ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

IBM

Department of Information Technology and
Electrical Engineering

Spring Semester 2021

Resonator Networks with Sparse Codes to Reduce Parameters in Deep Neural Networks

Bachelor Project

Angéline N. Pouget
apouget@ethz.ch

June 2021

Supervisors: Dr. Abbas Rahimi, abr@zurich.ibm.com
Michael Hersche, herschmi@iis.ee.ethz.ch
Professor: Prof. Dr. Luca Benini, lbenini@iis.ee.ethz.ch

Acknowledgements

I would like to thank my supervisors Dr. Abbas Rahimi and Michael Hersche for their continuous support and their valuable inputs. From initial explanations about the workings of resonator networks to numerous discussions of possible optimization approaches to useful tips regarding the presentation and this thesis, you always provided necessary guidance and enabled the steep learning curve I have been able to experience.

Abstract

High-dimensional computing is an emerging computing approach rooted in neuroscience and inspired by the understanding that the human brain computes with patterns not representable by numbers. Operating on high-dimensional vectors, it exploits their quasi-orthogonality and robustness to store, manipulate and reproduce information. Sparse high-dimensional vectors are of particular interest since they exhibit the same properties as their dense counterparts while at the same time being more memory and computation efficient.

Despite their importance, there so far exists only a small number of relatively little explored operations that can be performed on sparse vectors without reducing their sparsity. Moreover, there is no known model that reliably and efficiently factorizes a product of multiple sparse vectors to uncover its components. Said network could be used in an abundance of tasks like for instance the mapping of feature vectors to class labels in convolutional neural networks such as MobileNet-V2.

We introduce a novel weighted superposition operation designed specifically for sparse high-dimensional vectors as well as a randomized activation sparsification. This allows us to build a sparse resonator network that succeeds in solving sparse vector factorization problems with an accuracy of 100%. It does so across a large variety of sparsity levels and outperforms its dense counterpart in terms of runtime especially for problem sizes larger than 10^9. In this scale, the sparse network already converges more than an order of magnitude faster. We then reduce the parameter count of the MobileNet-V2 classifier by 99.79% through replacing its fully connected layer with a sparse resonator network. We achieve this by the use of learnable pooling with shared weights and a segment-wise spherical loss function that we propose.

Declaration of Originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor. For a detailed version of the declaration of originality, please refer to Appendix B.

Angéline N. Pouget,
Zurich, June 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial Neural Networks (ANNs) and especially Convolutional Neural Networks (CNNs), are employed for a wide variety of tasks and have surpassed human expert performance in various areas [3]. Though loosely based on information processing in biological systems such as the human brain, most of these models fail to support rule-based reasoning, a crucial component of human cognition. In addition, they rely heavily on biologically implausible processes such as for instance back-propagation [4]. High-dimensional computing is a computation approach rooted in neuroscience that aims to target these shortcomings by combining high-dimensional vectors with corresponding operations that together form an algebra. Information stored in randomly drawn high-dimensional quasi-orthogonal vectors is more robust to local alterations since it is represented redundantly and holistically instead of being translated to bits and numbers [5]. Along with other insights, this observation has led to the development of a resonator network which is essentially a type of recurrent neural network that efficiently factorizes high-dimensional product vectors [1].

This network is however based on dense high-dimensional vectors despite their sparse equivalents exhibiting the same remarkable properties while being more memory and computation efficient. In addition, neurobiology research has suggested that primate brains also rely on sparse representations of information and hence a resonator network based on sparse vectors would better approximate the brain's functioning [6]. One important reason why this has so far not been attempted is that there are only few relatively little explored arithmetic operations on vectors with only a small fraction of non-zeros [7]. In order to be able to solve factorization problems also for sparse binary high-dimensional vectors, we first present a number of novel operations such as weighted bundling or attention sparsification that can be performed on them without reducing their sparsity. We then make use of these operations to build a sparse resonator network. When iterating through search spaces larger than 10^9 in trying to find the underlying factors of

the product vector, our proposed solution significantly outperforms its dense counterpart with the average number of iterations being more than an order of magnitude lower.

There is a large number of possible applications of this novel network with an important one being the replacement of fully-connected layers in CNNs used for image classification. This affine transformation at the end of such a model can have a vast number of parameters growing linearly with the number of possible classes, thus requiring increasingly more resources for storage and computations. It has been shown that by sparsifying or even fixing this layer during training, one can gain significant memory and computational benefits while incurring only marginal or no loss of accuracy [8]. Recent discoveries even suggest that for some applications these layers can be removed entirely without it having impairing effects on the network's performance [9]. We now propose to replace the fully connected layer of a CNN by a resonator network with a fixed set of basis vectors. By exploring different interfaces between the feature extractor and the resonator network, various loss functions as well as different ways of initializing the resonator network, we manage to significantly reduce the number of network parameters at only a small drop in accuracy. Moreover, we introduce novel ways of mapping the feature vector to maximally sparse target vectors representing the corresponding classes. These can also be used in classification problems with randomly drawn target vectors since such methods have been attracting increasingly more interest over the past year [10].

The rest of this thesis is organized as follows. Chapter 2 dives into the work that has already been done in this area and lays the groundwork for what will be explained afterwards. Chapter 3 focuses on our main contribution, the sparse resonator network, and discusses its performance especially in comparison to the dense variant. We detail how this network can be used to replace the fully-connected CNN layers of MobileNet-V2 in Chapter 4. In Chapter 5 we draw conclusions from what we achieved and suggest further lines of research that could be interesting to delve into.

Chapter 2

Background and Preliminaries

2.1. High-Dimensional Computing with Sparse Vectors

2.1.1. High-Dimensional Computing

Connectionism is an approach in the field of cognitive science that aims to understand the brain's intellectual abilities by using artificial neural networks. The most widespread and well-known network used in connectionist modelling is the multilayer perceptron which relies heavily on processes such as supervised error correction and back-propagation that are biologically implausible [11]. *High-dimensional* or *hyperdimensional computing* describes more probable cognitive models that do not rely on such processes. They instead depend on very high-dimensional vectors and their manipulation by operations which produce new high-dimensional vectors. These models exploit certain properties of high-dimensional spaces such as redundant encoding of information, holistic representations and the quasi-orthogonality of randomly chosen vectors. Said characteristics lead to increased robustness and tolerance to error because information is distributed and hence the meaning of a data item is represented not by a single bit but by a population of bits [5]. *Vector Symbolic Architectures (VSAs)* are an example of such high-dimensional computing models that make use of the algebraic properties of the underlying vector representations [4]. VSAs allow one to express data structures holographically in a vector space of high but fixed dimensionality. The building blocks are random high-dimensional vectors and data structures built from them are typically vectors with the same dimension.

There is however a drawback coming with these robust and biologically plausible models, namely that computing with high-dimensional vectors can increase the space and computation complexity of a model. Hence it is advantageous with regard to both energy efficiency and memory capacity to use not dense but sparse binary vectors [7].

2.1.2. Arithmetic Operations on High-Dimensional Sparse Vectors

In order to compute with sparse binary high-dimensional vectors, we need to first define the necessary arithmetic operations.

Maximally Sparse Vectors

As proposed in [7], one can divide a binary sparse vector $\boldsymbol{a}$ of length D into k L-bit segments (with $D = kL$). In each of these segments, only one single bit location is set to 1 and such a vector is also called *maximally sparse*. Given two segmented random vectors with sufficiently large parameters k and L, the inner product between these vectors is close to zero meaning that the vectors do not resemble each other. Instead of representing $\boldsymbol{a}$ as a D-dimensional vector where the majority of entries is zero, it is more spatially and computationally efficient to use the k-dimensional *offset vector* $\boldsymbol{a'}$ which stores the location of the non-zero bit for each segment and hence has entries in $[0, L-1]$. An example for this representation is shown in Equation 2.1.

$$\boldsymbol{a} = \left(\begin{array}{c} 0 \\ 0 \\ 1 \\ \hline 1 \\ 0 \\ 0 \end{array}\right) \rightarrow \boldsymbol{a}' = \left(\begin{array}{c} 2 \\ \hline 0 \end{array}\right) \qquad \text{with } D = 6,\ k = 2,\ L = 3 \tag{2.1}$$

Binding and Bundling Operations

The *binding operation* $\otimes$ takes two vectors as inputs and outputs a vector that resembles none of the inputs. For two maximally sparse binary random vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ it is defined as the segment-wise cyclic shift of the elements in one vector (for example $\boldsymbol{b}$) where the number of shifts in each segment is defined by the location of the set bit in the other vector (in this case $\boldsymbol{a}$). An example for this procedure with two input vectors is shown in Equation 2.2.

