$d_{\text{quant},ij} = \lfloor d_{ij} \cdot 2^{f_p} \rceil \cdot 2^{-f_p} \quad \text{for } i = 1, \ldots, n \text{ and } j = 1, \ldots, n \tag{6.4}$$

where d_{ij} and $d_{\text{quant},ij}$ are the values at row i and column j of matrices $\boldsymbol{B}$ and $\boldsymbol{B}_{\text{quant}}$, respectively. Both matrices have f_p fractional bits used to encode the values of the two matrices. The encoding of the values of the two matrices requires also 1 sign bit while only $\boldsymbol{B}_{\text{quant}}$ requires 2 additional integer bits. Fixed-point matrix inversion and more specifically the Moore-Penrose pseudoinverse operation, which requires the singular value decomposition (SVD), are non-trivial tasks both in terms of computational complexity optimization and minimization of errors and their implementation is not obvious [133, 134, 135, 136]; as working on this is not the specific target of this analysis, for now we perform the second task with infinite precision while reducing only the precision of matrices $\boldsymbol{B}$ and $\boldsymbol{D}$ in order to save memory. The employment of efficient methods to perform the whole operation completely in fixed-point notation will be assessed in the next sections.

We measure performance on a synthetic ECG dataset generated as in Section 5.2. The dataset is composed of 8×10^5 instances each of them of size $n = 128$. A portion of the dataset which is the 80% of the total number of windows is used for both training (generation of $\boldsymbol{A}$ and the set of DNN parameters $\boldsymbol{C}$) and evaluation (define the threshold value $\boldsymbol{o}_{\text{th}}$). The remaining 20% is employed as test set for performance assessment. The sparsity matrix $\boldsymbol{D}$ we refer to is the $n \times n$ matrix representing the Symlet-6 wavelet family transformation as stated before. Training of the DNN is performed with stochastic gradient descent through 500 epochs with a mini-batch size of 30 instances using Keras API with Tensorflow backend [126, 125] while inference is performed with custom C++ compiled python operators which perform vector-matrix multiplications fully in fixed-point notation.

Each signal in the test set has been corrupted by additive white Gaussian noise to emulate possible sources of non-idealities. The added noise is such that intrinsic SNR is $60\,\text{dB}$.

Our goal is to measure how the reduction of data precision for measurements $\boldsymbol{y}$, for the DNN parameters C and for the matrices $\boldsymbol{D}$ and $\boldsymbol{B}$ impacts on the reconstruction of the compressed ECG signals. We take as reference the performance of a state-of-art method named rakeness-based CS [108, 109], which consists in the design of a sensing matrix adapted to the class of ECG signals, coupled with the standard BP decoder. Performance of this reference is evaluated with infinite precision.

6.2.1 Support Oracle with fixed-point precision

There are three degree of freedom we can work on when trying to reduce the complexity of the TCSSO: f_y, f_w and f_p. In this first phase Support Oracle performance alone is assessed by divining the support of $\boldsymbol{\xi}$ with low values of f_y and f_w while virtually keeping f_p to maximum, i.e. we perform pseudoinverse operations with infinite precision. In order to reduce the degrees of freedom we choose to fix f_y to 7 bits. Below this value, performance strongly deteriorates.

We test different values of f_w to minimize also the memory footprint of the application. Results can be seen in Figure 6.4 for CR values that go from 4 to 5.3. Even with low precision DNN parameters this method outperforms the aforementioned state-of-art CS approach (RAK+BP), whose performance is null for CR > 4. Furthermore, with $f_w = 7$, performance is about the same we have with infinite resolution.

6.2.2 Pseudoinverse operation with fixed-point precision

In the second phase we evaluate how using low values of f_p for matrices $\boldsymbol{D}$ and $\boldsymbol{B}$ affects the pseudoinverse operations while support divination is performed with infinite resolution (f_y and f_p are virtually kept to the maximum).

In Figure 6.5 we can see how performance varies with values of f_p which go from 16 bits to 5 bits. Performance starts to decrease at $f_p = 10$. With a higher compression ratio the RSNR values are less sensitive to the variation to f_p.

6.2.3 Full decoder with fixed-point precision

Finally, based on the discussed results, we set $f_y = 7$, $f_w = 5$ and $f_p = 10$. It should be noted that, by reducing the precision of $\boldsymbol{y}$ during the computation of $\hat{\boldsymbol{\xi}}_{|\hat{\boldsymbol{s}}}$, we are introducing a quantization error we did not estimate before. Nevertheless we observe in Figure 6.6 that, with this setting, the overall performance is mainly influenced by the resolution reduction of the DNN parameters.

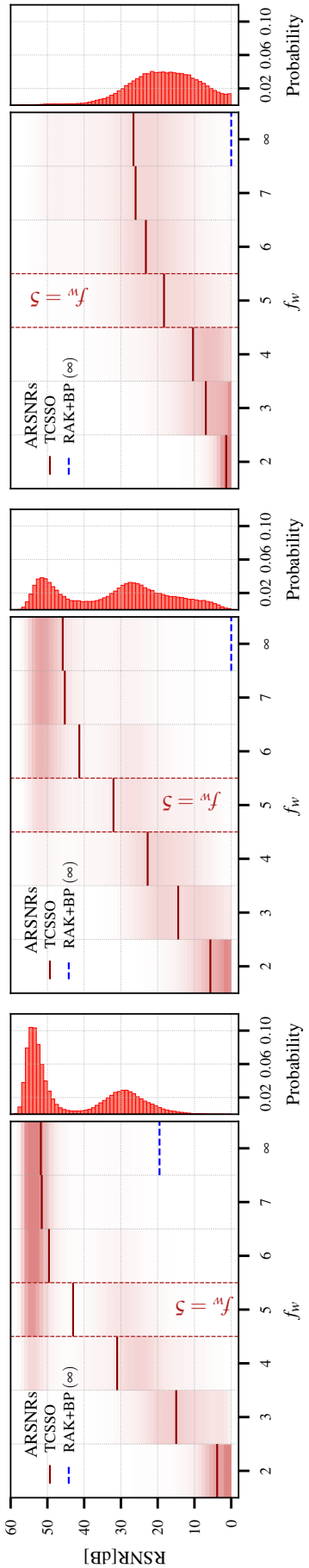


FIGURE 6.4: Performance for TCSSO with synthetic ECG dataset performed with fixed-point oracle and infinite precision pseudoinverse with the variation of f_w. The colored area in the background shows the probability of obtaining a given RSNR value while the values for $f_w = 5$ are shown on the histograms. The ARSNRs of TCSSO and of rakeness-based BP method are highlighted. (© 2020 IEEE)

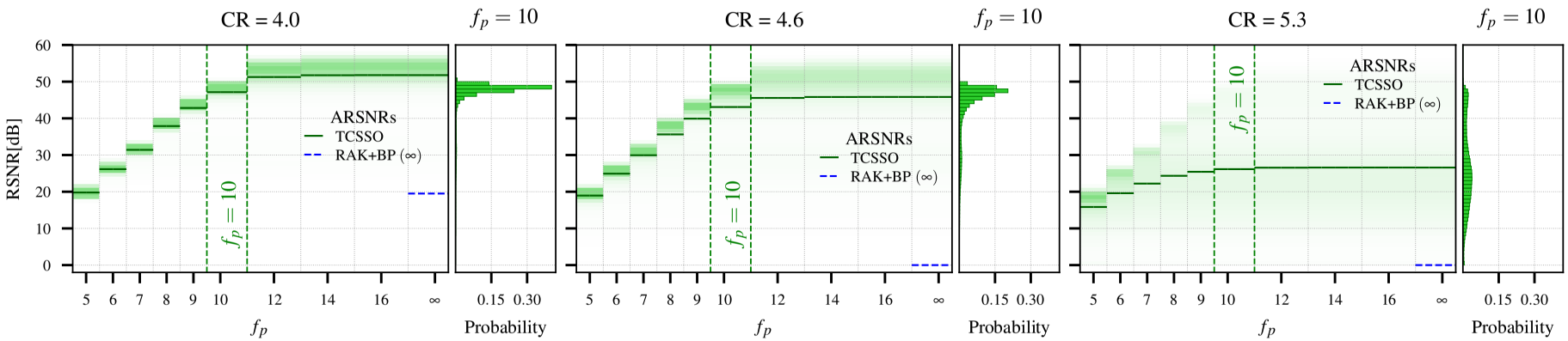


FIGURE 6.5: Performance for TCSSO with synthetic ECG dataset performed with infinite precision oracle and fixed-point $\boldsymbol{B}$ and $\boldsymbol{D}$ matrices with the variation of f_p. The colored area in the background shows the probability of obtaining a given RSNR value while the values for $f_p = 10$ are shown on the histograms. The ARSNRs of TCSSO and of rakeness-based BP method are highlighted. (© 2020 IEEE)

