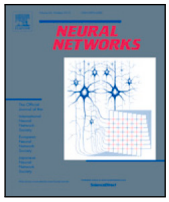
Neural Networks 178 (2024) 106411

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Full Length Article

Compressing neural networks via formal methods

Dalila Ressi [a,b,*], Riccardo Romanello [b], Sabina Rossi [a], Carla Piazza [b]

[a] Ca' Foscari University of Venice, Italy
[b] University of Udine, Italy

ARTICLE INFO

Keywords:
Neural networks
Compression
Pruning
Lumpability

ABSTRACT

Advancements in Neural Networks have led to larger models, challenging implementation on embedded devices with memory, battery, and computational constraints. Consequently, network compression has flourished, offering solutions to reduce operations and parameters. However, many methods rely on heuristics, often requiring re-training for accuracy. Model reduction techniques extend beyond Neural Networks, relevant in Verification and Performance Evaluation fields. This paper bridges widely-used reduction strategies with formal concepts like lumpability, designed for analyzing Markov Chains. We propose a pruning approach based on lumpability, preserving exact behavioral outcomes without data dependence or fine-tuning. Relaxing strict quotienting method definitions enables a formal understanding of common reduction techniques.

1. Introduction

Since 2012, following AlexNet's victory (Krizhevsky, Sutskever, & Hinton, 2012) in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), there has been an exponential surge in proposed *Artificial Neural Network* (*ANN* or *NN*) architectures. Neural networks, with their intrinsic flexibility and exceptional performance, have emerged as the preferred solution for a diverse range of tasks, often synonymous with Artificial Intelligence (AI) in various contexts.

As these models have progressed to handle extensive datasets and complex tasks, their complexity has grown in parallel (Deng, Li, Han, Shi, & Xie, 2020). These intricate networks, characterized by numerous layers, epitomize *Deep Learning* (*DL*) and excel in achieving high accuracy in challenging tasks (Xiao, Bahri, Sohl-Dickstein, Schoenholz, & Pennington, 2018).

While the academic focus has mainly revolved around training extensive and intricate models (Dai, Liu, Le, & Tan, 2021; Kolesnikov et al., 2020; Yu, Wang, Vasudevan, Yeung, Seyedhosseini, & Wu, 2022), integrating such networks into embedded devices presents significant challenges. Real-world limitations such as battery life, memory constraints, and computational capabilities impose restrictions on both the architectural parameter count and the required *Floating Point Operations* (*FLOPs*) during inference.

A commonly used strategy to address this problem is called *Network Compression*. Compression literature has had a substantial growth during the last years, and for this reason, there are many different ways to group together methods reducing a model in similar ways.

Methods focusing on finding the best possible structure to solve particular tasks can be grouped together as *Architecture-related* strategies. These kinds of methods usually require to train the network from scratch each time the structure is modified. In particular, *Neural Architecture Search (NAS)* techniques aim to find the best possible architecture for a certain task with minimal human intervention (Elsken, Metzen, & Hutter, 2019; Liu, Sun, Xue, Zhang, Yen, & Tan, 2021; Ren et al., 2021). This is usually made possible by modeling the search as an optimization problem and applying *Reinforcement Learning (LR)*-based methods to find the best architecture (B. Zoph, 2017; Baker, Gupta, Naik, & Raskar, 2017). In this group, we can also find *Tensor Decomposition* (Novikov, Podoprikhin, Osokin, & Vetrov, 2015), where matrix decomposition/factorization principles are applied to the d-dimensional tensors in neural networks. Tensor decomposition generalizes the widely used Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) to an arbitrary number of dimensions (Carroll & Chang, 1970; Harshman et al., 1970; Tucker, 1966). The goal of these techniques is to reduce the rank of tensors in order to efficiently decompose them into smaller ones and drastically reduce the number of operations (Deng et al., 2020). As the rank of a tensor is usually far from being small, the most common solutions consist in either forcing the network to learn filters with small rank or using an approximated decomposition (Denton, Zaremba, Bruna, LeCun, & Fergus, 2014; Eo, Kang, & Rhee, 2023).

Using a similar approach *Lightweight* or *Compact Networks* focus on modifying the design of the architecture such that it performs fewer operations while maintaining the same capability. It is the case of

* Corresponding author at: University of Udine, Italy.
E-mail address: dalila.ressi@uniud.it (D. Ressi).

https://doi.org/10.1016/j.neunet.2024.106411
Received 7 December 2023; Received in revised form 12 April 2024; Accepted 23 May 2024
Available online 29 May 2024

the MobileNet series (Howard et al., 2019; Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018a, 2018b), ShuffleNet series (Ma, Zhang, Zheng, & Sun, 2018; Zhang, Zhou, Lin, & Sun, 2018), and EfficientNet series (Tan & Le, 2019, 2021). They exploit the idea of using 1×1 filters introduced by Network in Network (Lin, Chen, & Yan, 2014) and GoogLeNet (Szegedy et al., 2015; Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2016) in their inception modules. A similar concept is explored by the SqueezeNet (Iandola, Han, Moskewicz, Ashraf, Dally, & Keutzer, 2016) architecture in their *Fire module*, where they substitute the classical convolutional layers such that they can achieve the same accuracy of AlexNet on ImageNet dataset but with a model 510 times smaller.