$$\boldsymbol{a} \otimes \boldsymbol{b} = \left(\begin{array}{c} 0 \\ 0 \\ 1 \\ \hline 1 \\ 0 \\ 0 \end{array}\right) \otimes \left(\begin{array}{c} 0 \\ 1 \\ 0 \\ \hline 0 \\ 1 \\ 0 \end{array}\right) = \left(\begin{array}{c} 1 \\ 0 \\ 0 \\ \hline 0 \\ 1 \\ 0 \end{array}\right) = \boldsymbol{e} \tag{2.2}$$

If we use the offset notation for vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ this operation can be written as shown in Equation 2.3.

$$\boldsymbol{a}' \otimes \boldsymbol{b}' = \begin{pmatrix}2\\0\end{pmatrix} \otimes \begin{pmatrix}1\\1\end{pmatrix} = (\boldsymbol{a}' + \boldsymbol{b}') \bmod L = \begin{pmatrix}3\\1\end{pmatrix} \bmod L = \begin{pmatrix}0\\1\end{pmatrix} = \boldsymbol{e}' \tag{2.3}$$

The binding operation commutes for maximally sparse vectors, i.e., $\boldsymbol{a} \otimes \boldsymbol{b} = \boldsymbol{b} \otimes \boldsymbol{a}$. Moreover, the operation preserves distance, left-distributes over addition and is invertible. This method is very robust since erroneous bits or noise in a particular segment only affect the binding result in that segment [5]. The inverse binding $\oslash$ can be written as seen in Equation 2.4. It is important to mention that this operation does not commute in contrast to its dense correspondence, the Hadamard product.

$$\boldsymbol{e}' \oslash \boldsymbol{b}' = \begin{pmatrix}0\\1\end{pmatrix} \oslash \begin{pmatrix}1\\1\end{pmatrix} = (\boldsymbol{e}' - \boldsymbol{b}') \bmod L = \begin{pmatrix}-1\\0\end{pmatrix} \bmod L = \begin{pmatrix}2\\0\end{pmatrix} = \boldsymbol{a}' \tag{2.4}$$

The *bundling operation* denoted by $\oplus$ is based on the subset sum operation introduced in [7] and takes two (or more) maximally sparse vectors as inputs. It computes the element-wise integer sum of these vectors, implying that the resulting vector resembles (in terms of the inner product) all input vectors. In order to restore the desired sparsity level after a bundling operation, we need to perform subsequent *thinning*. We achieve this by performing integer summation on the input vector and consecutively assigning value 1 to the maximizing element of each segment while setting all other entries to 0. If multiple locations have the same value, one of them is chosen at random. The bundling operation for three input vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ including the intermediate stage before the thinning operation is shown in Equation 2.5. Note that in the lower segment, one of the three bits has been chosen at random during the thinning process and hence $\boldsymbol{d}$ is not defined unambiguously which makes the bundling operation irreversible.

$$\boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c} = \begin{pmatrix}0\\0\\1\\\hline 1\\0\\0\end{pmatrix} \oplus \begin{pmatrix}0\\1\\0\\\hline 0\\1\\0\end{pmatrix} \oplus \begin{pmatrix}0\\0\\1\\\hline 0\\0\\1\end{pmatrix} = \begin{pmatrix}0\\1\\2\\\hline 1\\1\\1\end{pmatrix} \xrightarrow{\text{thinning}} \begin{pmatrix}0\\0\\1\\\hline 0\\1\\0\end{pmatrix} = \boldsymbol{d} \tag{2.5}$$

2.2. Dense Resonator Networks

In order to read out the components of a VSA-encoded data structure, the vector encoding must be decomposed into the building blocks it consists of. The *decoding* of a multiplicative composition is essentially a factorization problem [1]. Resonator networks are a type of recurrent neural network that rely on superposition to search through the combinatoric solution space without enumerating all possible factorizations explicitly. This allows them to solve the decoding problem efficiently. Given a high-dimensional

vector as input, a resonator network iteratively searches through many potential factorizations in parallel until a set of factors is found that agrees with the input. Solutions then emerge as stable fixed points in the network dynamic [1].

All entities in a VSA are represented as high-dimensional vectors in the same vector space. The basic building blocks are chosen randomly and the set of vectors representing specific items is stored in a *codebook* which is a matrix of dimension $D \times M$ where D is the vector dimension and M is the number of such building blocks.

We are given a dense bipolar *composite vector* $\boldsymbol{x}$

$$\boldsymbol{x} = \boldsymbol{a}_i \odot \boldsymbol{b}_j \odot \boldsymbol{c}_h \tag{2.6}$$

where $\boldsymbol{a}_i$, $\boldsymbol{b}_j$ and $\boldsymbol{c}_h$ are drawn from randomly generated codebooks $\boldsymbol{A} = \{\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{M-1}\}$, $\boldsymbol{B} = \{\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{M-1}\}$ and $\boldsymbol{C} = \{\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{M-1}\}$ respectively and $\odot$ represents a Hadamard product. The factorization problem now is to find i, j and h without exhaustively iterating through all possible vector combinations. The resonator network can be used in principle to solve this problem for composite vectors based on the binding of any number V of codebook vectors.

Its functionality is based on first estimating possible factors $\hat{\boldsymbol{a}}(0)$, $\hat{\boldsymbol{b}}(0)$ and $\hat{\boldsymbol{c}}(0)$ as the superposition of all possible codebook vectors. Since the Hadamard product is invertible, given for example $\hat{\boldsymbol{b}}(0)$ and $\hat{\boldsymbol{c}}(0)$, the updated version of the third factor, namely $\hat{\boldsymbol{a}}(1)$, can be inferred. This inference process is however noisy and hence the estimate $\hat{\boldsymbol{a}}_n(1)$ has to be cleaned by projecting it onto the span of codebook $\boldsymbol{A}$. This computes a measure of similarity between the estimate and each element in the codebook, quantifying the probability that this element is the sought-after factor. The updated estimate is then formed by the superposition of all codebook items weighted by this similarity metric. This produces a better guess for each of the factors and the inference can be repeated with better estimates which reduces the noise [1].

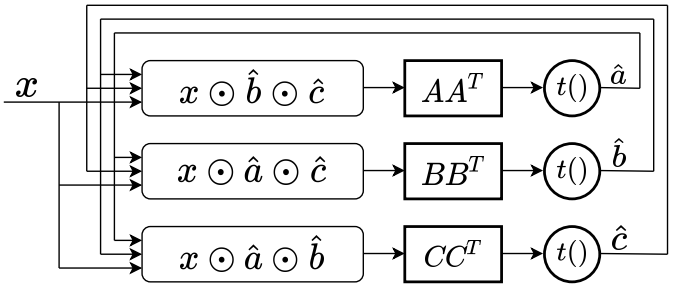


Figure 2.1.: A dense resonator network with $V = 3$ [1].

The dense resonator network is depicted in Figure 2.1 where $t()$ stands for a thresholding function needed to bipolarize the estimates. This is essential since the network is designed for operation on dense bipolar vectors but the weighted superposition results in integer vector elements.

2.3. MobileNet-V2

MobileNet-V2 is an efficient neural network architecture designed specifically for the use in resource-constrained environments such as handheld devices or embedded systems [2]. It is used mainly for image classification, object detection and other computer vision applications.

The model makes use of *depthwise separable convolutions* which replace each convolution operation with a factorized version that splits it into two separate layers. The first of these layers is a *depthwise convolution* which equals a spatial convolution performed independently over each input channel. It is followed by a *pointwise convolution* which is essentially a 1×1 convolution that changes the channel space dimensionality [12]. The replacement of standard convolutional layers by depthwise separable convolution layers in MobileNet-V2 leads to a computational cost that is 8 to 9 times smaller at only a small reduction in accuracy [13]. Additionally, MobileNet-V2 introduces *linear bottleneck layers* which prevent nonlinearities from distorting important information. Lastly, based on the intuition that these bottlenecks contain all the necessary information, shortcuts between the bottlenecks are used. The latter are also called *inverted residuals* and they aim to improve the ability of the gradient to propagate across several layers without leading to large increases in memory requirements [2].

The feature extractor of the MobileNet-V2 architecture is displayed in Table 2.1 where each line describes a sequence of identical layers repeated n times. All spatial convolutions use 3×3 kernels and the first layer of each sequence has stride s whereas all others use stride 1. All layers in one sequence have c output channels [2].