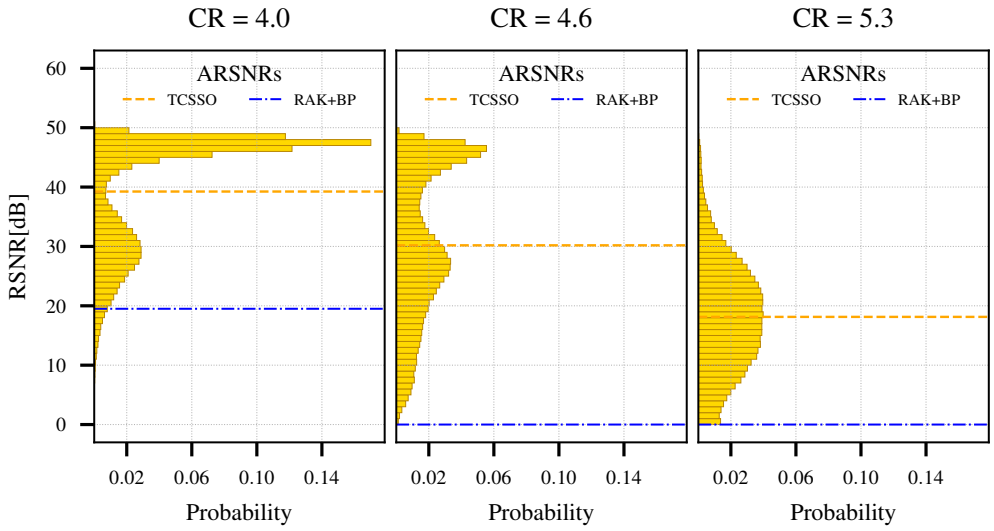


FIGURE 6.6: RSNR histogram for TCSSO with synthetic ECG dataset performed with fixed-point oracle and fixed-point B and D matrices: $f_y = 7$, $f_w = 5$ and $f_p = 10$. ARSNRs of TCSSO, rakeness-based BP and SDA are highlighted. (© 2020 IEEE)

6.3 Quantization-aware trained TSOC and implementation

We have seen in the previous section a preliminary test on a quantized version of TCSSO decoder. In this section, we explore the quantization of the improved TSOC decoder for compressible bio-signals without pre-sparsification. This time we both try to maximize the accuracy through quantization-aware techniques as introduced in Section 1.1 and to simplify the computational complexity of the operations by substituting the expensive pseudoinverse operation with a simple Least Mean Squares (LMS) filter.

The overall quantized system we are using is very similar to the one in the previous section and has been depicted in Figure 6.7 where we highlight the fact that each digital signal is associated to a number of bits. At the decoder side we can identify two main blocks: *i*) the oracle divining $\boldsymbol{z}$, *ii*) the reconstructor that uses the oracle output to recover the original waveform.

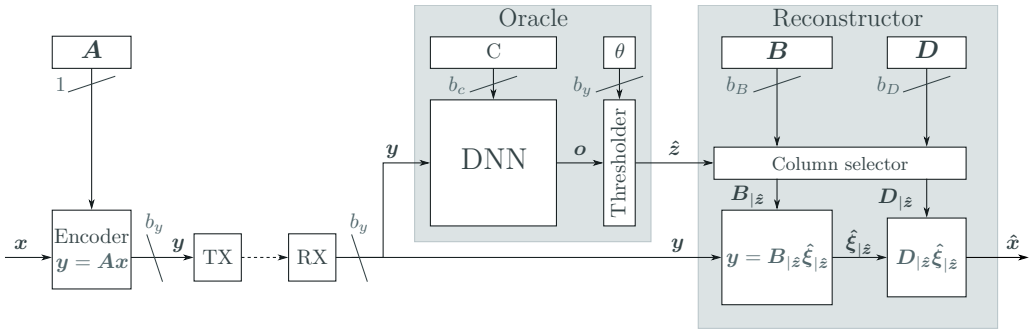


FIGURE 6.7: Block diagram of the overall system including CS encoder, measurements dispatch and the proposed TSOC decoder. The arrows represent digital quantities and for each of them the number of bits is highlighted. (© 2020 IEEE)

The representation of each system quantity with a finite number of bits addresses a trade-off between computational burden/memory footprint and capability to correctly reconstruct input signals.

In the previous section we quantized $\boldsymbol{y}$ with a naive approach, normalizing it in $[-1, 1)$. In reality, we do not know beforehand the values of $\boldsymbol{y}$ and we cannot define its range statically. So, the first issue we address is the quantization of the input of the decoder $\boldsymbol{y}$. Our analysis assumes that $\boldsymbol{y}$ is quantized by a mid-tread uniform quantizer, with 2^{b_y} levels.

Quantization may come either from the quantization of a measurement vector computed by an analog CS encoder block or from the digital processing of a digital input signal $\boldsymbol{x}$. In this setting, we need to remember that if n is large enough, each y_i, $i = 1, \ldots, m$ can be considered as a zero mean Gaussian distributed random variable, and setting a conversion range that includes all possible values is not possible. As introduced in Section 1.1, we then need to select the most convenient range value $\gamma\sigma_y$ to minimize the performance loss, where σ_y is the standard deviation of the values of $\boldsymbol{y}$ measured in our dataset and γ is a constant to be freely set.

The quantized $\boldsymbol{y}$ values feed both the oracle and the reconstructor. Each of the two structures internally uses parameters that can be quantized to reduce the complexity and memory footprint.

6.3.1 The quantization-aware trained oracle

As a first modification to reduce computational complexity of our DNN structure, we replace the sigmoid function in the output layer with a linear function. Since the sigmoid is monotone and the output is thresholded, it is sufficient to adapt the threshold value θ.

Then, the parameters of the oracle are quantized to minimize the memory footprint and reduce the resources needed for the support divination. All parameters of $\mathcal{N}_C$ are encoded with b_c bits and constrained to be in the discrete set $\Sigma_c = \{-1, -1 + \Delta_c, \ldots 1 - \Delta_c\}$, where Δ_c is the quantization step chosen be compliant with the adopted fixed point representation.

Moreover, in the hidden layers, all the neurons outputs (activations) are represented with only the b_y most significant bits such that their representation is coherent with the one of the DNN input. As a result, since both activations and parameters are represented in fixed point,

integer arithmetic is sufficient to produce the oracle output. In each neuron, the inputs are multiplied by the weights and then summed, therefore, the number of bits required by the arithmetic unit is $b_y + b_c + \log_2(n_L + 1)$ where n_L is the number of neuron inputs that in our case never exceeds $2n$.

In order to quantize the values, we adopt quantization-aware strategies during the training to reduce the performance loss due to quantization. In particular, we perform the following tasks:

- limiting the parameters in the set Σ_c so that they are not clipped during quantization by using bathtub regulraization and parameter recycling;
- limiting the activations in the set Σ_y to avoid overflow during inference with the use of a saturated ReLU activation function;
- updating parameters considering the effect of quantization (quantization-aware training), both with fake-quantization approach and cosine regularization approach.

All the details of these methods can be found in Section 1.1.

6.3.2 The LMS-based reconstructor

The final stage retrieves the vector $\boldsymbol{\xi}_{|\hat{\boldsymbol{z}}}$ by solving the equation $\boldsymbol{y} = \boldsymbol{B}_{|\hat{\boldsymbol{z}}}\boldsymbol{\xi}_{|\hat{\boldsymbol{z}}}$. Note that this is a resource-hungry operation.

Indeed, Moore-Penrose pseudo inversion is a computationally expensive operation since it requires a matrix inversion and several square roots operations. As a consequence, devices embedding a floating-point unit are preferred.

The first step adopted to reduce the reconstruction complexity is the quantization of both matrices $\boldsymbol{D}$ and $\boldsymbol{B}$ with respectively b_D and b_B bits. As a result, the memory footprint required for storing them is significantly reduced.

Moreover, since the pseudoinverse operation fundamentally solves the Least Mean Squares problem (LMS), it is possible to compute $\boldsymbol{\xi}_{|\hat{\boldsymbol{z}}}$ as the output of a 1-st order LMS filter [137], [138, Ch. 6]. The LMS filter employs only additions and multiplications, allowing the use of low-power fixed-point arithmetic, not possible with the pseudoinverse approach. Its mechanism is based on a gradient descent algorithm with a fixed amount of iterations q and a learning rate $\eta \ll 1$. The working principle of the 1-st order LMS filter can be summarized as follows. Given a system $\boldsymbol{y} = \boldsymbol{H}\boldsymbol{c}$, we have $\boldsymbol{c} \in \mathbb{R}^n$ unknown and $\boldsymbol{y} \in \mathbb{R}^m$ and $\boldsymbol{H} \in \mathbb{R}^{m \times n}$ known. We also define $\hat{\boldsymbol{c}} \in \mathbb{R}^n$ as the estimated coefficients vector. The pseudocode of the LMS filter is described in Algorithm 6.1.