An alternative technique exploits a larger model, or *teacher*, to force a more compact network called *student* to produce the same output. This paradigm is called *Knowledge Distillation* (Hinton, Vinyals, & Dean, 2015) and there are many variants of it: some of them use multiple teachers (Shang, Li, Zhu, Jiao, & Li, 2023), while others focus on the efficiency of the training process when labeled samples are scarce (He, Ding, Zhang, & Li, 2022; Zhou, Wang, Zhou, Yu, Bandara, & Bu, 2023).

A different methodology consists in training a big model from the start, and then *Pruning* superfluous parameters. In particular, *Weight Pruning* consists in zeroing connections or parameters already close to zero (LeCun, Denker, & Solla, 1990), but more elaborated methods can also take into consideration the impact of the single weights on the final results (Han, Pool, Tran, & Dally, 2015). Even if weight pruning is a very powerful tool to reduce the network parameters (Frankle & Carbin, 2019), its major drawback is that it does not actually reduce the number of FLOPs at inference time.

A more effective solution consists instead of skipping completely some of the operations. It is the case of *Filter Pruning*, where whole nodes or filters (in the case of convolutional layers) are removed from the architecture. Pruning usually requires some degree of re-training to recover the lost accuracy due to the reduced network capability, but an interesting phenomenon that happens in the early stages of pruning is that most of the time the test accuracy actually increases, due to the regularization effect that pruning unnecessary parameters has on the network. While weight pruning allows more control on what parameters to remove, filter pruning is usually the best solution compression-wise as it allows a drastic reduction of the network parameters such that the models can be actually implemented in small embedded devices (Ressi, Pistellato, Albarelli, & Bergamasco, 2022).

Another technique often used in conjunction with pruning is called *Quantization* (Han, Mao, & Dally, 2016). While pruning aims to reduce the number of parameters, quantization instead targets their precision (Pistellato et al., 2023). As the weights are usually represented by floating point numbers, it is possible to reduce the bits used for the number representation down to single bits (Rastegari, Ordonez, Redmon, & Farhadi, 2016), without affecting the network accuracy.

Finally, pruning techniques might be designed according to the device they will be implemented on. Recently, a new type of neural network has gained interest from the research community: *Memristive Neural Networks* (memNNs). It is a special type of ANN that is based on the "memristor bionic synapse", which replaces the traditional resistors with memristors in the circuit realization process. Pruning algorithms can be designed to improve efficiency and compression rate on this kind of network (Hong, Xiao, Fan, & Du, 2024; Mou et al., 2021; Wen et al., 2019). Targeted pruning algorithms have been developed for hardware components beyond just memristors. For instance, the same idea has been applied to Field-Programmable Gate Array (FPGA) based NNs (Li et al., 2022; Ressi et al., 2022; Shimoda, Sada, & Nakahara, 2019).

In the context of performance evaluation of computer systems, stochastic models whose underlying stochastic processes are Markov chains, play a key role in providing a sound high-level framework for the analysis of software and hardware architectures. Although the use of high-level modeling formalism greatly simplifies the specification of quantitative models (*e.g.*, by exploiting the compositionality properties (Hillston, 1994)), the stochastic process underlying even a very compact model may have several states that make its analysis a difficult, sometimes computationally impossible, task. In order to study models with a large state space without using approximations or resorting to simulations, one can attempt to reduce the state space of the underlying Markov chain by aggregating states with equivalent behaviors. *Lumpability* (Kemeny & Snell, 1976) is an aggregation technique used to cope with the state space explosion problem inherent to the computation of the stationary performance indices of large stochastic models. The lumpability method turns out to be useful on Markov chains exhibiting some structural regularity. Moreover, it allows one to efficiently compute the exact values of the performance indices when the model is actually lumpable. In the literature, several notions of lumping have been introduced: ordinary and weak lumping (Kemeny & Snell, 1976), exact lumping (Schweitzer, 1984), and strict lumping (Buchholz, 1994).

In this paper, we propose a filter pruning technique where we treat fully connected nodes of a neural network as states of a Markov Chain. By borrowing the concept of lumpability and applying it to the neural network, we aggregate some of its nodes and we update the remaining connections such that the output of the network is not affected. We offer background insights into lumpability and neural networks to facilitate understanding for researchers in both domains. Furthermore, we elaborate on the concept of proportional lumpability, demonstrating how it can be extended to linearly dependent weights. We show how to update the weights in such a specific case. While our method requires rigid constraints on the nature of the weights, we examine two different scenarios with relaxed constraints to mirror real-world scenarios. We conduct various experiments on the MNIST benchmark dataset to compress two different architecture types. We also discuss our findings and how they can be combined with other state-of-the-art techniques to improve the efficiency of our method.

With our work, we aim to link together the work of two different communities, the first one focusing on machine learning and network compression and the second one focusing on lumping-based aggregation techniques for performance evaluation. Even if a large number of possible efficient compression techniques have already been published, our main goal is to give a formal demonstration of how it is possible to deterministically remove some of the network parameters to obtain a smaller network with the same performance. Our method condenses many different concepts together, such as some of the ideas exploited by tensor decomposition methods, filter pruning, and the lumpability used to evaluate the performance of complex systems.