Input	Operator	t	c	n	s	Parameters
$224^2 \times 3$	conv2d	-	32	1	2	992
$112^2 \times 32$	bottleneck	1	16	1	1	992
$112^2 \times 16$	bottleneck	6	24	2	2	11,913
$56^2 \times 24$	bottleneck	6	32	3	2	42,000
$28^2 \times 32$	bottleneck	6	64	4	2	189,760
$14^2 \times 64$	bottleneck	6	96	3	1	309,888
$14^2 \times 96$	bottleneck	6	160	3	2	802,368
$7^2 \times 160$	bottleneck	6	320	1	1	478,400
$7^2 \times 320$	conv2d 1x1	-	1280	1	1	409,600
$7^2 \times 1280$	avgpool 7x7	-	-	1	-	0
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-		$1,280k$

Table 2.1.: MobileNet-V2 Architecture [2].

2.4. Fully Connected Layers in CNNs

The MobileNet-V2 architecture has significantly less parameters than similar well-known CNN models which is due largely to a reduction of the number of parameters in the feature extractor. What has not changed, however, is the classifier and hence the parameters of the fully connected layer account for 37% of the overall parameters for 1000 classes and even more for larger problem sizes [9]. It has been shown that replacing the fully connected layers by a *fixed classifier* leads to significant computational and memory benefits with little or no loss of accuracy, depending on the application [8]. It has even been proposed to remove the classifier entirely which equals replacing the fully connected layers by a fixed identity matrix and has resulted in minor losses in classification performance [9]. The output of the global average pooling layer is in this case used directly to compute classification scores.

2.5. Loss Functions

The *softmax activation function* is used by a majority of neural network models designed to solve multi-class classification problems to map the output of the last fully connected layer to a probabilistic categorical distribution. It is common practice to then apply the *cross-entropy or log loss* to tune the parameters and optimize the performance of these models. CNNs trained with the softmax cross-entropy loss have achieved state-of-the-art performance on an abundance of tasks [14]. For certain applications, however, it is beneficial to take into account spherical loss functions that make use of the *spherical embedding geometry* which changes the similarity structure of the embedding space compared to the standard Euclidean embedding geometry [15]. The reason for this is that the geometry determines the mapping from a deep embedding to a class posterior and hence can be of crucial importance with regard to model performance. In contrast to Euclidean loss functions, spherical loss functions are based only on the similarity between two vectors and not on their magnitude. This can be beneficial if we want to maximize the overlap between any two output and target vectors in our training data as we do not prioritize differently dependent on the target vector magnitude [15].

In this work, we make use of different classification losses that compute the cross-entropy between a predicted class distribution and a one-hot target distribution. They lie in the Euclidean and in the spherical space respectively which is why we will here summarize these two embedding geometries.

2.5.1. Euclidean Embeddings

The well-known dot-product softmax [16] maps to the Euclidean space and has the form

$$p(y|\boldsymbol{z}) = \frac{\exp(\boldsymbol{w}_y^T \boldsymbol{z})}{\sum_j \exp(\boldsymbol{w}_j^T \boldsymbol{z})} \tag{2.7}$$

where $\boldsymbol{z}$ is a deep embedding and $\boldsymbol{w}_j$ are the weights corresponding to class j. The cross-entropy loss based on this function is then given by

$$L(y|\boldsymbol{z}) = -\log(p(y|\boldsymbol{z})) = -\log\left(\frac{\exp(\boldsymbol{w}_y^T \boldsymbol{z})}{\sum_j \exp(\boldsymbol{w}_j^T \boldsymbol{z})}\right) \tag{2.8}$$

and the losses are generally averaged across observations for each minibatch.

2.5.2. Spherical Embeddings

The best-known loss based on spherical embeddings uses cosine distance to compute a probability measure [17]. The *cosine embedding* is defined as

$$p(y|\boldsymbol{z}) = \frac{\exp(s\cos(\theta_y))}{\sum_j \exp(s\cos(\theta_j)} = \frac{\exp\left(s\frac{\boldsymbol{w}_y^T \boldsymbol{z}}{||\boldsymbol{w}_y^T||\cdot||\boldsymbol{z}||}\right)}{\sum_j \exp\left(s\frac{\boldsymbol{w}_j^T \boldsymbol{z}}{||\boldsymbol{w}_j^T||\cdot||\boldsymbol{z}||}\right)} \tag{2.9}$$

where the parameter $s > 0$ is a trainable inverse temperature parameter and θ_j is the angle between $\boldsymbol{z}$ and $\boldsymbol{w}_j$ as is obvious from the second equality. Notice that in comparison to the standard softmax function, only the directions of the weight and embedding vectors are taken into account and not their magnitude.

A second spherical probability measure that is derived from the cosine embedding is the *ArcFace function* [18]. It adds an additive angular margin hyperparameter $m > 0$ that penalizes the true class and is given by

$$p(y|\boldsymbol{z}) = \frac{\exp(s\cos(\theta_y + m))}{\exp(s\cos(\theta_y + m)) + \sum_{j\neq y}\exp(s\cos(\theta_y))}. \tag{2.10}$$

The third and last spherical probability measure introduced here is the *CosFace embedding* which is very similar to the ArcFace embedding but defines the decision margin in cosine space rather than in angle space [19]. With $m > 0$ as before, it can be written as

$$p(y|\boldsymbol{z}) = \frac{\exp(s(\cos(\theta_y) - m))}{\exp(s(\cos(\theta_y) - m)) + \sum_{j\neq y}\exp(s\cos(\theta_y))}. \tag{2.11}$$

The loss function for any of these spherical embeddings is as before given by

$$L(y|\boldsymbol{z}) = -\log(p(y|\boldsymbol{z})) \tag{2.12}$$

which is minimized during training.

Chapter 3

First Contribution: Sparse Resonator

3.1. Functionality of Sparse Resonator Networks

Both sparse and dense resonators aim to efficiently solve the factorization problems in the high-dimensional vector space detailed in Section 2.2. By developing a sparse version which is still closely related to the dense model, we reduce both the spatial and computational complexity of the network while still exploiting the remarkable properties of high-dimensional randomly generated vectors to solve the factorization problem. In order to achieve this, we have to adapt the initial bundling of the three component vectors, the initialization of $\hat{\boldsymbol{a}}$, $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{c}}$ as well as the factorization procedure which is why this section will be split into three parts.

3.1.1. Problem

We are given a D-dimensional composite vector $\boldsymbol{x}$ that is the product of three maximally sparse D-dimensional vectors $\boldsymbol{a}_i$, $\boldsymbol{b}_j$ and $\boldsymbol{c}_h$ with only k nonzero entries each. We can write this as

$$\boldsymbol{x} = \boldsymbol{a}_i \otimes \boldsymbol{b}_j \otimes \boldsymbol{c}_h \tag{3.1}$$

where $\boldsymbol{a}_i$, $\boldsymbol{b}_j$ and $\boldsymbol{c}_h$ are drawn from codebooks $\boldsymbol{A} = \{\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{M-1}\}$, $\boldsymbol{B} = \{\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{M-1}\}$ and $\boldsymbol{C} = \{\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{M-1}\}$ respectively. The problem is to find i, j and h given $\boldsymbol{x}$ and the codebooks $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ by solving the factorization problem.

3.1.2. Initialization

The current estimates for each factor are given by $\hat{\boldsymbol{a}}$, $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{c}}$ and they are initialized to the bundling (including the subsequent thinning) of the vectors in the corresponding

codebook.

$$\hat{\boldsymbol{a}}(0) = \boldsymbol{a}_1 \oplus \cdots \oplus \boldsymbol{a}_M, \quad \hat{\boldsymbol{b}}(0) = \boldsymbol{b}_1 \oplus \cdots \oplus \boldsymbol{b}_M, \quad \hat{\boldsymbol{c}}(0) = \boldsymbol{c}_1 \oplus \cdots \oplus \boldsymbol{c}_M \tag{3.2}$$

3.1.3. Iterative Factorization Procedure

The binding operation is invertible and hence given $\boldsymbol{x}$, and the estimates $\hat{\boldsymbol{b}}(0)$ and $\hat{\boldsymbol{c}}(0)$ for instance, we can infer an updated estimate $\hat{\boldsymbol{a}}_n(1)$.

$$\hat{\boldsymbol{a}}_n(1) = \boldsymbol{x} \oslash \hat{\boldsymbol{b}}(0) \oslash \hat{\boldsymbol{c}}(0) \tag{3.3}$$