In the algorithm, y_j is the j-th value of vector $\boldsymbol{y}$, $\boldsymbol{h}_{j,\cdot}$ is the j-th row of matrix $\boldsymbol{H}$ and $\eta \ll 1$ is the learning rate coefficient. In line 4, the error ϵ on the forward prediction of the single value y_j is evaluated. Then, in line 5, ϵ is backpropagated to update $\hat{\boldsymbol{c}}$. Both operations are first executed for all the entries of $\boldsymbol{y}$ (and so, for each row of $\boldsymbol{H}$) and then iterated q times to make the solution converge.

Algorithm 6.1 LMS filter

Set all the entries of $\hat{\boldsymbol{c}}$ to 0
for $i = 1, \ldots, q$ **do**
  for $j = 1, \ldots, m$ **do**
    $\epsilon = y_j - \boldsymbol{h}_{j,\cdot} \cdot \hat{\boldsymbol{c}}^\top$
    $\hat{\boldsymbol{c}} = \hat{\boldsymbol{c}} + \eta\epsilon \cdot \boldsymbol{h}_{j,\cdot}$
  end for
end for

6.3.3 Architecture design and results

Many hyperparameters need tuning for the final implementation of the overall decoder. They are summarized in Table 6.1. We focus here on the case CR $= 4$. To speed up the process, we split this investigation into two phases. Firstly, we determine the setting characterizing the LMS filter where the oracle is supposed ideal. Then, we focus on the tuning of the DNN hyperparameters.

TABLE 6.1: Hyper-parameters for the low-resources TSOC implementation

Decoder input	
b_y	Number of bits used to represent measurements y
$\gamma\sigma_y$	Half range of measurements y
Oracle	
b_c	Number of bits used to represent DNN weights
λ	Strength of cosine regularization
Reconstructor	
b_D	Number of bits used to store matrices D
b_B	Number of bits used to store matrices $B = AD$
q	Iterations of the LMS filter
η	LMS filter learning rate

For the LMS setting, we first perform a series of tests with 5000 signal instances, imposing $\hat{\boldsymbol{z}} = \boldsymbol{z}$, i.e., replacing the DNN with the ideal oracle. In practice, $b_B = 9$, the learning rate η of the LMS filter equal to 2^{-6} and $\gamma = 4$ are values that correspond to the maximum ARSNR value.

Conversely, q, b_D and b_y exhibit profiles that saturate as their values increase, as reported for a few configurations in Figure 6.8. In these cases we select the lower value below which the performance starts to degrade. The final values are $q = 512$, $b_D = 10$ and $b_y = 10$.

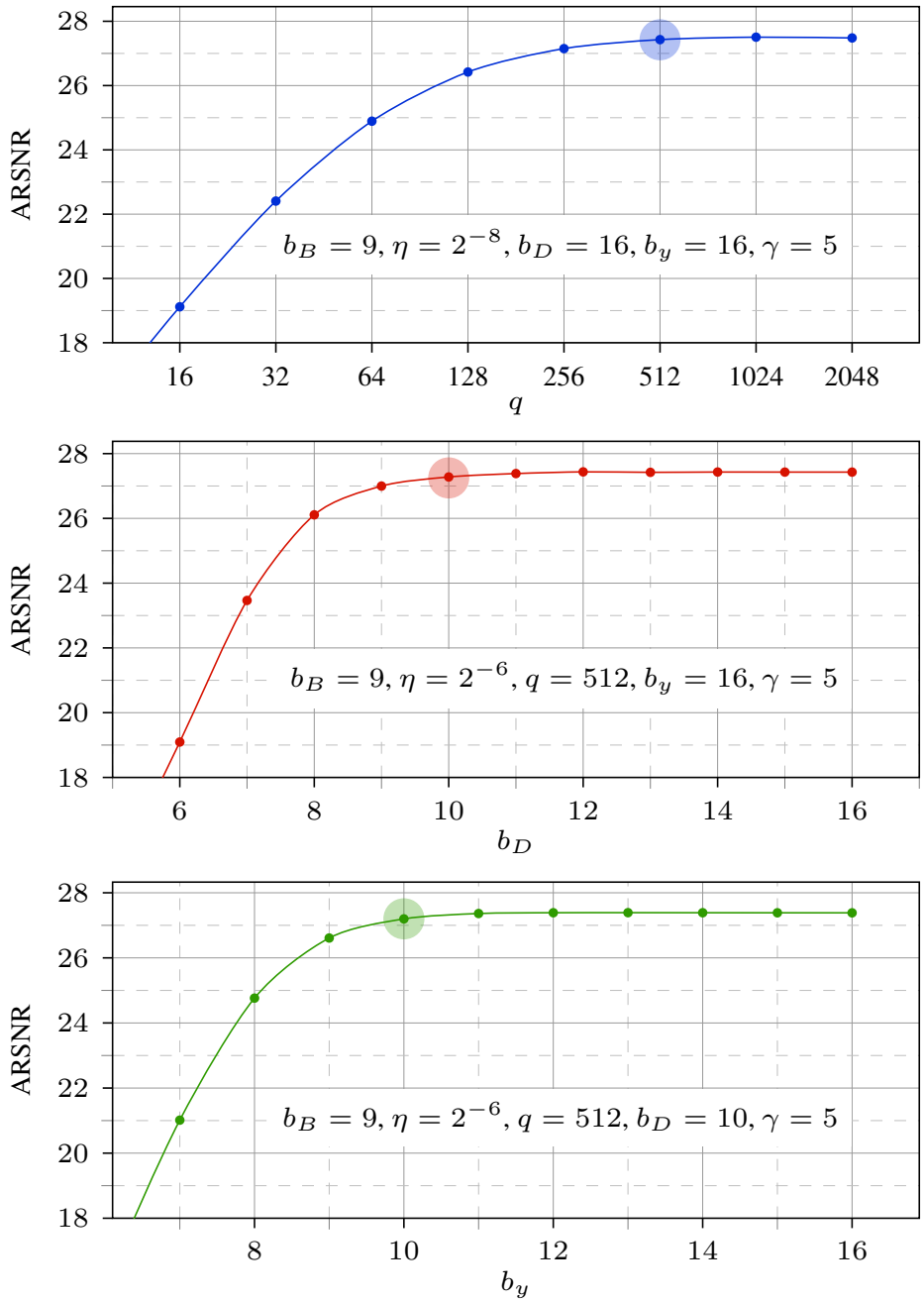


FIGURE 6.8: Performance of the decoder with $\hat{\boldsymbol{z}} = \boldsymbol{z}$, with the variation of the hyper-parameters q, b_D and b_y, which have not an optimum value (i.e. a single value with best performance). The values highlighted are the choices for our implementation test. (© 2020 IEEE)

Once the LMS filter setting is fixed, we consider the overall non-ideal decoder to tune the oracle hyperparameters b_c and λ. We, therefore, perform a DNN training for each hyperparameters configuration. Even considering different values for λ, the performance is almost constant when b_c is equal or greater than 4, and it significantly degrades only for b_c lower than 4. Since one of the aims is to reduce the memory footprint dedicated to the oracle, we set $b_c = 4$, and for that value, the optimal λ is 10^{-8}.

Therefore, considering $b_c = 4$, we need $495\,\mathrm{kbit}$ to store the DNN parameters (that are $123\,648$ for $m = 32$), $37\,\mathrm{kbit}$ for the 4096 entries of $\boldsymbol{B}$ ($b_B = 9$ bits each), and $164\,\mathrm{kbit}$ for the 16384 entries of $\boldsymbol{D}$ ($b_D = 10$ bits each). The overall memory footprint of the decoder is $695\,\mathrm{kbit}$, to which we must add $4\,\mathrm{kbit}$ needed for the 32×128 antipodal sensing matrix $\boldsymbol{A}^{\pm}$ in the encoder.