In summary, our contributions include:

- a data-free novel filter pruning technique inspired by the concept of lumpability studied for aggregating complex Markov chains.
- a formalization of the proposed method as well as mathematical proof of the preservation of the network behavior.
- an approximate formalization of the proposed method to better approach real-case scenarios.
- an extensive experimental setup.

This paper extends the findings presented in Ressi, Romanello, Piazza, and Rossi (2022). We offer comprehensive proofs of our results and demonstrate the adaptability of our pruning method, particularly when weights represent linear combinations rather than mere proportional relations within the same layer. Additionally, we extend our experiments by examining two different scenarios with relaxed constraints on two different model architectures, and we present our results in a dedicated section.

The paper is structured as follows. In Section 2 we provide a literature review. Section 3 gives the necessary background on neural networks and lumpability. Section 4 formally describes our technique exploiting the notion of exact lumpability for quotienting neural networks. Section 5 presents some experimental results. In Section 6 we discuss the results we obtained and propose different ways to enhance our algorithm. Finally, Section 7 concludes the paper.

2. Related work

To the best of our knowledge, the only paper similar to our work is Prabhakar (2022), where the authors introduce the classical notion of equivalence between systems in Process Algebra to reduce a neural network into another one semantically equivalent. They propose a filter pruning technique, based on some properties of the network, that does not need any data to perform the compression. They also define an approximated version of their algorithm to relax some of the strong constraints they pose on the weights of the network.

Pruning methods can be divided into two macro-categories, online pruning and offline pruning. In the former group, we can find all methods where the number of nodes, connections, or even layers is adapted during the training phase. These techniques are heavily data-dependent. The latter group, instead, consists of all post-training methods, which may or may not require the original dataset.

While data-free pruning algorithms are convenient when a dataset is incomplete, unbalanced, or missing, they usually achieve poorer results compared to data-based compression solutions. Indeed, most pruning techniques usually require at least one stage of fine-tuning of the model. The recovery is often performed iteratively after removing a single parameter, but some techniques re-train the model only after a certain level of compression has been carried out (Blalock, Gonzalez Ortiz, Frankle, & Guttag, 2020).

The authors in Lin et al. (2020) proved how filters with high-rank feature maps retain important information, resulting in minimal performance degradation even when partially unaltered. They also defined how filter pruning techniques can be divided according to *property importance* or *adaptive importance*. In the first group, we find pruning methods that look at intrinsic properties of the networks and do not modify the training loss, such as He, Liu, Wang, Hu, and Yang (2019), Hu, Peng, Tai, and Tang (2016), Prabhakar (2022) and Ressi et al. (2022). In Hu et al. (2016), authors introduce *network trimming*, which iteratively optimizes the network by pruning unimportant neurons based on analysis of their outputs on a large dataset. Authors in He et al. (2019) propose a Filter Pruning via Geometric Median (FPGM), where they redesign the criterion to prune filters with smaller norm values in a convolutional neural network by considering not only filters with large norm deviation but also if the minimum norm of the filters is small. Finally, in Ressi et al. (2022) the authors consider instead the average output of the neurons, removing first the units that are rarely activated.

We can consider our original algorithm to belong to *property importance* methods, as it only looks at the network without using any data. Adaptive importance pruning algorithms like Lin et al. (2019), Liu, Li, Shen, Huang, Yan, and Zhang (2017) and Wang, Liu, Wang, Liu, Alibhai, and He (2022) usually drastically change the loss function, requiring a heavy retrain step and looking for a new proper set of hyperparameters, even though they often achieve better performances with respect to property importance methods. For example, Liu et al. (2017) proposes a method enforcing channel-level sparsity, where insignificant channels are automatically identified during training and pruned afterward, obtaining thin and compact models with comparable accuracy. The authors in Lin et al. (2019) propose an end-to-end structured pruning method that simultaneously prunes filters and other structures by introducing a soft mask to scale their outputs. This approach utilizes generative adversarial learning (GAL) to efficiently learn sparse soft masks, enabling the removal of corresponding structures using the fast iterative shrinkage-thresholding algorithm (FISTA). While Lin et al. (2019) and Liu et al. (2017) are designed for compressing CNNs, Wang et al. (2022) introduces a novel locality-based transfer learning method to enhance the training efficiency of compression autoencoder (CAE) for scientific data compression. By leveraging incremental learning and Kullback Leibler (KL) Divergence – a measure of the closeness of two distributions – as an indicator, this approach accelerates training speed while maintaining high prediction accuracy, addressing the limitations of existing methods.