However, this estimate has to be cleaned because it can be very noisy and does not necessarily lie in the span of codebook $\boldsymbol{A}$. The reason for this is that $\hat{\boldsymbol{b}}(0)$ and $\hat{\boldsymbol{c}}(0)$ are given by the superposition of M codebook vectors each and they therefore represent all possible combinations of these vectors at the same time. This is on the one hand very powerful as it allows us to quickly converge towards more probable estimations but it also introduces a lot of noise since only one of these combinations is the one that was initially used to create $\boldsymbol{x}$. We achieve the denoising by computing the *overlap* (inner product) between the noisy estimate $\hat{\boldsymbol{a}}_n(1)$ and the codebook matrix $\boldsymbol{A}$ which gives us a weight vector $\boldsymbol{w}'_{\boldsymbol{A}}$. This weight vector which is also called *attention vector* is then sparsified to focus on solutions that seem more probable in the current state of the resonator. As can be seen in Equation 3.4, we add a vector $\boldsymbol{r} \in \mathbb{R}^D_{[0,1)}$ to $\boldsymbol{w}'_{\boldsymbol{A}}$ before the sparsification. This randomization is very important because the vector $\boldsymbol{w}'_{\boldsymbol{A}}$ is an integer vector and hence the sparsification is ambiguous if there are multiple entries with the same value. This is not taken into account if we use a standard `topk` operation without first adding $\boldsymbol{r}$. We then sparsify the weight vector by preserving the a largest elements in $\boldsymbol{w}'_{\boldsymbol{A}} + \boldsymbol{r}$, whereas all the other entries are set to zero. The choice of the *attention vector sparsification constant* a depends on the problem dimension D and the codebook vector vector sparsity constant k. This attention sparsification prevents codebook vectors that are very dissimilar to $\hat{\boldsymbol{a}}_n(1)$ from introducing additional noise.

$$\boldsymbol{A}^T \hat{\boldsymbol{a}}_n(1) + r = \boldsymbol{w}'_{\boldsymbol{A}} + \boldsymbol{r} \quad \xrightarrow{\text{sparsify}} \quad \boldsymbol{w}_{\boldsymbol{A}} \in \mathbb{R}^M \tag{3.4}$$

Given these attentions $\boldsymbol{w}_{\boldsymbol{A}}$, we now perform *weighted bundling* over the codebook vectors with subsequent thinning which yields the denoised prediction $\hat{\boldsymbol{a}}(1)$ as shown in Equation 3.5. The use of weighted bundling instead of the normal bundling introduced in Section 2.1.2 allows us to emphasize certain vectors in the bundling and give less weight to others. This influences the random selection during the thinning process. We can therefore influence how similar the resulting vector is to a component vector by assigning that component vector a smaller or larger weight.

$$\boldsymbol{w}_{\boldsymbol{A}}[0]\boldsymbol{a}_0 \oplus \cdots \oplus \boldsymbol{w}'_{\boldsymbol{A}}[M-1]\boldsymbol{a}_{M-1} \quad \xrightarrow{\text{thinning}} \quad \hat{\boldsymbol{a}}(1) \tag{3.5}$$

This procedure can then be repeated with the updated estimates $\hat{\boldsymbol{a}}(1)$, $\hat{\boldsymbol{b}}(1)$ and $\hat{\boldsymbol{c}}(1)$ until the estimates converge to single codebook vectors. The entire iterative factorization procedure is also shown in Figure 3.1 where $\oplus$ stands for the weighted bundling with subsequent thinning.

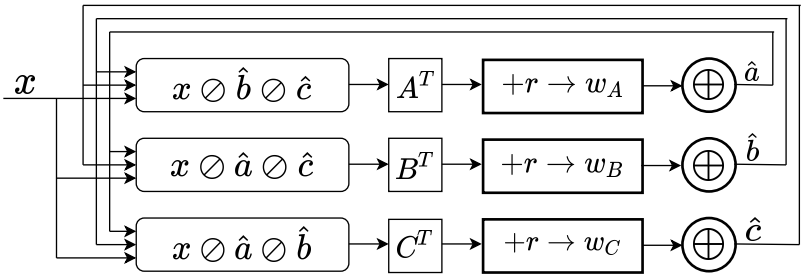


Figure 3.1.: A sparse resonator network with $V = 3$.

3.2. Implementation

The sparse resonator network was implemented using the deep learning library PyTorch [20]. All operations are performed on offset vectors which reduces the runtime by more than an order of magnitude. The only operation that cannot be executed on offset vectors is the weighted bundling over the codebook vectors. However, this has no significant influence on the runtime since the materialized codebook only needs to be computed and stored once, during initialization. This reduces the bundling to a simple matrix multiplication with subsequent thinning; the resulting vector is transformed back to offset notation using an `argmax` operation.

The iterative factorization procedure is repeated until the estimations converge to a fixed point where the states do not change anymore between two subsequent iterations.

$$\hat{\boldsymbol{a}}(t) = \hat{\boldsymbol{a}}(t-1) \wedge \hat{\boldsymbol{b}}(t) = \hat{\boldsymbol{b}}(t-1) \wedge \hat{\boldsymbol{c}}(t) = \hat{\boldsymbol{c}}(t-1) \tag{3.6}$$

In some cases this scenario is never reached which is why the number of iterations is limited by a problem size dependent maximum [21].

$$max\_it = \max(2000, \frac{M^V}{1000}) \tag{3.7}$$

The resonator network factorization procedure is executed on GPUs in batches of size 256.

3.3. Results

The sparse resonator network efficiently factorizes high-dimensional maximally sparse vectors. The attention vectors successfully converge to one-hot vectors indicating the correct indices i, j and h of the codebook vectors underlying $\boldsymbol{x}$. The resonator's performance depends on the choice of D, k and a which will be explored in the following. In the last part, we will compare the performance of the sparse resonator network to the performance of the dense resonator network.

3.3.1. Attention Vector Convergence

As is the case for resonator networks in general, the system seems to cluelessly look for the right vectors until seemingly out of nowhere, the correct solution appears. This aimless search as well as the sudden convergence can be observed by examining the attention vectors $\boldsymbol{w_A}$, $\boldsymbol{w_B}$ and $\boldsymbol{w_C}$ as displayed in Figure 3.2. In this case, the correct solution therefore is $i = 6$, $j = 9$ and $h = 8$. This solution then resonates with the network dynamics which reinforces the correct choice.

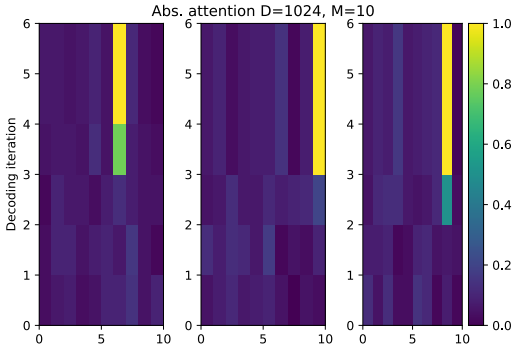


Figure 3.2.: The attention vectors of a decoding with $V = 3$ and $M = 10$.

There is no mathematical guarantee for this convergence but given input parameters and problem sizes that are within the resonator's operational capacity, the network empirically finds the right solution. This convergence can also be seen in Figure 3.3. During iteration 0, all attention vector entries are approximately 0 because the predictions are still very inaccurate and hence the decoded vectors $\hat{\boldsymbol{a}}(0)$, $\hat{\boldsymbol{b}}(0)$ and $\hat{\boldsymbol{c}}(0)$ do not yet resemble any of the codebook vectors (meaning that these predictions are approximately orthogonal to all codebook vectors). This leads to both the mean and the standard deviation being very small. As soon as the convergence takes place, both the mean and the standard deviation increase because one entry is now approximately 1 whereas all others are still close to 0.

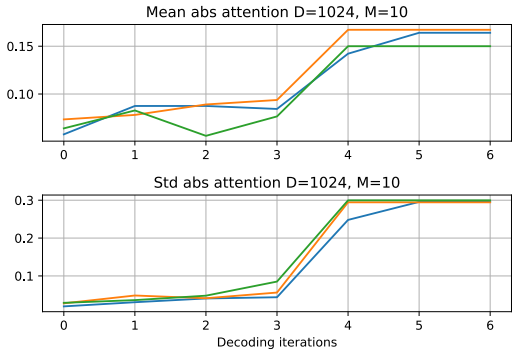


Figure 3.3.: The mean and the standard deviation of the three attention vectors.