We finally test the whole quantized architecture, with quantized $\boldsymbol{y}$, C, $\boldsymbol{B}$ and $\boldsymbol{D}$, along with either the pseudoinverse on a floating-point unit or the LMS filter approach with fixed-point arithmetic. For this final test, we use 9×10^4 ECG windows with the same setup described in Section 5.7.1. The distribution of the RSNR can be found in Figure 6.9, along with the results presented in Section 5.7.1 for the TSOC case (without quantization). The main three cases to consider are therefore the ideal (i.e. high-precision) TSOC with pseudoinverse (PINV) and the two "constrained" cases of the quantized TSOC (qTSOC) with either the pseudoinverse (PINV) or the LMS filter. As it can be seen, the observed variances of each distribution are similar to each other, while the ARSNR for the qTSOC+LMS case is $25.9\,\mathrm{dB}$ and $26.6\,\mathrm{dB}$ for qTSOC+PINV, therefore showing only a slight degradation with respect to the $28.2\,\mathrm{dB}$ characterizing the reference TSOC+PINV setting. To offer an additional reference point, the performance of the case where the DNN parameters are quantized a-posteriori (post-qTSOC), i.e. after the training, is also shown in Figure 6.9. Without proper quantization-aware training techniques the performance is greatly degraded, with an ARSNR of $17.7\,\mathrm{dB}$.

Finally, In Figure 6.10 it is also possible to see some reconstructed ECG samples in the case of qTSOC with LMS filter along with the original instances.

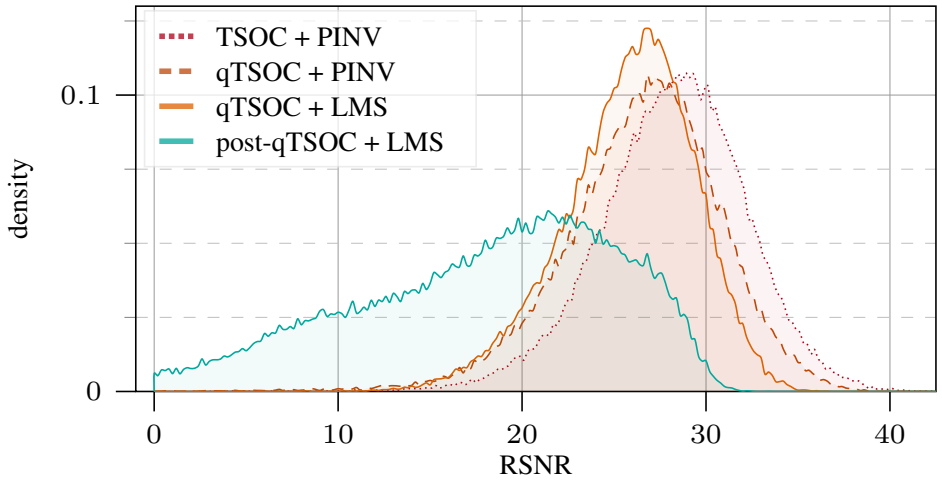


FIGURE 6.9: Probability density functions of RSNR values with CR= 4 ($m = 32$) for the high-precision TSOC and the quantized TSOC (qTSOC) with either the pseudoinverse (PINV) or the LMS filter. The performance in the case of a-posteriori quantization is also shown (post-qTSOC). (© 2020 IEEE)

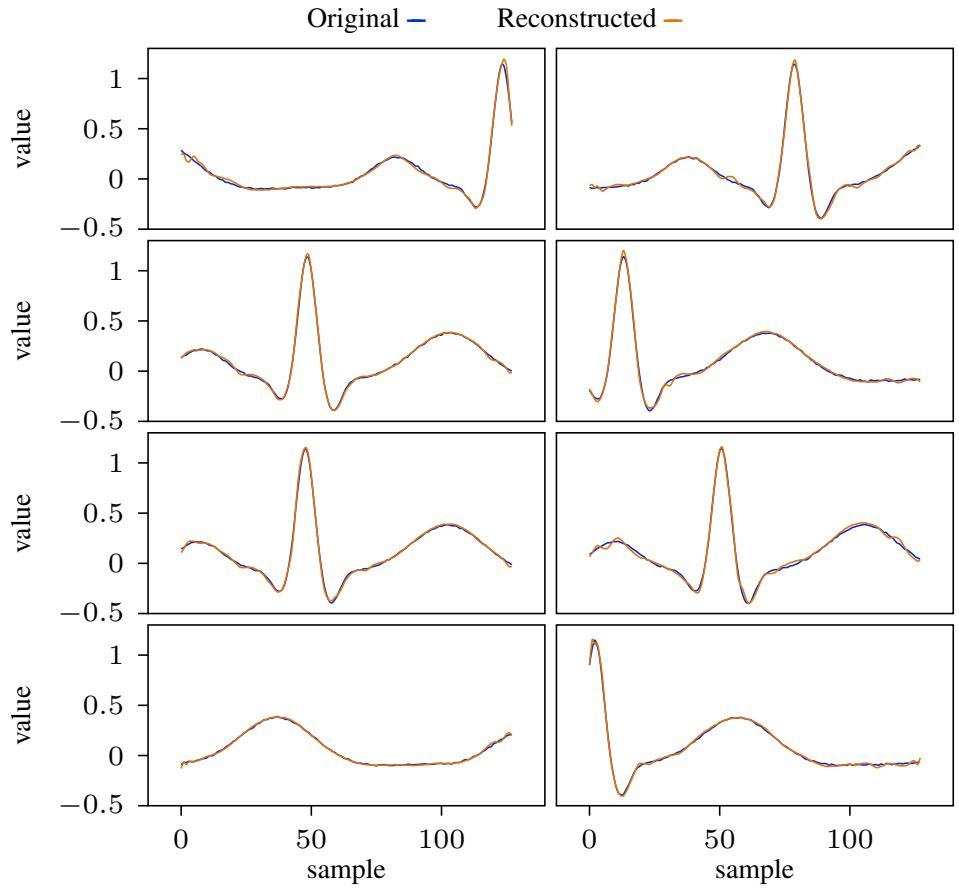


FIGURE 6.10: Samples of the original ECG windows compared with the windows reconstructed using qTSOC + LMS filter with CR $= 4$ ($m = 32$).

6.4 Chapter conclusion

We have applied to the two step CS decoders, namely TCSSO and TSOC, different pruning and quantization approaches that have been introduced in the previous chapters of this dissertation.

In a first experiment, we applied MAM^2 to TCSSO to prune the DNN-based support oracle structure. When employed in the decoder, MAM^2-based layers pruned with the probability of selection method allow a further reduction of the total number of parameters compared to classic MAC-based layers. This is achieved with no retraining thus shortening the design-to-deployment time of the decoder.

Then, we performed a preliminary test with fixed-point values for the parameters of the framework, i.e., we assumed a reduced number of bits for data, network parameters and sparsity matrix. The parameters are quantized a-posteriori. The achieved results on ECG signals show that the considered approach performs better with respect to other state-of-art approach even with a low resolution implementation.

Finally, we have considered the improved TSOC for support identification for compressed synthetic ECG signals, and we have trained the Support Oracle with a quantization-aware approach in order to minimize the accuracy loss due to the compression of the model. Also, we use an LMS filter, opposed to the pseudoinverse operation, to implement the reconstruction phase of the decoder. Results show that the proposed architecture is robust to quantization error and numerical errors. Moreover, with a low-precision fixed-point arithmetic unit, we achieve results that are a couple of dB lower with respect to those achievable by a standard high-precision implementation.

Conclusion

In this dissertation, we have presented different approaches to low-power, low-resources implementations for edge computing, referring both to the compression of the computational models – i.e., reduction in size and computational complexity of DNNs used for data elaboration at the edge – and to the deployment of data compression-decoding techniques – i.e., CS for biomedical data.

In Part I, we have presented different architectures and alternatives to standard MAC-based DNNs that allow a reduction in computational complexity and size of the implementation. In particular, SAM-based structures would allow both the removal of multiplications and the implementation of deeply pruned models, with the drawback of a loss in performance. On the other hand, MAM-based structures allow the implementation of deeply pruned models with a performance comparable to standard MAC-based solutions. Finally, a RSNN implementation on MCU has been tested as a starting point for the fast deployment of these low-power structures for time-dependent tasks. Future works on these topics revolve on the exploration of more difficult tasks (e.g., complex applications for robotics with event-based cameras), biggest models (e.g., transformer models) and on the implementation and test of convolutional layers (which have been left out in this dissertation).

In Part II, a novel task-specific, DNN-based, two-step CS decoder has been presented showing a state-of-the-art reconstruction performance. Given the small size of the model employed for the reconstruction, this structure can be deployed at the edge, thus removing the need of a stable connection to a central node when performing real time tasks and reducing the memory requirements for stored data. Last but not least, the DNN-based decoder can be deeply compressed in terms of computational complexity and size by using any of the methods in Part I, namely parameters quantization and interconnections pruning. In future works, the decoder should be tested on more signal classes, with broader real signal datasets, and on-device implementations should be tested in order to measure the energy consumption and on-field compression-decoding capabilities.

Bibliography

[1] T Mitchell et al. “Machine Learning”. In: *Annual Review of Computer Science* 4.1 (1990), pp. 417–433. DOI: 10.1146/annurev.cs.04.060190.002221. (Visited on 11/11/2022).