Avoiding to re-train the network at each pruning step as in Lin et al. (2020) and Wang, Xie, and Shi (2021) is usually faster than other solutions, but there is a higher risk of not being able to recover the performances. For instance, authors in Wang et al. (2021) present RFPruning, a two-stage Retraining-Free pruning method for deep convolutional neural networks (DCNNs) that aims to accelerate inference without sacrificing performance. In the first stage, a sparse learning approach is utilized for rough channel selection during network training. In the second stage, a genetic algorithm is employed to fine-tune the pruning process, achieving the balance between performance and model size. While most pruning techniques focus on reducing the number of operations, sometimes removing whole branches (Abrar & Samad, 2022; Zhou et al., 2023) of the architecture, these algorithms require the availability of the original dataset in order to recover the lost accuracy. When such a scenario is not feasible, it is better to exploit techniques that avoid further training steps.

Another option consists of deciding which parameters to remove according to the impact they have on the rest of the network (Molchanov, Mallya, Tyree, Frosio, & Kautz, 2019; Yu et al., 2018). Finally, while most of the already mentioned methods focus on removing whole filters or kernels from convolutional layers, some other methods target only fully connected layers, or are made to compress classical neural networks (Ashiquzzaman, Van Ma, Kim, Lee, Um, & Kim, 2019; Tan & Motani, 2020).

Contrary to other pruning methods, our approach completely preserves the behavior of the network and it does not depend on the input data, thus not requiring any re-training or fine-tuning. Given the strong assumptions on the weights of the network our technique requires, we also test our pruning under more relaxed constraints, allowing for the updating step to approximate the behavior of the network instead. While our algorithm can be integrated with other techniques to improve the compression rate or to better preserve the original performance of the model, we do not claim our method to perform better than other existing ones but rather explore how formal methods can be applied in this field.

Furthermore, even if there is a close relationship with compressing techniques that force the rank of the tensor to be small, such that redundant sub-structure of the network can be removed without affecting the outcome (Grasedyck, Kressner, & Tobler, 2013), most of these methods introduce some sort of approximation and completely alter the model's architecture. For all these reasons, our scope is not to compare our pruning approach with other existing methods but rather to test how far we can stretch the constraints from the original assumptions.

3. Preliminaries

In this section we formally introduce the notion of neural network in the style of Prabhakar (2022). Moreover, we recall the concept of exact lumpability as it has been defined in the context of continuous-time Markov chains.

Neural networks

A neural network is formed by a layered set of nodes or neurons, consisting of an input layer, an output layer, and one or more hidden layers. Each node that does not belong to the input layer is annotated with a bias and an activation function. Moreover, there are weighted edges between nodes of adjacent layers. We use the following formal definition of neural networks.

For $k \in \mathbb{N}$, we denote by $[k]$ the set $\{0, 1, \dots, k\}$, by $(k]$ the set $\{1, \dots, k\}$, by $[k)$ the set $\{0, \dots, k-1\}$, and by (k) the set $\{1, \dots, k-1\}$.

Definition 1 (*Neural Network*)**.** A *Neural Network (NN)* is a tuple $\mathcal{N} = (k, Act, \{S_\ell\}_{\ell \in [k]}, \{W_\ell\}_{\ell \in (k]}, \{b_\ell\}_{\ell \in (k]}, \{A_\ell\}_{\ell \in (k]})$ where:

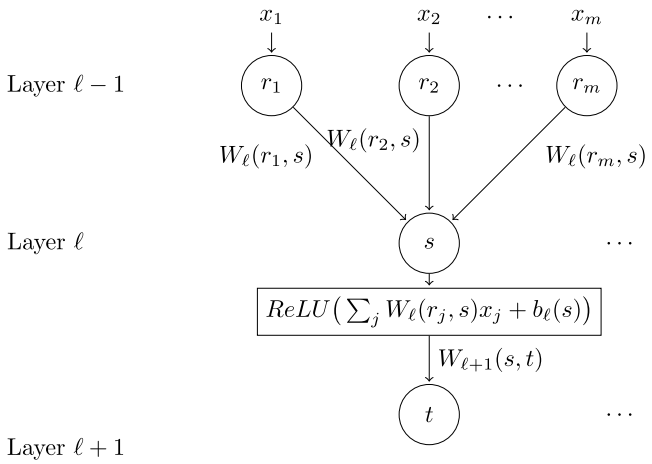


Fig. 1. Node s behavior on input $x_1, x_2, \ldots, x_m$.

- k is the number of layers (except the input layer);
- $\mathcal{A}ct$ is the set of activation functions;
- for $\ell \in [k]$, S_ℓ is the set of nodes of layer ℓ with $S_\ell \cap S_{\ell'} = \emptyset$ for $\ell \neq \ell'$;
- for $\ell \in (k]$, $W_\ell : S_{\ell-1} \times S_\ell \to \mathbb{R}$ is the weight function that associates a weight with edges between nodes at layer $\ell - 1$ and ℓ;
- for $\ell \in (k]$, $b_\ell : S_\ell \to \mathbb{R}$ is the bias function that associates a bias with nodes at layer ℓ;
- for $\ell \in (k]$, $A_\ell : S_\ell \to \mathcal{A}ct$ is the activation association function that associates an activation function with nodes of layer ℓ.

S_0 and S_k denote the nodes in the input and output layers, respectively.

In the rest of the paper, we will refer to NNs in which all the activation association functions are constant, *i.e.*, all the neurons of a layer share the same activation function. Moreover, such activation functions A_ℓ are either ReLU (Rectified Linear Unit) or LeakyReLU, *i.e.*, they are combinations of linear functions. So, from now on we omit the set $\mathcal{A}ct$ from the definition of the NNs.