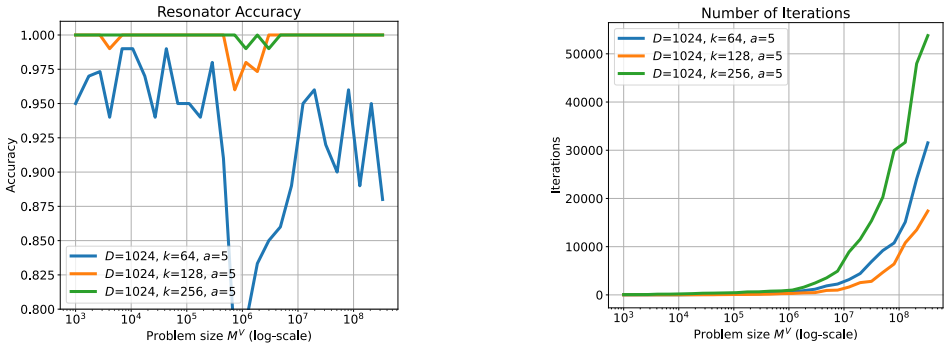
3.3.2. Sparse Resonator Decoding Performance

The decoding accuracy and runtime of the resonator network are highly dependent on the choice of input parameters, especially on the choice of vector dimension D, vector sparsity k and the attention vector sparsification constant a. Given a suitable choice of these three parameters, however, the decoding accuracy stays consistently at 1.0 even for problem sizes $M^V > 10^9$. We were not able to determine at what problem size the accuracy starts to decrease due to computational limitations. It is important to mention that the drop occuring at problem size 10^6 which can be observed in nearly all following diagrams is due to the maximum number of iterations being too low for that region. Hence this does not represent a failure of the sparse resonator network and could be overcome easily by increasing the maximum number of iterations.

Dependence of the Resonator Performance on D and k

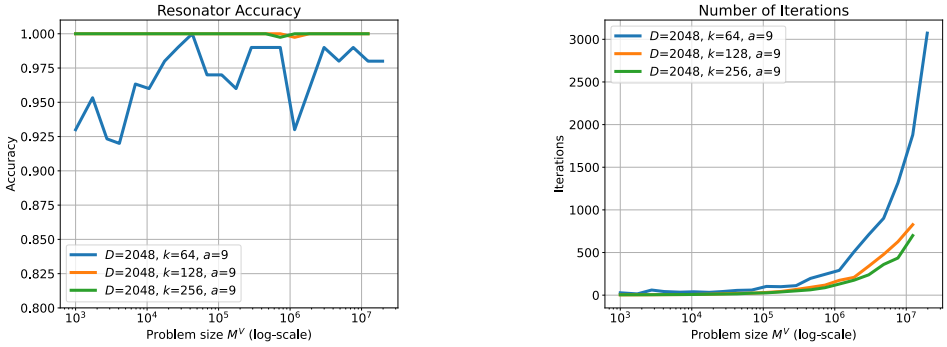
As has been mentioned before and is clearly visible in Figure 3.4 and Figure 3.5, the resonator accuracy is highly dependent on the choice of k given certain parameters D and a. The lower the parameter k, the more information is lost during the thinning processes and hence it becomes more difficult for the network to solve the factorization problem. The reason for this is that for a large parameter k, information is stored redundantly and hence even if the estimation for a small number of the k segments might be wrong, the network is able to find the right decoding by relying on the information stored in the remaining segments. Hence the attention vector indicates the right direction more clearly which leads to faster and more precise convergence. This phenomenon is especially visible in Figure 3.4a in the domain where $M^V = 10^6$ because there the number of iterations is too low and we can see that for smaller k this has a much more pronounced effect on the accuracy than for larger k.

This does however not necessarily mean that larger k lead to faster convergence for all dimensions D and across all problem sizes as can be seen very clearly in Figure 3.4b and



(a) Resonator Accuracy (b) Number of Iterations

Figure 3.4.: Sparse resonator performance with varying sparsity levels k for $D = 1024$.



(a) Resonator Accuracy (b) Number of Iterations

Figure 3.5.: Sparse resonator performance with varying sparsity levels k for $D = 2048$.

Figure 3.5b. It appears that for larger vector dimensions like for example $D = 2048$, less sparse codebook vectors lead to a faster convergence, whereas the opposite holds true for smaller dimensions such as $D = 1024$. This is also supported by the results discussed in Section 3.3.3 which are based on a vector dimension $D = 512$. For large problem sizes and $D = 1024$ we should hence choose the smallest k that still leads to a decoding accuracy of 1.0 to minimize the number of iterations whereas for $D = 2048$ it is better to choose a larger parameter k. To summarize, there is a trade-off between the desired sparsity level, the resonator performance in terms of decoding accuracy and the resonator performance with regard to runtime.

In general, the accuracy is higher and the number of iterations is lower across all problem sizes for $D = 2048$ when compared to $D = 1024$. This makes sense intuitively, since there is more redundancy for $D = 2048$ and hence information can still be recovered correctly even when a larger number of segments have been permuted.

Dependence of the Resonator Performance on a

The performance is also highly dependent on the attention vector sparsification constant a as can be seen in Figure 3.6. This is most explicitly visible when looking at the pink line which represents the resonator accuracy if the attention vector is not sparsified at all. Setting a too low or too high results in a drop of accuracy which is observed across a range

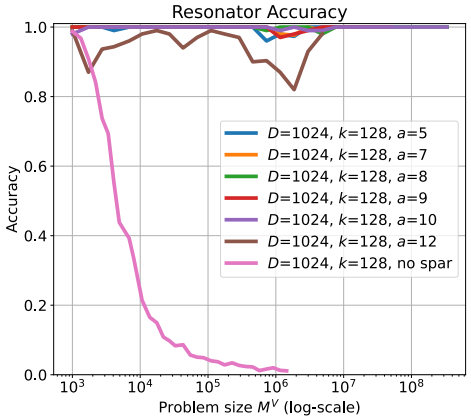


Figure 3.6.: Sparse resonator accuracy with varying attention sparsifications a.

of different combinations of D and k. Hence we have to carefully fine-tune this number for a given choice of the other parameters to optimize the network performance. The reason for this phenomenon might be that if a is too low, we might accidentally filter out the right component during the attention sparsification and if a is too high, very dissimilar codebook vectors are able to re-introduce a certain amount of noise during the weighted bundling. In contrast to the choice of k, the selection of a has no influence on the number of iterations and hence can be chosen independently to optimize the decoding accuracy.

3.3.3. Comparison with the Dense Resonator

By sparsifying the resonator network we were able to decrease its memory usage but we are also interested in the influence this sparsification has on factors such as the average number of iterations which correlates linearly with the runtime and the computational complexity. When comparing the sparse resonator network to its dense counterpart, it can be observed that especially for problem sizes $M^V > 10^7$, the sparse resonator network converges notably faster. This difference in the number of iterations becomes even larger as the problem size increases and for $M^V = 10^9$ the sparse resonator network is already an order of magnitude faster than the dense version as can be seen in Figure 3.7. For smaller problem sizes on the other hand, the sparse resonator network is slightly less

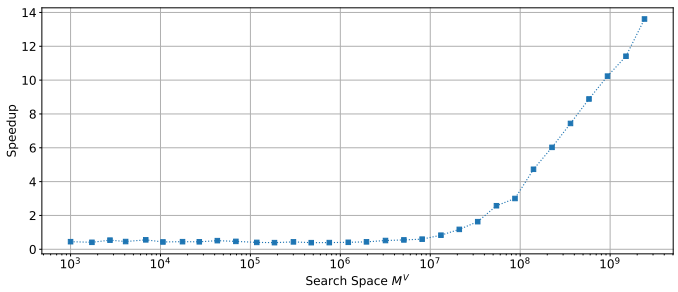


Figure 3.7.: The speedup achieved by using a sparse resonator for $D = 512$ and $k = 64$.

efficient with approximately twice the number of iterations until convergence is reached. This is however of less importance as convergence is very fast in these domains for both the sparse and the dense resonator as clearly shown in Figure 3.4b and Figure 3.5b. For larger problem sizes on the other hand, the number of iterations increases exponentially and hence using a sparse resonator network to solve these problems can have a significant impact on the algorithm runtime.

Given a large enough maximum number of iterations, the accuracy stays consistently at 1.0 for both networks in the examined problem size range and does therefore not differentiate the two versions.