[2] Christopher M Bishop. *Pattern Recognition and Machine Learning*. en. Springer New York. ISBN: 978-1-4939-3843-8.

[3] Christian Janiesch, Patrick Zschech, and Kai Heinrich. “Machine learning and deep learning”. en. In: *Electronic Markets* 31.3 (Sept. 2021), pp. 685–695. ISSN: 1422-8890. DOI: 10.1007/s12525-021-00475-2.

[4] Poonam Sharma and Akansha Singh. “Era of deep neural networks: A review”. In: *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. July 2017, pp. 1–5. DOI: 10.1109/ICCCNT.2017.8203938.

[5] Shanzhi Chen et al. “A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective”. In: *IEEE Internet of Things Journal* 1.4 (Aug. 2014). Conference Name: IEEE Internet of Things Journal, pp. 349–359. ISSN: 2327-4662. DOI: 10.1109/JIOT.2014.2337336.

[6] R. G. Baraniuk et al. “Special issue on compressive sampling”. In: *IEEE Signal Processing Magazine* 25.2 (Mar. 2008). ISSN: 1053-5888.

[7] D. L. Donoho. “Compressed Sensing”. In: *IEEE Trans. on Inf. Theory* 52.4 (Apr. 2006), pp. 1289–1306. ISSN: 0018-9448. DOI: 10.1109/TIT.2006.871582.

[8] E. J. Candes, J. Romberg, and T. Tao. “Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information”. In: *IEEE Trans. on Inf. Theory* 52.2 (Feb. 2006), pp. 489–509. ISSN: 0018-9448. DOI: 10.1109/TIT.2005.862083.

[9] W. Shi et al. “Edge Computing: Vision and Challenges”. In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2579198.

[10] Dr. Lachit Dutta and Swapna Bharali. “TinyML Meets IoT: A Comprehensive Survey”. en. In: *Internet of Things* 16 (Dec. 2021), p. 100461. ISSN: 2542-6605. DOI: 10.1016/j.iot.2021.100461. (Visited on 11/11/2022).

[11] Partha Pratim Ray. "A review on TinyML: State-of-the-art and prospects". en. In: *Journal of King Saud University - Computer and Information Sciences* 34.4 (Apr. 2022), pp. 1595–1623. ISSN: 1319-1578. DOI: 10.1016/j.jksuci.2021.11.019. (Visited on 11/11/2022).

[12] Ofir Zafrir et al. "Q8BERT: Quantized 8Bit BERT". In: *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*. Dec. 2019, pp. 36–39. DOI: 10.1109/EMC2-NIPS53020.2019.00016.

[13] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "BinaryConnect: Training Deep Neural Networks with Binary Weights During Propagations". In: *28th International Conference on Neural Information Processing Systems (NIPS'15)*. MIT Press, Dec. 2015, pp. 3123–3131.

[14] Matthieu Courbariaux et al. "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1". In: *arXiv:1602.02830 [cs]* (Mar. 2016). (Visited on 10/06/2021).

[15] Mohammad Rastegari et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". In: *14th European Conference on Computer Vision (ECCV 2016)*. Ed. by Bastian Leibe et al. Oct. 2016, pp. 525–542. ISBN: 978-3-319-46493-0. DOI: 10.1007/978-3-319-46493-0_32.

[16] Shien Zhu, Luan H. K. Duong, and Weichen Liu. "XOR-Net: An Efficient Computation Pipeline for Binary Neural Network Inference on Edge Devices". In: *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. Dec. 2020, pp. 124–131. DOI: 10.1109/ICPADS51040.2020.00026.

[17] Valentino Peluso and Andrea Calimera. "Energy-driven precision scaling for fixed-point ConvNets". In: *2018 IFIP/IEEE int. Conf. Very large scale integr.* Oct. 2018, pp. 113–118. DOI: 10.1109/VLSI-SoC.2018.8644902.

[18] Chang Song et al. "A quantization-aware regularized learning method in multilevel memristor-based neuromorphic computing system". In: *2017 IEEE 6th non-volatile mem. Syst. Appl. Symp.* Aug. 2017, pp. 1–6. DOI: 10.1109/NVMSA.2017.8064465.

[19] Mauro Mangia et al. "Coping with saturating projection stages in RMPI-based Compressive Sensing". In: *2012 IEEE int. Symp. Circuits syst.* IEEE, May 2012, pp. 2805–2808. DOI: 10.1109/ISCAS.2012.6271893.

[20] Fabio Pareschi et al. "Hardware-Algorithms Co-Design and Implementation of an Analog-to-Information Converter for Biosignals Based on Compressed Sensing". In: *IEEE Trans. Biomed. Circuits Syst.* 10.1 (Feb. 2016), pp. 149–162. ISSN: 1932-4545. DOI: 10.1109/TBCAS.2015.2444276.

[21] Luciano Prono et al. "Deep Neural Oracle With Support Identification in the Compressed Domain". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 10.4 (Dec. 2020), pp. 458–468. ISSN: 2156-3365. DOI: 10.1109/JETCAS.2020.3039731.

[22] Davis Blalock et al. "What is the State of Neural Network Pruning?" en. In: *Proceedings of Machine Learning and Systems* 2 (Mar. 2020), pp. 129–146. (Visited on 08/30/2022).

[23] Zhuang Liu et al. "Rethinking the Value of Network Pruning". en. In: *7th International Conference on Learning Representations (ICLR 2019)*. Sept. 2018. (Visited on 01/20/2022).

[24] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *7th International Conference on Learning Representations (ICLR 2019)*. Sept. 2018. (Visited on 01/20/2022).

[25] Song Han et al. "Learning both Weights and Connections for Efficient Neural Network". In: *28th International Conference on Neural Information Processing Systems (NIPS'15)*. Vol. 28. Oct. 2015.

[26] Chaoqi Wang, Guodong Zhang, and Roger Grosse. "Picking Winning Tickets Before Training by Preserving Gradient Flow". en. In: *11th International Conference on Learning Representations (ICLR)*. Feb. 2022. (Visited on 08/29/2022).

[27] Christos Louizos, Max Welling, and Diederik P. Kingma. "Learning Sparse Neural Networks through L0 Regularization". In: *6th International Conference on Learning Representations (ICLR 2018)*. Feb. 2018. (Visited on 01/20/2022).

[28] Ao Ren et al. "ADMM-NN: An Algorithm-Hardware Co-Design Framework of DNNs Using Alternating Direction Methods of Multipliers". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 925–938. ISBN: 978-1-4503-6240-5. DOI: 10.1145/3297858.3304076. (Visited on 08/29/2022).

[29] Peisong Wang et al. "Extremely Sparse Networks via Binary Augmented Pruning for Fast Image Classification". In: *IEEE Transactions on Neural Networks and Learning Systems* (2021), pp. 1–14. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2021.3120409.

[30] X. Dai, H. Yin, and N. K. Jha. "NeST: A Neural Network Synthesis Tool Based on a Grow-and-Prune Paradigm". In: *IEEE Transactions on Computers* 68.10 (Oct. 2019), pp. 1487–1497. ISSN: 1557-9956. DOI: 10.1109/TC.2019.2914438.

[31] Yihui He, Xiangyu Zhang, and Jian Sun. "Channel Pruning for Accelerating Very Deep Neural Networks". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. ISSN: 2380-7504. Oct. 2017, pp. 1398–1406. DOI: 10.1109/ICCV.2017.155.

[32] Zhiqiang Chen et al. “Dynamical Channel Pruning by Conditional Accuracy Change for Deep Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.2 (Feb. 2021), pp. 799–813. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2020.2979517.

[33] Yang He et al. “Filter Pruning by Switching to Neighboring CNNs With Good Attributes”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022), pp. 1–13. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2022.3149332.

[34] Kaveena Persand, Andrew Anderson, and David Gregg. “Taxonomy of Saliency Metrics for Channel Pruning”. In: *IEEE Access* 9 (2021), pp. 120110–120126. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3108545.

[35] Wei Niu et al. “PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-based Weight Pruning”. In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’20. New York, NY, USA: Association for Computing Machinery, Mar. 2020, pp. 907–922. ISBN: 978-1-4503-7102-5. DOI: 10.1145/3373376.3378534. (Visited on 09/05/2022).

[36] Luciano Prono et al. “A Non-conventional Sum-and-Max based Neural Network layer for Low Power Classification”. In: *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. May 2022.

[37] Philippe Bich et al. “Aggressively prunable MAM²-based Deep Neural Oracle for ECG acquisition by Compressed Sensing”. In: *2022 IEEE Biomedical Circuits and Systems Conference (BIOCAS)*. Oct. 2022.