Example 1. Fig. 1 shows the behavior of node s in Layer ℓ. The input values $x_1, x_2, \ldots x_m$ are propagated by nodes $r_1, r_2, \ldots r_m$ respectively. Node s computes the $ReLU$ of the weighted sum of the inputs plus the bias. The result of this application is the output of s and it is propagated to t.

The operational semantics of a neural network is as follows. Let $v : S_\ell \to \mathbb{R}$ be a valuation for the ℓ-th layer of $\mathcal{N}$ and $Val(S_\ell)$ be the set of all valuations for the ℓ-th layer of $\mathcal{N}$. The operational semantics of $\mathcal{N}$, denoted by $[\![\mathcal{N}]\!]$, is defined in terms of the semantics of its layers $[\![\mathcal{N}]\!]_\ell$, where each $[\![\mathcal{N}]\!]_\ell$ associates with any valuation v for layer $\ell-1$ the corresponding valuation for layer ℓ according to the definition of $\mathcal{N}$. The valuation for the output layer of $\mathcal{N}$ is then obtained by the composition of functions $[\![\mathcal{N}]\!]_\ell$.

Definition 2. The semantics of the ℓ-th layer is the function $[\![\mathcal{N}]\!]_\ell : Val(S_{\ell-1}) \to Val(S_\ell)$ where for all $v \in Val(S_{\ell-1})$, $[\![\mathcal{N}]\!]_\ell(v) = v'$ and for all $s \in S_\ell$, $v'(s)$ is defined by the following Eq. (1):

$$v'(s) = A_\ell(s)\left(\sum_{r \in S_{\ell-1}} W_\ell(r,s)v(r) + b_\ell(s)\right). \tag{1}$$

The input–output semantics of $\mathcal{N}$ is obtained by composing these one-layer semantics. More precisely, we denote by $[\![\mathcal{N}]\!]^\ell$ the composition of the first ℓ layers so that $[\![\mathcal{N}]\!]^\ell(v)$ provides the valuation of the ℓ-th layer given $v \in Val(S_0)$ as input. Formally, $[\![\mathcal{N}]\!]^\ell$ is inductively defined by Eqs. (2) and (3):

$$[\![\mathcal{N}]\!]^1 = [\![\mathcal{N}]\!]_1 \tag{2}$$

$$[\![\mathcal{N}]\!]^\ell = [\![\mathcal{N}]\!]_\ell \circ [\![\mathcal{N}]\!]^{\ell-1} \ \forall \ell \in (k] \tag{3}$$

where $\circ$ denotes the function composition.

We are now in a position to define the semantics of $\mathcal{N}$ as the input–output semantic function $[\![\mathcal{N}]\!]$ defined below.

Definition 3. The input–output semantic function $[\![\mathcal{N}]\!] : Val(S_0) \to Val(S_k)$ is defined by the following Eq. (4):

$$[\![\mathcal{N}]\!] = [\![\mathcal{N}]\!]^k. \tag{4}$$

Lumpability

The notion of *lumpability* has been introduced in the context of performance and reliability analysis. It provides a model aggregation technique that can be used for generating a Markov chain that is smaller than the original one while allowing one to determine exact results for the original process.

The concept of lumpability can be formalized in terms of equivalence relations over the state space of the Markov chain. Any such equivalence induces a *partition* on the state space of the Markov chain and aggregation is achieved by clustering equivalent states into macro-states, reducing the overall state space.

Let S be a finite state space. A (time-homogeneous) Continuous-Time Markov Chain (CTMC) over S is defined by a function

$$Q : S \times S \to \mathbb{R}$$

such that for all $x, y \in S$ with $x \neq y$ it holds that:

- $Q(x,y) \geq 0$ and
- $Q(x,x) = -\sum_{y \in S, x \neq y} Q(x,y)$.

A CTMC defined over S by Q models a stochastic process where a transition from state x to state y, with $x \neq y$, can occur according to an exponential distribution with rate $Q(x,y)$.

Given an initial probability distribution p over the states of a CTMC, one can consider the problem of computing the probability distribution to which p converges when the time tends to infinity. This is the *stationary* distribution and it exists only when the chain satisfies additional constraints. The stationary distribution reveals the limit behavior of a CTMC. Many other performance indexes and temporal logic properties can be defined for studying both the transient and limit behavior of the chain.

Different notions of lumpability have been introduced to reduce the number of states of the chain while preserving its behavior (Alzetta, Marin, Piazza, & Rossi, 2018; Bossi, Focardi, Macedonio, Piazza, & Rossi, 2004; Buchholz, 1994; Hillston, Marin, Piazza, & Rossi, 2013, 2021; Kemeny & Snell, 1976; Marin, Piazza, & Rossi, 2019, 2022; Schweitzer, 1984). In particular, we consider here the notion of *exact lumpability* (Buchholz, 1994; Schweitzer, 1984).