Chapter 4

Second Contribution: Replacement of Fully-Connected CNN Layers

As the title of this thesis already implies, sparse resonator networks can be used to reduce parameters of deep neural networks by replacing the fully-connected layers of CNNs. These layers map the feature vector $\boldsymbol{x}$ shown in Figure 4.1 to the correct class label. In the case of MobileNet-V2, the fully connected layer has $1.28 \cdot 10^6$ parameters and is therefore very memory-intensive. We aim to solve the task of mapping $\boldsymbol{x}$ to a class label $y \in [0, 999]$ by using a resonator network which hence increases the space efficiency of the model without reducing the accuracy significantly. As will be elaborated on below, we explored several different interfaces between the resonator network and the convolutional layers of MobileNet-V2 as well as multiple loss functions and an array of sparsity constants k. All models were trained, tested and evaluated within the PyTorch framework [20] on the ImageNet-1000 dataset [22].

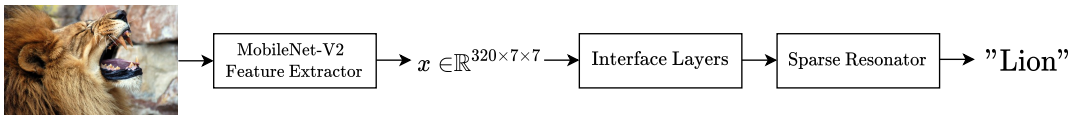


Figure 4.1.: The task is to find suitable interface layers and resonator configurations.

4.1. Interface

4.1.1. Problem Transformation

To be able to replace the classification layers by a resonator network, we must first transform the underlying problem from mapping a feature vector $\boldsymbol{x}$ to a label $y \in \mathbb{N}_{[0,999]}$

to solving a factorization problem. This can be done in multiple ways of which we will in the following explain two.

Resonator Network with Three Factors

We can transform the problem by defining a resonator network with $V = 3$ and $M = 10$ which is able to efficiently and correctly factorize exactly $M^V = 1000$ different combinations of codebook vectors. Assuming that our codebooks are again denoted by $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$, we are given an input vector $\boldsymbol{y'} = \boldsymbol{a}_i \otimes \boldsymbol{b}_j \otimes \boldsymbol{c}_h$ and we aim to find i, j and h which is exactly the problem defined in Section 3.1.1 and solved by our sparse resonator network.

We can translate a class label y to three indices (i, j, h) by looking at all three digits of y separately and mapping them to i, j and h respectively.

$$i = \lfloor y/100 \rfloor \tag{4.1}$$
$$j = \lfloor (y \bmod 100)/10 \rfloor \tag{4.2}$$
$$h = y \bmod 10 \tag{4.3}$$

This transformation is invertible with the inversion being the following.

$$y = i * 100 + j * 10 + h \tag{4.4}$$

The task is now to find an efficient mapping from $\boldsymbol{x} \in \mathbb{R}^{320\times7\times7}$ to the offset vectors representing the suitable resonator input $\boldsymbol{y'} = \boldsymbol{a}_i \otimes \boldsymbol{b}_j \otimes \boldsymbol{c}_h \in \mathbb{N}^k_{[0,L)}$. This vector can naturally also be represented as a maximally sparse vector $\boldsymbol{y} \in \mathbb{N}^D_{[0,1]}$. By mapping the output $\boldsymbol{x}$ to a prediction $\boldsymbol{y'}$ that can then be accurately factorized by the resonator network, we replace the fully connected layer and hence reduce the parameter count of the neural network by more than 1/3. We have limited ourselves to $D = 1280$ and hence, given a sparsity constant k, L is defined unambiguously.

Resonator Network with Two Factors

Alternatively, we can define a network with $V = 2$ and $M = 32$ and 24 null labels (due to the $M^V = 1024$ possible combinations). Given $\boldsymbol{y'} = \boldsymbol{a}_i \otimes \boldsymbol{b}_j$, the problem is now to find i and j with $i, j \in [0, 32)$.

We can transform y to two indices (i, j) for the network with $V = 2$ as follows.

$$i = \lfloor y/32 \rfloor \tag{4.5}$$
$$j = y \bmod 32 \tag{4.6}$$

This transformation is naturally invertible as well.

$$y = i * 32 + j \tag{4.7}$$

The task is in this case to find an efficient mapping from $\boldsymbol{x} \in \mathbb{R}^{320\times7\times7}$ to the offset vectors representing the correct resonator input $\boldsymbol{y'} = \boldsymbol{a}_i \otimes \boldsymbol{b}_j \in \mathbb{N}^k_{[0,L)}$.

4.1.2. Mapping Feature Vector to Resonator Input

An intuitive way of mapping $\boldsymbol{x}$ to $\boldsymbol{z}$, which should then be trained to equal $\boldsymbol{y}$, is shown in Figure 4.2. The dimension reduction through pooling can be achieved for example by applying average pooling over the signal x and hence reducing the 7×7 features per channel to 1. A second possibility is to perform weighted pooling with a weight vector $\boldsymbol{w}_p \in \mathbb{R}^{49}$ shared across all channels. This leads to a better performance at a negligible increase in the number of parameters which will be discussed more in depth in Section 4.4. More sophisticated solutions than the one presented here (e.g. weighted pooling where the weights are not shared across all channels) were not examined in detail since they require a larger number of parameters which would defeat the purpose of using a resonator network to reduce the parameter count.

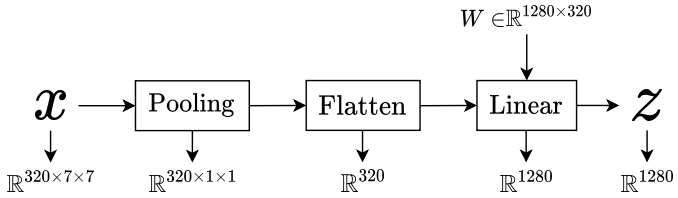


Figure 4.2.: An intuitive way of mapping $\boldsymbol{x}$ to $\boldsymbol{z}$.

4.2. Training

We have tried both different loss functions (cross-entropy loss, binary cross-entropy loss, cosine loss, ArcFace loss, CosFace loss) as well as two different ways of passing the vectors $\boldsymbol{z}$ and $\boldsymbol{y'}$ to these functions which will be referred to as *segment-wise classification* and *overlap classification*. Since the initial models trained with cross-entropy loss or cosine loss have outperformed the final prediction accuracy of those trained with binary cross-entropy loss, ArcFace loss or CosFace loss by at least 20%, the latter have not been explored thoroughly and will therefore not be discussed in depth. All models were trained using stochastic gradient descent with an initial learning rate of 0.04 and Nesterov momentum as described in [23] with a momentum factor of $4{\cdot}10^{-5}$. To adjust the learning rate during training, we used the cosine scheduler proposed in [24]. The batch size was set to 256 and the number of training epochs ranged between 100 and 400.

4.2.1. Segment-Wise Classification

For the segment-wise classification, we transform the problem from training the D-dimensional vector $\boldsymbol{z}$ to equal the correct resonator input vector $\boldsymbol{y}$ to training the k segments of $\boldsymbol{z}$ to correctly indicate which of the L bits should be set to 1 whereas all others are set to 0. This essentially corresponds to splitting the problem into k subproblems where each one of these subproblems is a classification problem with L classes. The correct labels for these k subproblems are stored in the offset vector $\boldsymbol{y}'$. We train this model using either the softmax cross-entropy loss or the cosine loss, both explained in Section 2.5.

4.2.2. Overlap Classification

The overlap classification makes use of the matrix $\boldsymbol{O} \in \mathbb{R}^{1280\times1000}$ which stores the vectors $\boldsymbol{y}$ corresponding to each of the 1000 possible classes. We then maximize the overlap (as measured by the inner product) between vectors $\boldsymbol{z}$ and the correct $\boldsymbol{y}$ by computing the inner products between $\boldsymbol{z}$ and $\boldsymbol{y}$ for all of the 1000 classes, mapping these to probability space and applying either cross-entropy loss or cosine loss. We hence assume that the vector given by $\boldsymbol{O}^T\boldsymbol{z} \in \mathbb{R}^{1000}$ stores the probabilities of the image belonging to each of the 1000 classes which allows us to directly apply one of the mentioned loss functions.

4.3. Inference

During inference, we set the maximizing entry for each of the segments in $\boldsymbol{z}$ to 1 whereas all others are set to 0 which gives us a k-sparse vector that we then input into the resonator. Using the sparse resonator gives us the possibility to fine-tune the model by carefully choosing parameters k and a. To gain a better understanding of the performance of different models, we monitor the segment-wise classification accuracy averaged across all subproblems, the overlap accuracy as well as the resonator accuracy. The overlap accuracy is defined as the top-1 accuracy achieved by assuming that the maximizing entry in $\boldsymbol{O}^T\boldsymbol{z}$ indicates the correct class label. The resonator accuracy is the top-1 accuracy if we consider the class label predicted by the resonator network.