[38] C. G. Khatri and C. Radhakrishna Rao. “Solutions to Some Functional Equations and Their Applications to Characterization of Probability Distributions”. In: *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)* 30.2 (1968). Publisher: Springer, pp. 167–180. ISSN: 0581-572X. (Visited on 08/31/2022).

[39] Mauro Mangia et al. *Adapted compressed sensing for effective hardware implementations: A design flow for signal-level optimization of compressed sensing stages*. Springer International Publishing, 2018. ISBN: 978-3-319-61372-7.

[40] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 2017. DOI: 10.48550/arXiv.1412.6980. (Visited on 09/05/2022).

[41] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791.

[42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. (Visited on 09/20/2022).

[43] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *3rd International Conference on Learning Representations (ICLR 2015)*. May 2015.

[44] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. Publisher: Citeseer. University of Toronto, 2009.

[45] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[46] Patrick E McSharry et al. "A dynamical model for generating synthetic electrocardiogram signals". In: *IEEE Trans. on Biom. Eng.* 50.3 (Mar. 2003), pp. 289–294. ISSN: 0018-9294. DOI: 10.1109/TBME.2003.808805.

[47] D. P. Kingma and L. J. Ba. "Adam: A Method for Stochastic Optimization". en. In: (2015). Publisher: Ithaca, NYarXiv.org. URL: https://dare.uva.nl/search?identifier=a20791d3-1aff-464a-8544-268383c33a75 (visited on 10/12/2021).

[48] Samanwoy Ghosh-Dastidar and Hojjat Adeli. "Spiking neural networks". In: *International Journal of Neural Systems* 19.04 (Aug. 2009), pp. 295–308. ISSN: 0129-0657. DOI: 10.1142/S0129065709002002. (Visited on 08/30/2022).

[49] Amirhossein Tavanaei et al. "Deep learning in spiking neural networks". en. In: *Neural Networks* 111 (Mar. 2019), pp. 47–63. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2018.12.002. (Visited on 08/30/2022).

[50] Guillaume Bellec et al. "A solution to the learning dilemma for recurrent networks of spiking neurons". en. In: *Nature Communications* 11.1 (July 2020), p. 3625. ISSN: 2041-1723. DOI: 10.1038/s41467-020-17236-y. (Visited on 08/30/2022).

[51] P. Lichtsteiner, C. Posch, and T. Delbruck. "A 128 X 128 120db 30mw asynchronous vision sensor that responds to relative intensity change". In: *2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers*. ISSN: 2376-8606. Feb. 2006, pp. 2060–2069. DOI: 10.1109/ISSCC.2006.1696265.

[52] Arnon Amir et al. "A Low Power, Fully Event-Based Gesture Recognition System". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 1063-6919. July 2017, pp. 7388–7397. DOI: 10.1109/CVPR.2017.781.

[53] Yechan Park et al. "A Wireless Power and Data Transfer IC for Neural Prostheses Using a Single Inductive Link With Frequency-Splitting Characteristic". In: *IEEE Transactions on Biomedical Circuits and Systems* 15.6 (Dec. 2021), pp. 1306–1319. ISSN: 1932-4545, 1940-9990. DOI: 10.1109/TBCAS.2021.3135843.

[54] Hongmin Li et al. "CIFAR10-DVS: An Event-Stream Dataset for Object Classification". In: *Frontiers in Neuroscience* 11 (2017). ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00309. (Visited on 10/21/2022).

[55] Cedric Scheerlinck et al. "CED: Color Event Camera Dataset". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. ISSN: 2160-7516. June 2019, pp. 1684–1693. DOI: 10.1109/CVPRW.2019.00215.

[56] Mathias Gehrig et al. “DSEC: A Stereo Event Camera Dataset for Driving Scenarios”. In: *IEEE Robotics and Automation Letters* 6.3 (July 2021), pp. 4947–4954. ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3068942.

[57] Etienne Perot et al. “Learning to Detect Objects with a 1 Megapixel Event Camera”. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 16639–16652. (Visited on 10/21/2022).

[58] Peter Dayan and Laurence F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. en. Google-Books-ID: fLT4DwAAQBAJ. MIT Press, Aug. 2005. ISBN: 978-0-262-54185-5.

[59] Ali Samadzadeh et al. *Convolutional Spiking Neural Networks for Spatio-Temporal Feature Extraction*. Jan. 2021. DOI: 10.48550/arXiv.2003.12346. (Visited on 10/21/2022).

[60] Bing Han and Kaushik Roy. “Deep Spiking Neural Network: Energy Efficiency Through Time Based Coding”. en. In: *Computer Vision – ECCV 2020*. 2020, pp. 388–404. ISBN: 978-3-030-58607-2. DOI: 10.1007/978-3-030-58607-2_23.

[61] Arun M. George et al. “A Reservoir-based Convolutional Spiking Neural Network for Gesture Recognition from DVS Input”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. ISSN: 2161-4407. July 2020, pp. 1–9. DOI: 10.1109/IJCNN48605.2020.9206681.

[62] Yongqiang Cao, Yang Chen, and Deepak Khosla. “Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition”. en. In: *International Journal of Computer Vision* 113.1 (May 2015), pp. 54–66. ISSN: 1573-1405. DOI: 10.1007/s11263-014-0788-3. (Visited on 10/21/2022).

[63] Seijoon Kim et al. “Spiking-YOLO: Spiking Neural Network for Energy-Efficient Object Detection”. en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.07 (Apr. 2020), pp. 11270–11277. ISSN: 2374-3468. DOI: 10.1609/aaai.v34i07.6787. (Visited on 10/21/2022).

[64] Chankyu Lee et al. “Spike-FlowNet: Event-Based Optical Flow Estimation with Energy-Efficient Hybrid Neural Networks”. en. In: *Computer Vision – ECCV 2020*. 2020, pp. 366–382. ISBN: 978-3-030-58526-6. DOI: 10.1007/978-3-030-58526-6_22.

[65] Bojian Yin, Federico Corradi, and Sander M. Bohté. “Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks”. en. In: *Nature Machine Intelligence* 3.10 (Oct. 2021), pp. 905–913. ISSN: 2522-5839. DOI: 10.1038/s42256-021-00397-w. (Visited on 10/30/2022).

[66] Charlotte Frenkel and Giacomo Indiveri. “ReckOn: A 28nm Sub-mm2 Task-Agnostic Spiking Recurrent Neural Network Processor Enabling On-Chip Learning over Second-Long Timescales”. In: *2022 IEEE International Solid- State Circuits Conference (ISSCC)*. Vol. 65. ISSN: 2376-8606. Feb. 2022, pp. 1–3. DOI: 10.1109/ISSCC42614.2022.9731734.

[67] Garrick Orchard et al. "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades". In: *Frontiers in Neuroscience* 9 (2015). ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00437. (Visited on 10/21/2022).

[68] P.J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (Oct. 1990), pp. 1550–1560. ISSN: 1558-2256. DOI: 10.1109/5.58337.

[69] Ruthvik Vaila, John Chiasson, and Vishal Saxena. "Feature Extraction using Spiking Convolutional Neural Networks". In: *Proceedings of the International Conference on Neuromorphic Systems (ICONS '19)*. New York, NY, USA, 2019, pp. 1–8. ISBN: 978-1-4503-7680-8. DOI: 10.1145/3354265.3354279. (Visited on 10/24/2022).

[70] Xiang Cheng et al. "LISNN: Improving Spiking Neural Networks with Lateral Interactions for Robust Object Recognition". en. In: *Twenty-Ninth International Joint Conference on Artificial Intelligence*. Vol. 2. ISSN: 1045-0823. July 2020, pp. 1519–1525. DOI: 10.24963/ijcai.2020/211. (Visited on 10/24/2022).

[71] Sumit Bam Shrestha and Garrick Orchard. "SLAYER: spike layer error reassignment in time". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Red Hook, NY, USA, 2018, pp. 1419–1428. (Visited on 10/24/2022).

[72] Wenrui Zhang and Peng Li. "Temporal Spike Sequence Learning via Backpropagation for Deep Spiking Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 12022–12033. (Visited on 10/24/2022).

[73] Alexandros Pantelopoulos and Nikolaos G. Bourbakis. "A Survey on Wearable Sensor-Based Systems for Health Monitoring and Prognosis". In: *IEEE Transactions on Systems, Man, and Cybernetics* 40.1 (Jan. 2010), pp. 1–12. ISSN: 1558-2442. DOI: 10.1109/TSMCC.2009.2032660.

[74] R. G. Baraniuk et al. "Special issue on applications of sparse representation and compressive sensing". In: *Proceedings of the IEEE* 98.6 (June 2010). ISSN: 0018-9219.