Definition 4 (*Exact Lumpability*)**.** Let (S,Q) be a CTMC and $\mathcal{R}$ be an equivalence relation over S. $\mathcal{R}$ is an *exact lumpability* if for all $S, S' \in \mathcal{R}/S$, for all $x, y \in S$ it holds that:

$$\sum_{z \in S'} Q(z,x) = \sum_{z \in S'} Q(z,y). \tag{5}$$

There exists always a unique maximum exact lumpability relation which allows us to quotient the chain by taking one state for each equivalence class and replacing the rates of the incoming edges with the sum of the rates from equivalent states.

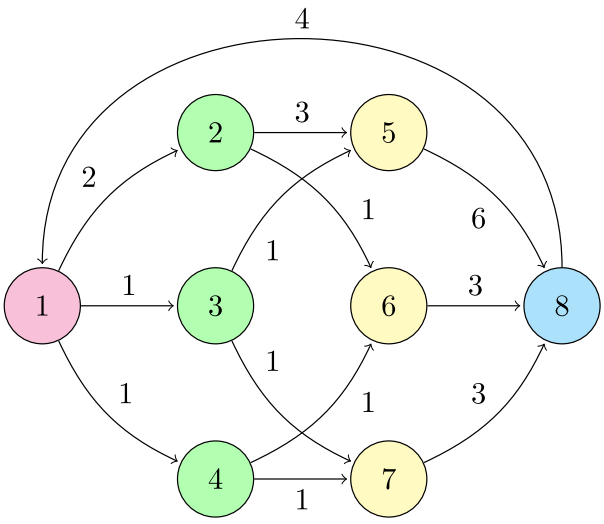


Fig. 2. Proportionally exact lumpable CTMC.

The notion of exact lumpability is in many applicative domains too demanding, thus providing poor reductions. This issue is well-known for all lumpability notions that do not allow any form of approximation. To obtain smaller quotients, still avoiding rough approximations, the notion of *proportional lumpability* has been presented in Marin et al. (2019, 2022) and Piazza and Rossi (2021) as a relaxation of ordinary lumpability. In this paper instead, we introduce the notion of *proportional exact lumpability*, defined as follows.

Definition 5 (*Proportional Exact Lumpability*)**.** Let (S, Q) be a CTMC and $\mathcal{R}$ be an equivalence relation over S. $\mathcal{R}$ is a *proportional exact lumpability* if there exists a function $\rho : S \to \mathbb{R}_{>0}$ such that for all $S, S' \in S/\mathcal{R}$, for all $x, y \in S$ it holds that:

$$\rho(x) \sum_{z \in S'} Q(z, x) = \rho(y) \sum_{z \in S'} Q(z, y). \tag{6}$$

Notice that Eq. (6) is equal to Eq. (5) except for the coefficients $\rho(x)$ and $\rho(y)$.

It can be proved that there exists a unique maximum proportional exact lumpability that can be computed in polynomial time. This is true also if (S, Q) is a *Labeled Graph* instead of a CTMC, *i.e.*, no constraints are imposed on Q.

Example 2. Fig. 2 shows a proportionally exact lumpable Markov chain with respect to the function ρ defined as: $\rho(1) = 1, \rho(2) = 1, \rho(3) = 2, \rho(4) = 2, \rho(5) = 1, \rho(6) = 2, \rho(7) = 2, \rho(8) = 1$ and the equivalence classes $S_1 = \{1\}, S_2 = \{2, 3, 4\}, S_3 = \{5, 6, 7\}, S_4 = \{8\}$.

4. Lumping neural networks

The idea of exploiting exact lumpability for quotienting NN has been proposed in Prabhakar (2022) where a notion of pre-sum preserving backward bisimulation has been considered. It can be easily observed that such a notion coincides with that of exact lumpability. The term (probabilistic) bisimulation is standard in the area of Model Checking, where (probabilistic) temporal logical properties are used for both specifying and synthesizing systems having a desired behavior (Anticoli, Piazza, Taglialegne, & Zuliani, 2016; Bugliesi, Gallina, Marin, Rossi, & Hamadou, 2012; Gallina, Hamadou, Marin, & Rossi, 2011). Since such logics usually formalize the behaviors in terms of forward temporal operators, the bisimulation notions tend to preserve the rates of the outgoing edges (Sproston & Donatelli, 2004). However, as proved in Prabhakar (2022), in order to preserve the behavior of a NN it is necessary to refer to the rates/weights of the incoming edges. This is referred to as *backward probabilistic bisimulation* and coincides with the well-known notion of *exact lumpability* used in the area of performance evaluation.

What was already described in Prabhakar (2022) is extended in this proposal. The key difference between our work and Prabhakar (2022) is that while the latter stops after proving that exact lumpability preserves the behavior of a Neural Network, we extend it by adding two main ingredients. Firstly, we introduce a notion of proportionality between the nodes' incoming rates/biases. Subsequently, we *relax* this notion by exploring the definition of *quasi* proportionality between rates.

Formally, we prove that in the case of ReLU and LeakyReLU activations, proportional exact lumpability preserves the behavior of the network allowing it to obtain smaller quotients. It does not require any retraining step and it ensures the same behavior on all possible inputs. Moreover, since the neural networks we refer to are acyclic it can be computed in linear time.