4.4. Results

As a first baseline to compare all our results to we utilize the performance of the identity classifier which is essentially a 1-sparse segment-wise classifier introduced in [9]. This corresponds to not using any classifier at all and simply interpreting $\boldsymbol{z}$ directly as a probability distribution over 1280 classes where 280 of them are null classes. Trained with cosine loss for 100 epochs, this model achieves a top-1 accuracy of 70.8%. The second

baseline is the accuracy of the originally proposed MobileNet-V2 model including the fully connected layer which equals 72.0% [2]. None of our proposed solutions has surpassed either one of these two accuracies, however, the performance of our best models is comparable to them. Moreover, we believe that our additions such as the weighted pooling could help increase the accuracy of these models and are hence valuable independent of whether or not they are used in combination with a sparse resonator network. The same holds for the segment-wise classification which can also be used in connection with randomly drawn sparse target vectors [10]. Lastly, our model is scalable, meaning that the number of classes can be increased significantly without it having an impact on the number of parameters. This is not the case for the other two versions, since the identity classifier can be used for at most 1280 classes and the classifier based on a fully-connected layer would come with 1280 new parameters per additional class.

Unless otherwise indicated, the results below were achieved with the model based on $V = 3$ and $M = 10$.

4.4.1. The Choice of Loss Function

Softmax cross-entropy loss and cosine loss performed superior to all other tested loss functions by a large margin. As shown in Table 4.1, using a spherical embedding is better suited to solve the problem at hand than mapping to Euclidean space. Both models were trained for 100 epochs with $k = 80$ and use standard average pooling. The entries in Table 4.1 denote the top-1 classification accuracies based on the resonator decoding of $\boldsymbol{z}$ (Resonator), the overlap of $\boldsymbol{z}$ with the vectors in matrix $\boldsymbol{O}$ (Overlap) and the averaged segment-wise classification accuracies over the k segments (Segment-wise). Given the correct input $\boldsymbol{y}$, the sparse resonator network used would have been able to find the corresponding class with 100% accuracy.

Model	**Resonator**	**Overlap**	**Segment-wise**
Softmax CEL	63.2%	63.9%	64.5%
Cosine Loss	66.5%	66.7%	68.1%

Table 4.1.: Accuracy dependence on the loss function.

4.4.2. The Impact of Weighted Pooling

Using weighted pooling instead of standard average pooling increases the number of parameters by 49 which is negligible compared to the total number of parameters of a deep neural network. However, it does increase the model accuracy by around 0.3% independent of whether we consider the resonator, the overlap or the segment-wise accuracy as displayed in Table 4.2. This is quite significant given that the network has only been

slightly altered. Both models were trained for 100 epochs with $k = 80$ and cosine loss.

Model	Resonator	Overlap	Segment-wise
Average Pooling	66.5%	66.7%	68.1%
Weighted Pooling	66.8%	67.1%	68.4%

Table 4.2.: Accuracy dependence on the pooling.

4.4.3. Varying the Sparsity Constant k

As discussed previously, the sparsity constant k has to be selected carefully since it has a significant impact on the resonator performance. Table 4.3 shows the results of models with weighted pooling that were trained for 200 epochs with cosine loss. The

Model	Resonator	Overlap	Segment-wise
$k = 40$	60.6%	69.7%	70.1%
$k = 64$	68.3%	69.1%	70.1%
$k = 80$	68.2%	68.6%	70.1%
$k = 128$	67.4%	67.5%	70.3%
$k = 160$	66.4%	66.8%	70.4%

Table 4.3.: Accuracy dependence on choice of sparsity constant k.

performance is optimal for $k = 64$ because for smaller k the resonator does no longer work properly (even with the correct input $\boldsymbol{y}$ the decoding accuracy is not 100% for $k = 40$ and $D = 1280$) and for larger k, the prediction $\boldsymbol{z}$ seems to be less similar to the target vector $\boldsymbol{y}$ as is visible in the declining overlap accuracy.

4.4.4. Increasing the Number of Epochs

For deep neural networks it generally holds true that an increase of the number of epochs and hence the use of more computing power increases the model accuracy. The same can also be observed for our application as is demonstrated in Table 4.4. The model has sparsity $k = 64$, uses weighted pooling and was trained with the segment-wise cosine loss.

Model	Resonator	Overlap	Segment-wise
200 epochs	68.3%	69.1%	70.1%
400 epochs	68.8%	70.1%	71.1%

Table 4.4.: Accuracy dependence on the number of training epochs.

4.4.5. Segment-Wise and Overlap Classification

So far, all considered models have been trained by segment-wise application of the loss function. However, as can be clearly seen in Table 4.5, if we want to maximize the overlap accuracy which is the prediction based on the inner product of $\boldsymbol{z}$ with the vectors in matrix $\boldsymbol{O}$, it is beneficial to already take this into account during training. In contrast to maximizing the resonator accuracy this is possible since it only consists of a matrix multiplication which can of course be inverted during back-propagation. The models were trained for 200 epochs with $k = 64$ and cosine loss. Since overlap classification does not

Model	**Resonator**	**Overlap**	**Segment-wise**
Segment-wise	68.3%	69.1%	70.1%
Overlap	$< 5\%$	70.4%	–

Table 4.5.: Comparison of segment-wise and overlap classification.

perform well in terms of the resonator accuracy, it should only be used either in models that do not rely on the resonator at all or in combination with a second loss function applied segment-wise. This latter option would even introduce a new hyperparameter weighting the different losses which could be carefully tuned to optimize performance.

4.4.6. Lowering the Number of Factors

We have introduced two possible solutions in Section 4.1.1, one with $V = 3$ and $M = 10$ and the other one with $V = 2$ and $M = 32$. The comparison of these two solutions yields that in terms of the resonator accuracy, the model based on $V = 2$ clearly outperforms the other one. This might be due to the fact that this model is still able to correctly factorize problems with 100% accuracy for sparser vectors (e.g. $k = 40$) given the correct input vector. An interesting observation clearly visible in Table 4.6 is that for $V = 3$

Model	**Resonator**	**Overlap**	**Segment-wise**
$V = 3$, $k = 40$	60.6%	69.7%	70.1%
$V = 3$, $k = 64$	68.3%	69.1%	70.1%
$V = 2$, $k = 40$	69.1%	69.5%	69.8%
$V = 2$, $k = 64$	69.0%	69.2%	70.1%

Table 4.6.: Comparison between models with $V = 3$ and $V = 2$.

there are significant differences between the three accuracy measures (they have a range of 9.5% for $k = 40$ and a range of 1.8% for $k = 64$). These are significantly smaller for $V = 2$ (they have a range of 0.7% for $k = 40$ and a range of 1.1% for $k = 64$).

4.4.7. Reduction of the Number of Parameters

Our main goal for this second contribution was to achieve a significant reduction of the number of parameters in the MobileNet-V2 classifier. The fully-connected layer as presented in [2] has $1.28 \cdot 10^6$ weights that are tuned during training and have to be stored thereafter. Our models replaced this layer by either 30 (for $V = 3$) or 64 (for $V = 2$) codebook vectors in addition to 49 weights used for the weighted pooling. In general the codebook vectors have dimension $D = 1280$ each but since we can store them in offset vector notation, they only have dimension k. Hence, the best performing model in terms of the resonator accuracy ($V = 2$ and $k = 40$) has $49 + 40 \cdot 64 = 2609$ parameters. This corresponds to a classifier parameter reduction of 99.79%.

Chapter 5

Conclusion and Future Work

In this thesis, we presented a sparse resonator network that is able to correctly factorize sparse high-dimensional vectors across a large range of problem sizes. This is due mainly to the randomized attention sparsification of the weight vector we developed, which is then used during the bundling process to update our estimates. The analysis of our results has shown that in general, there is a sparsity optimum which maximizes the accuracy while still converging in a reasonably small number of iterations. This could be investigated further by refining the operations such that the resonator network performs well for even sparser vectors, further reducing memory requirements and increasing the network efficiency. In addition, it would be interesting to examine the problem size at which the network accuracy starts to decrease from 1.0, especially in comparison with the dense resonator network. This could yield additional information on a suitable choice of hyperparameters since now several models have an accuracy that stays consistently at 1.0 across all problem sizes and are hence difficult to compare.