[75] D. Allstot, R. Rovatti, and G. Setti (Eds.) "Special issue on circuits, systems and algorithms for compressed sensing". In: *IEEE J. Emerg. Sel. Topics Circuits Syst.* 2.3 (Sept. 2012). ISSN: 2156-3357.

[76] Giacomo Oliveri, Paolo Rocca, and Andrea Massa. "A Bayesian Compressive Sampling-Based Inversion for Imaging Sparse Scatterers". In: *IEEE Transactions on Geoscience and Remote Sensing* 49.10 (Oct. 2011), pp. 3993–4006. ISSN: 1558-0644. DOI: 10.1109/TGRS.2011.2128329.

[77] H. O. A. Ahmed and A. K. Nandi. "Three-stage hybrid fault diagnosis for rolling bearings with compressively sampled data and subspace learning techniques". In: *IEEE Transactions on Industrial Electronics* 66.7 (July 2019), pp. 5516–5524. ISSN: 1557-9948. DOI: 10.1109/TIE.2018.2868259.

[78] H. Shen et al. “Compressed sensing-based inpainting of aqua moderate resolution imaging spectroradiometer band 6 using adaptive spectrum-weighted sparse bayesian dictionary learning”. In: *IEEE Trans. Geosci. Remote Sens.* 52.2 (Feb. 2014), pp. 894–906. ISSN: 1558-0644. DOI: 10.1109/TGRS.2013.2245509.

[79] D. Gangopadhyay et al. “Compressed sensing analog front-end for bio-sensor applications”. In: *IEEE J. Solid-State Circuits* 49.2 (Feb. 2014), pp. 426–438. ISSN: 0018-9200. DOI: 10.1109/JSSC.2013.2284673.

[80] A. M. R. Dixon et al. “Compressed sensing system considerations for ECG and EMG wireless biosensors”. In: *IEEE Trans. Biomed. Circuits Syst.* 6.2 (Apr. 2012), pp. 156–166. ISSN: 1932-4545. DOI: 10.1109/TBCAS.2012.2193668.

[81] Mahsa Shoaran et al. “Compact low-power cortical recording architecture for compressive multichannel data acquisition”. In: *IEEE Trans. Biomed. Circuits Syst.* 8.6 (Dec. 2014), pp. 857–870. ISSN: 1932-4545. DOI: 10.1109/TBCAS.2014.2304582.

[82] Michael Lustig et al. “Compressed sensing MRI”. In: *IEEE Signal Processing Magazine* 25.2 (Mar. 2008), pp. 72–82. DOI: 10.1109/MSP.2007.914728.

[83] ON Jaspan, R Fleysher, and M L Lipton. “Compressed sensing MRI: a review of the clinical literature”. In: *British Journal of Radiology* 88.1056 (2015). ISSN: 1051-2004. DOI: 10.1259/bjr.20150487.

[84] E. J. Candes and T. Tao. “Decoding by linear programming”. In: *IEEE Trans. on Inf. Theory* 51.12 (Dec. 2005), pp. 4203–4215. ISSN: 0018-9448. DOI: 10.1109/TIT.2005.858979.

[85] E. van den Berg and M. P. Friedlander. *SPGL1: A solver for large-scale sparse reconstruction*. 2007.

[86] S. Rangan. “Generalized approximate message passing for estimation with random linear mixing”. In: *2011 IEEE international symposium on information theory proceedings*. July 2011, pp. 2168–2172. DOI: 10.1109/ISIT.2011.6033942.

[87] Joel A Tropp and Anna C Gilbert. “Signal recovery from random measurements via orthogonal matching pursuit”. In: *IEEE Trans. on Inf. Theory* 53.12 (Dec. 2007), pp. 4655–4666. ISSN: 0018-9448. DOI: 10.1109/TIT.2007.909108.

[88] Deanna Needell and Joel A Tropp. “CoSaMP: Iterative signal recovery from incomplete and inaccurate samples”. In: *Applied and Computational Harmonic Analysis* 26.3 (May 2009), pp. 301–321. ISSN: 1063-5203. DOI: 10.1016/j.acha.2008.07.002.

[89] Fabio Pareschi et al. “Energy analysis of decoders for rakeness-based compressed sensing of ECG signals”. In: *IEEE Trans. Biomed. Circuits Syst.* 11.6 (Dec. 2017), pp. 1278–1289. ISSN: 1932-4545. DOI: 10.1109/TBCAS.2017.2740059.

[90] Luisa F Polania et al. “Exploiting prior knowledge in compressed sensing wireless ECG systems”. In: *IEEE J. Biomed. Heal. Informatics* 19.2 (Mar. 2015), pp. 508–519. ISSN: 2168-2194. DOI: 10.1109/JBHI.2014.2325017.

[91] J. Zhang et al. "Energy-efficient ECG compression on wireless biosensors via minimal coherence sensing and weighted ℓ_1 minimization reconstruction". In: *IEEE Journal of Biomedical and Health Informatics* 19.2 (Mar. 2015), pp. 520–528. ISSN: 2168-2194. DOI: 10.1109/JBHI.2014.2312374.

[92] A. Marchioni et al. "Sparse sensing matrix based compressed sensing in low-power ECG sensor nodes". In: *2017 IEEE biomedical circuits and systems conference (BioCAS)*. Oct. 2017, pp. 1–4. DOI: 10.1109/BIOCAS.2017.8325155.

[93] A. Mirrashid and A. A. Beheshti. "Compressed remote sensing by using deep learning". In: *9th international symposium on telecommunications (IST)*. Dec. 2018, pp. 549–552. DOI: 10.1109/ISTEL.2018.8661112.

[94] K. Kulkarni et al. "ReconNet: non-iterative reconstruction of images from compressively sensed measurements". In: *IEEE conference on computer vision and pattern recognition (CVPR)*. June 2016, pp. 449–458. DOI: 10.1109/CVPR.2016.55.

[95] A. Mousavi and R. G. Baraniuk. "Learning to invert: signal recovery via deep convolutional networks". In: *IEEE international conference on acoustics, speech and signal processing (ICASSP)*. Mar. 2017, pp. 2272–2276. DOI: 10.1109/ICASSP.2017.7952561.

[96] W. Shi et al. "Deep networks for compressed image sensing". In: *IEEE international conference on multimedia and expo (ICME)*. July 2017, pp. 877–882. DOI: 10.1109/ICME.2017.8019428.

[97] A. Mousavi, A. B. Patel, and R. G. Baraniuk. "A deep learning approach to structured signal recovery". In: *3rd annual allerton conference on communication, control, and computing (allerton)*. Sept. 2015, pp. 1336–1343. DOI: 10.1109/ALLERTON.2015.7447163.

[98] Michael Iliadis, Leonidas Spinoulas, and Aggelos K. Katsaggelos. "Deep fully-connected networks for video compressive sensing". In: *Digital Signal Processing* 72 (2018), pp. 9–18. ISSN: 1051-2004. DOI: 10.1016/j.dsp.2017.09.010.

[99] J. Zhang and B. Ghanem. "ISTA-Net: interpretable optimization-inspired deep network for image compressive sensing". In: *IEEE/CVF conference on computer vision and pattern recognition*. June 2018, pp. 1828–1837. DOI: 10.1109/CVPR.2018.00196.

[100] B. Sun et al. "A deep learning framework of quantized compressed sensing for wireless neural recording". In: *IEEE Access* 4 (2016), pp. 5169–5178. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2604397.

[101] Amir Beck and Marc Teboulle. "A fast iterative shrinkage-thresholding algorithm for linear inverse problems". In: *SIAM Journal on Imaging Sciences* 2.1 (2009), pp. 183–202. DOI: 10.1137/080716542.

[102] Mauro Mangia et al. "Deep Neural Oracles for Short-window Optimized Compressed Sensing of Biosignals". In: *IEEE Trans. Biomed. Circuits Syst.* 14.3 (June 2020), pp. 545–557. ISSN: 1932-4545. DOI: 10.1109/TBCAS.2020.2982824.

[103] Mauro Mangia et al. “Low-power ECG acquisition by compressed sensing with deep neural oracles”. In: *2020 2nd IEEE international conference on artificial intelligence circuits and systems (AICAS)*. Aug. 2020, pp. 158–162. DOI: 10.1109/AICAS48895.2020.9073945.

[104] L. Prono et al. “Low-power fixed-point compressed sensing decoder with support oracle”. In: *2020 IEEE international symposium on circuits and systems (ISCAS)*. Oct. 2020, pp. 1–5. DOI: 10.1109/ISCAS45731.2020.9180502.

[105] Philippe Bich et al. “Aggressively prunable MAM²-based Deep Neural Oracle for ECG acquisition by Compressed Sensing”. In: *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. Oct. 2022, pp. 163–167. DOI: 10.1109/BioCAS54905.2022.9948676.