We then replaced the fully-connected layers of a CNN by this network to reduce the number of parameters of deep neural networks. In the process, we proposed several ways of mapping feature vectors to maximally sparse target vectors and explored multiple loss functions, detecting that a cosine probability space embedding is best suited to solve the task at hand. As our proposed network has not been able to outperform the network that uses an identity matrix classifier, an obvious topic that would need to be investigated further is whether this really is the optimal solution to the fixed classifier problem. It would also be interesting to repeat our experiments with a resonator network able to deal with even sparser vectors since this seems to benefit from a higher segment-wise classification accuracy.

Other lines of research that could be delved into in the near future are of course the numerous other applications of sparse resonator networks. Given that we have developed

and introduced such a network for the first time, there are now opportunities for sparse high-dimensional computing that were previously not possible.

Appendix A

Task Description

Bachelor Thesis at the Department of
Information Technology and Electrical Engineering

Spring Semester 2021

Angéline Pouget

Resonator Networks with Sparse Codes to Reduce Parameters in Deep Neural Networks

June 2, 2021

Advisors:	Michael Hersche, ETZ J 76.2, herschmi@iis.ee.ethz.ch
	Abbas Rahimi, IBM Research Zurich, abr@zurich.ibm.com
Handout:	March 1, 2021
Due:	June 7, 2021

1 Introduction

Neural networks are commonly used as models for classification for a wide variety of tasks. Typically, an affine transformation is placed at the end of such models, yielding a per-class value used for classification. This classifier can have a vast number of parameters, which grows linearly with the number of possible classes, thus requiring increasingly more resources for storage and computations. The fully-connected layers are still commonly used as classification layers in various neural network architectures, transforming from the dimension of network features D to the number of required class categories C. Therefore, each classification model must hold D $\times$ C number of trainable parameters that grows in a linear manner with the number of classes (i.e., C) [1]. However, it has been shown that the fully-connected classification layer does not to be trained but can be fixed during training. In the extreme case, the fully-connected layer can be as simple as a Hadamard matrix [2].

Randomly-drawn distributed high-dimensional vectors form a quasi-orthogonal basis. It has been shown that a resonator network can efficiently factorize any product vector generated from such basis [3, 4]. As a result, the fully-connected layer of deep neural networks can be replace by a resonator network with a fixed basis. Current resonator networks are designed to operate on dense high-dimensional vectors; however, deep neural networks do not necessarily emit dense but sparse vectors [1].

2 Project Description

In this thesis, you explore resonator networks with high-dimensional sparse codes and their application in deep neural networks.

Task I: Design a resonator with sparse vectors

You start familiarizing with the concept of high-dimensional computing and resonator networks. As a fist step, you re-design the resonator network for k-sparse block codes [5, 6].

- Exploring various attention and summation/thresholding functions.
- Compare the operational capacity of your sparse resonator network to its dense variant.

Task II: Substituting fully connected layer in MobileNetV2 with sparse resonator networks

- Exploring various interfaces to the resonator networks, loss functions, and optimizations.
- Bonus: evaluating the performance and benefits on other networks.

3 Project Realization

3.1 Meetings

Weekly meetings and reports must be held. The exact time and location of these meetings will be determined within the first week of the project in order to fit the student's and the assistant's schedule. These meetings will be used to evaluate the status and progress of the project. Besides these regular meetings, additional meetings can be organized to address urgent issues as well.

3.2 Report

Documentation is an important and often overlooked aspect of engineering. One final report has to be completed within this project. The common language of engineering is de facto English. Therefore, the final report of the work is preferred to be written in English. Any form of word processing software is allowed for writing the reports, nevertheless, the use of LaTeX with Tgif[1] or any other vector drawing software (for block diagrams) is strongly encouraged by the IIS staff.

Final Report The final report has to be presented at the end of the project and a digital copy need to be handed in. Note that this task description is part of your report and has to be attached to your final report.

3.3 Presentation

There will be a presentation (15 min for the semester thesis, and 20 min for the MS thesis presentation followed by 5 min Q&A) at the end of this project in order to present your results to a wider audience. The exact date will be determined towards the end of the work.

[1] See: `http://bourbon.usc.edu:8001/tgif/index.html` and `http://www.dz.ee.ethz.ch/en/information/how-to/drawing-schematics.html`.

References

[1] Z. Qian, T. L. Hayes, K. Kafle, and C. Kanan, "Do we need fully connected output layers in convolutional networks?" *arXiv preprint arXiv:2004.13587*, 2020.

[2] E. Hoffer, I. Hubara, and D. Soudry, "Fix your classifier: the marginal value of training the last weight layer," *arXiv preprint arXiv:1801.04540*, 2018.

[3] E. P. Frady, S. J. Kent, B. A. Olshausen, and F. T. Sommer, "Resonator networks, 1: An efficient solution for factoring high-dimensional, distributed representations of data structures," *Neural computation*, vol. 32, no. 12, pp. 2311–2331, 2020.

[4] S. J. Kent, E. P. Frady, F. T. Sommer, and B. A. Olshausen, "Resonator networks, 2: Factorization performance and capacity compared to optimization-based methods," *Neural computation*, vol. 32, no. 12, pp. 2332–2388, 2020.

[5] M. Laiho, J. H. Poikonen, P. Kanerva, and E. Lehtonen, "High-dimensional computing with sparse vectors," in *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2015, pp. 1–4.

[6] E. P. Frady, D. Kleyko, and F. T. Sommer, "Variable binding for sparse distributed representations: Theory and applications," *arXiv preprint arXiv:2009.06734*, 2020.

Zurich, June 2, 2021 Prof. Dr. Luca Benini

Appendix B

Declaration of Originality

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Resonator Networks with Sparse Codes to Reduce Parameters in Deep Neural Networks

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Pouget	Angéline

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 06.06.2021	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Bibliography

[1] E. P. Frady, S. J. Kent, B. A. Olshausen, and F. T. Sommer, "Resonator networks, 1: An efficient solution for factoring high-dimensional, distributed representations of data structures," *Neural computation*, vol. 32, no. 12, pp. 2311–2331, 2020.

[2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[3] C. Nagpal and S. R. Dubey, "A performance evaluation of convolutional neural networks for face anti spoofing," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[4] R. W. Gayler, "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," *arXiv preprint cs/0412059*, 2004.

[5] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation*, vol. 1, no. 2, pp. 139–159, 2009.

[6] B. A. Olshausen and D. J. Field, "Sparse coding of sensory inputs," *Current opinion in neurobiology*, vol. 14, no. 4, pp. 481–487, 2004.

[7] M. Laiho, J. H. Poikonen, P. Kanerva, and E. Lehtonen, "High-dimensional computing with sparse vectors," in *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2015, pp. 1–4.

[8] E. Hoffer, I. Hubara, and D. Soudry, "Fix your classifier: the marginal value of training the last weight layer," *arXiv preprint arXiv:1801.04540*, 2018.

[9] Z. Qian, T. L. Hayes, K. Kafle, and C. Kanan, "Do we need fully connected output layers in convolutional networks?" *arXiv preprint arXiv:2004.13587*, 2020.

[10] G.-L. Shalev, G. Shalev, and J. Keshet, "On randomized classification layers and their implications in natural language generation," in *Proceedings of the Third Workshop on Multimodal Artificial Intelligence*, 2021, pp. 6–11.

[11] S. D. Levy and R. Gayler, "Vector symbolic architectures: A new building material for artificial general intelligence," in *Proceedings of the 2008 conference on artificial general intelligence 2008: Proceedings of the first AGI conference*, 2008, pp. 414–418.

[12] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[15] T. R. Scott, A. C. Gallagher, and M. C. Mozer, "von mises–fisher loss: An exploration of embedding geometries for supervised learning," *arXiv preprint arXiv:2103.15718*, 2021.

[16] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*. Springer, 1990, pp. 227–236.

[17] A. Zhai and H.-Y. Wu, "Classification is a strong baseline for deep metric learning," *arXiv preprint arXiv:1811.12649*, 2018.

[18] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4690–4699.

[19] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu, "Cosface: Large margin cosine loss for deep face recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5265–5274.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[21] S. J. Kent, E. P. Frady, F. T. Sommer, and B. A. Olshausen, "Resonator networks, 2: Factorization performance and capacity compared to optimization-based methods," *Neural computation*, vol. 32, no. 12, pp. 2332–2388, 2020.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[23] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*. PMLR, 2013, pp. 1139–1147.

[24] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[25] X. Zhang, R. Zhao, Y. Qiao, X. Wang, and H. Li, "Adacos: Adaptively scaling cosine logits for effectively learning deep face representations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 823–10 832.