[106] Michael Elad. “Optimized projections for compressed sensing”. In: *Signal Processing, IEEE Transactions on* 55.12 (2007). tex.publisher: IEEE, pp. 5695–5702. DOI: 10.1109/TSP.2007.900760.

[107] Jianping Xu, Yiming Pi, and Zongjie Cao. “Optimized projection matrix for compressive sensing”. In: *EURASIP Journal on Advances in Signal Processing* (2010), p. 560349. ISSN: 1687-6180. DOI: 10.1155/2010/560349.

[108] Mauro Mangia, Riccardo Rovatti, and Gianluca Setti. “Rakeness in the design of analog-to-information conversion of sparse and localized signals”. In: *IEEE Trans. Circuits Syst. I, Reg. Papers* 59.5 (May 2012), pp. 1001–1014. ISSN: 1549-8328. DOI: 10.1109/TCSI.2012.2191312.

[109] Mauro Mangia et al. “Rakeness-Based Design of Low-Complexity Compressed Sensing”. In: *IEEE Trans. Circuits Syst. I, Reg. Papers* 64.5 (May 2017). ISSN: 1549-8328. DOI: 10.1109/TCSI.2017.2649572.

[110] Javier Haboba et al. “A Pragmatic Look at Some Compressive Sensing Architectures With Saturation and Quantization”. In: *IEEE J. Emerg. Sel. Topics Circuits Syst.* 2.3 (Sept. 2012). tex.ids= haboba2012pragmatic, pp. 443–459. ISSN: 2156-3357. DOI: 10.1109/JETCAS.2012.2220392.

[111] Mauro Mangia et al. “Adapted compressed sensing: A game worth playing”. In: *IEEE Circuits Syst. Mag.* 20.1 (2020), pp. 40–60. ISSN: 1531-636X. DOI: 10.1109/MCAS.2019.2961727.

[112] Stephane Mallat. *A wavelet tour of signal processing: the sparse way*. Access Online via Elsevier, 2008.

[113] N. Bertoni et al. “Low-power EEG monitor based on compressed sensing with compressed domain noise rejection”. In: *2016 IEEE international symposium on circuits and systems (ISCAS)*. May 2016, pp. 522–525. DOI: 10.1109/ISCAS.2016.7527292.

[114] Nick Yeung et al. “Detection of synchronized oscillations in the electroencephalogram: An evaluation of methods”. In: *Psychophysiology* 41.6 (2004). tex.eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-8986.2004.00239.x, pp. 822–832. DOI: 10.1111/j.1469-8986.2004.00239.x.

[115] Valerio Cambareri et al. “Low-complexity multiclass encryption by compressed sensing”. In: *IEEE Transactions on Signal Processing* 63.9 (May 2015). tex.publisher: IEEE, pp. 2183–2195.

[116] V. Cambareri et al. “On known-plaintext attacks to a compressed sensing-based encryption: A quantitative analysis”. In: *IEEE Transactions on Information Forensics and Security* 10.10 (Oct. 2015), pp. 2182–2195. ISSN: 1556-6013. DOI: 10.1109/TIFS.2015.2450676.

[117] T. Bianchi, V. Bioglio, and E. Magli. “Analysis of one-time random projections for privacy preserving compressed sensing”. In: *IEEE Transactions on Information Forensics and Security* 11.2 (Feb. 2016), pp. 313–327. ISSN: 1556-6013. DOI: 10.1109/TIFS.2015.2493982.

[118] Y. Zhang et al. “A review of compressive sensing in information security field”. In: *IEEE Access* 4 (2016), pp. 2507–2519. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2569421.

[119] J. Zhang et al. “An efficient and compact compressed sensing microsystem for implantable neural recordings”. In: *IEEE Trans. Biomed. Circuits Syst.* 8.4 (Aug. 2014), pp. 485–496. ISSN: 1932-4545. DOI: 10.1109/TBCAS.2013.2284254.

[120] David Bellasi et al. “A broadband multi-mode compressive sensing current sensor SoC in 0.16μm CMOS”. In: *IEEE Trans. Circuits Syst. I, Reg. Papers* 66.1 (Jan. 2019), pp. 105–118. ISSN: 1549-8328. DOI: 10.1109/TCSI.2018.2846573.

[121] D. E. Bellasi et al. “A low-power architecture for punctured compressed sensing and estimation in wireless sensor-nodes”. In: *IEEE Trans. Circuits Syst. I, Reg. Papers* 62.5 (May 2015), pp. 1296–1305. ISSN: 1549-8328. DOI: 10.1109/TCSI.2015.2418833.

[122] Juhwan Yoo et al. “A 100MHz-2GHz 12.5x sub-Nyquist rate receiver in 90nm CMOS”. In: *2012 IEEE Radio Frequency Integrated Circuits Symposium*. June 2012, pp. 31–34. DOI: 10.1109/RFIC.2012.6242225.

[123] Xi Chen et al. “A sub-nyquist rate compressive sensing data acquisition front-end”. In: *IEEE J. Emerg. Sel. Topics Circuits Syst.* 2.3 (Sept. 2012), pp. 542–551. ISSN: 2156-3357. DOI: 10.1109/JETCAS.2012.2221531.

[124] Carmine Paolino et al. “A Practical Architecture for SAR-based ADCs with Embedded Compressed Sensing Capabilities”. In: *15th conference on Ph.D. research in microelectronics and electronics (PRIME)*. July 2019, pp. 133–136. DOI: 10.1109/PRIME.2019.8787816.

[125] Martín Abadi et al. *TensorFlow: large-scale machine learning on heterogeneous systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[126] François Chollet et al. "Keras". In: (2015). URL: https://keras.io.

[127] A. L. Goldberger et al. "Physiobank, Physiotoolkit, and Physionet: components of a new research resource for complex physiologic signals". In: *Circulation* 101.23 (June 2000), pp. 215–220. DOI: 10.1161/01.cir.101.23.e215.

[128] Ary L. Goldberger et al. "Physiobank, Physiotoolkit, and Physionet: components of a new research resource for complex physiologic signals". In: *Circulation* 101.23 (2000), pp. 215–220.

[129] G.B. Moody, R.G. Mark, and A.L. Goldberger. "Evaluation of the 'TRIM' ECG data compressor". In: *Proceedings. Computers in Cardiology 1988*. Sept. 1988, pp. 167–170. DOI: 10.1109/CIC.1988.72591.

[130] Y. Zigei, A. Cohen, and A. Katz. "The weighted diagnostic distortion (WDD) measure for ECG signal compression". In: *IEEE Trans. on Biom. Eng.* 47.11 (Nov. 2000), pp. 1422–1430. ISSN: 0018-9294. DOI: 10.1109/TBME.2000.880093.

[131] Ewout van den Berg and Michael P Friedlander. *SPGL1: A solver for large-scale sparse reconstruction*. Dec. 2019. URL: https://friedlander.io/spgl1.

[132] Jesse Read et al. "Classifier chains for multi-label classification". In: *Machine Learning* 85.3 (June 2011), p. 333. ISSN: 1573-0565. DOI: 10.1007/s10994-011-5256-5.

[133] P. Salmela et al. "Several approaches to fixed-point implementation of matrix inversion". In: *International Symposium on Signals, Circuits and Systems, 2005. ISSCS 2005.* Vol. 2. July 2005, 497–500 Vol. 2. DOI: 10.1109/ISSCS.2005.1511286.

[134] Ramanarayan Mohanty et al. "Design and Performance Analysis of Fixed-point Jacobi SVD Algorithm on Reconfigurable System". en. In: *IERI Procedia*. International Conference on Applied Computing, Computer Science, and Computer Engineering (ICACC 2013) 7 (Jan. 2014), pp. 21–27. ISSN: 2212-6678. DOI: 10.1016/j.ieri.2014.08.005. (Visited on 10/29/2022).

[135] Bibek Kabi. "ANALYSIS OF FIXED-POINT SINGULAR VALUE DECOMPOSITION ALGORITHMS". PhD thesis. ndian Institute of Technology, Kharagpur, Aug. 2018. DOI: 10.13140/RG.2.2.11920.25600.

[136] Tapan Pradhan, Bibek Kabi, and Aurobinda Routray. "Fixed-Point Hestenes Algorithm for Singular Value Decomposition of Symmetric Matrices". en. In: (2013), p. 9.

[137] Bernard Widrow and Marcian E Hoff. "Adaptive switching circuits". In: *IRE WESCON conv. Rec.* Vol. 4. 1960, pp. 96–104.

[138] Simon O. Haykin. *Adaptive filter theory*. 5th Edition. Ontario, Canada: Pearson, 2014. ISBN: 978-0-13-267149-1.