Definition 6 (*Proportional Exact Lumpability over a NN*)**.** Let $\mathcal{N}$ be a NN. Let $\mathcal{R} = \cup_{\ell \in [k]} \mathcal{R}_\ell$ be such that $\mathcal{R}_\ell$ is an equivalence relation over S_ℓ, for all $\ell \in (k)$ and $\mathcal{R}_0$ is the identity relation over S_0. We say that $\mathcal{R}$ is a *proportional exact lumpability* over $\mathcal{N}$ if for each $\ell \in (k)$ there exists $\rho_\ell : S_\ell \to \mathbb{R}_{>0}$ such that for all $S \in S_\ell/\mathcal{R}_\ell$, for all $S' \in S_{\ell-1}/\mathcal{R}_{\ell-1}$, for all $s_1, s_2 \in S$ the following Eqs. (7) and (8) hold:

$$\rho_\ell(s_1) b_\ell(s_1) = \rho_\ell(s_2) b_\ell(s_2), \tag{7}$$

$$\rho_\ell(s_1) \sum_{r \in S'} W_\ell(r, s_1) = \rho_\ell(s_2) \sum_{r \in S'} W_\ell(r, s_2). \tag{8}$$

There are some differences with respect to the definition of proportional exact lumpability over CTMCs. First, we impose that two equivalent neurons have to belong to the same layer. However, we could have omitted such restriction from the definition and proved that neurons from different layers are never equivalent. This is an immediate consequence of the fact that we refer to acyclic NNs. Moreover, we demand that on input and output nodes the only admissible relation is the identity. This is a substantial difference. Since the nodes in the input layer have no incoming edges the definition of proportional lumpability given over CTMCs allows to collapse them. However, the input nodes in NNs hold the input values that have to be propagated, so they cannot be collapsed. This is true also for the output nodes since they represent the result of the computation.

It can be proved that there always exists a unique maximum proportional exact lumpability over a NN. If we use proportional exact lumpability for reducing the dimension of a NN by collapsing the equivalent neurons, we have to modify the topology and the weights of the NN as formalized below.

Definition 7 (*Proportional Reduced NN*)**.** Let $\mathcal{N} = (k, \{S_\ell\}_{\ell \in [k]}, \{W_\ell\}_{\ell \in (k]}, \{b_\ell\}_{\ell \in (k]}, \{A_\ell\}_{\ell \in (k]})$ be a NN. Let $\mathcal{R}$ be a proportional exact lumpability over $\mathcal{N}$. The NN $\mathcal{N}/\mathcal{R} = (k, \{S'_\ell\}_{\ell \in [k]}, \{W'_\ell\}_{\ell \in (k]}, \{b'_\ell\}_{\ell \in (k]}, \{A'_\ell\}_{\ell \in (k]})$ is defined by:

- $S'_\ell = \{[s] \mid [s] \in S_\ell/\mathcal{R}\}$, where s is an arbitrarily chosen representative for the class;
- $W'_\ell([s_1], [s_2]) = \rho_{\ell-1}(s_1) \sum_{r \in [s_1]} \frac{W_\ell(r, s_2)}{\rho_{\ell-1}(r)}$;
- $b'_\ell([s]) = b_\ell(s)$;
- $A'_\ell([s]) = A_\ell(s)$.

Despite the arbitrary choice of the representative, we can prove that the reduced NN's behavior coincides with that of the initial one over all the inputs.

We first introduce a Lemma to characterize the semantics of a pruned layer.

Lemma 1. *Let $v \in Val(S_{\ell-1})$ be a valuation for layer $\ell-1$. Let $v'' = \left([\![\mathcal{N}]\!]_{\ell+1} \circ [\![\mathcal{N}]\!]_{\ell}\right)(v)$ and $u'' = \left([\![\mathcal{N}/\mathcal{R}_\ell]\!]_{\ell+1} \circ [\![\mathcal{N}/\mathcal{R}_\ell]\!]_{\ell}\right)(v)$ be the valuation for layer $\ell+1$ in $\mathcal{N}$ and $\mathcal{N}/\mathcal{R}_\ell$, respectively. The following Eq. (9) hold:*

$$v'' = u'' . \tag{9}$$

Proof. Let $t \in S_{\ell+1}$. For the sake of readability, we make some assumptions on layer ℓ. In any case, the theorem is easily extendable to the generic case. Let $s_1, s_2, s_3, s_4, s_5 \in S_\ell$ be five neurons from layer ℓ such that $s_2, s_3 \in [s_1]_{\mathcal{R}_\ell}$ and $s_5 \in [s_4]_{\mathcal{R}_\ell}$. Let $v' = [\![\mathcal{N}]\!]_\ell(v)$ be the valuation at layer ℓ. The following relations between $v'(s_1), v'(s_2), v'(s_3), v'(s_4)$, and $v'(s_5)$, represented in Eqs. (10), (11), and (12), hold:

$$\rho_\ell(s_1)v'(s_1) = \rho_\ell(s_2)v'(s_2) \tag{10}$$

$$\rho_\ell(s_1)v'(s_1) = \rho_\ell(s_3)v'(s_3) \tag{11}$$

$$\rho_\ell(s_4)v'(s_4) = \rho_\ell(s_5)v'(s_5) \tag{12}$$

The reason for such a result is twofold: first, we are not lumping layer $\ell-1$. Hence, there is no change in the valuation for layer $\ell-1$. Second, s_1, s_2, and s_3 are equivalent according to $\mathcal{R}_\ell$ (the same holds for s_4 and s_5). By definition, the value of $v''(t)$ is as in (13)

$$\begin{aligned} v''(t) = \; & A_{\ell+1}(t)\big(W_{\ell+1}(s_1,t)v'(s_1) + \\ & + W_{\ell+1}(s_2,t)v'(s_2) + W_{\ell+1}(s_3,t)v'(s_3) + W_{\ell+1}(s_4,t)v'(s_4) \\ & + W_{\ell+1}(s_5,t)v'(s_5) + b(t)\big) \end{aligned} \tag{13}$$

Using the relation between the values of v' represented in Eqs. (10), (11), and (12), we can rewrite Eq. (13) as in the following Eq. (14):

$$\begin{aligned} & = A_{\ell+1}(t)\left(\sum_{i=1}^{3}\left[W_{\ell+1}(s_i,t)\frac{\rho_\ell(s_1)}{\rho_\ell(s_i)}v'(s_1)\right] + \sum_{i=4}^{5}\left[W_{\ell+1}(s_i,t)\frac{\rho_\ell(s_4)}{\rho_\ell(s_i)}v'(s_4)\right]\right) \\ & = A_{\ell+1}(t)\Bigg(v'(s_1)\left[W_{\ell+1}(s_1,t) + W_{\ell+1}(s_2,t)\frac{\rho_\ell(s_1)}{\rho_\ell(s_2)} + W_{\ell+1}(s_3,t)\frac{\rho_\ell(s_1)}{\rho_\ell(s_3)}\right] + \\ & \quad + v'(s_4)\left[W_{\ell+1}(s_4,t) + W_{\ell+1}(s_5,t)\frac{\rho_\ell(s_4)}{\rho_\ell(s_5)}\right]\Bigg) \end{aligned} \tag{14}$$

which is equal to $u''(t)$. □

In order to prove the overall equivalence between the initial and the reduced networks we introduce the following Lemma:

Lemma 2. *Let i, j be two layers index. If $j < i-1$ or $j > i$, then $[\![\mathcal{N}]\!]_i = [\![\mathcal{N}/\mathcal{R}_j]\!]_i$*

Proof. Let i, j be two layers. In the first case, $j > i$. Since the reduced layer is after the ith, the semantic at layer i is unchanged. In the second case, $j < i-1$. We are reducing a layer that is *before* the ith. By Lemma 1, we know that the semantic of layer j remains the same. Hence, layer i will not be affected by the reduction □

We can now conclude that the input–output behavior of the network is preserved after the reduction.

Theorem 1. *Let $\mathcal{N}$ be a NN and $\mathcal{R}$ be a proportional exact lumpability over $\mathcal{N}$. It holds that*

$$[\![\mathcal{N}/\mathcal{R}]\!] = [\![\mathcal{N}]\!] . \tag{15}$$

Proof. By definition of $[\![\mathcal{N}]\!]$, we have:

$$[\![\mathcal{N}]\!] = [\![\mathcal{N}]\!]_k \circ [\![\mathcal{N}]\!]_{k-1} \circ \cdots \circ [\![\mathcal{N}]\!]_1 \circ [\![\mathcal{N}]\!]_0 \tag{16}$$

The application of Lemma 1 to the two rightmost terms of Eq. (16) yields to:

$$[\![\mathcal{N}]\!]_k \circ [\![\mathcal{N}]\!]_{k-1} \circ \cdots \circ [\![\mathcal{N}/\mathcal{R}_0]\!]_1 \circ [\![\mathcal{N}/\mathcal{R}_0]\!]_0 \tag{17}$$

Using Lemma 2 on all the other terms of (17) we get:

$$[\![\mathcal{N}/\mathcal{R}_0]\!]_k \circ [\![\mathcal{N}/\mathcal{R}_0]\!]_{k-1} \circ \cdots \circ [\![\mathcal{N}/\mathcal{R}_0]\!]_2 \circ [\![\mathcal{N}/\mathcal{R}_0]\!]_1 \circ [\![\mathcal{N}/\mathcal{R}_0]\!]_0 \tag{18}$$

We can interpret $[\![\mathcal{N}/\mathcal{R}_0]\!]$ as our *new* network. Hence, we proceed by applying Lemma 1 to $[\![\mathcal{N}/\mathcal{R}_0]\!]_2 \circ [\![\mathcal{N}/\mathcal{R}_0]\!]_1$ obtaining $[\![\mathcal{N}/(\mathcal{R}_0 \cup \mathcal{R}_1)]\!]_2 \circ [\![\mathcal{N}/(\mathcal{R}_0 \cup \mathcal{R}_1)]\!]_1$. Consequently, the application of Lemma 2 to the remaining terms of (18) yields to:

$$\begin{aligned} & [\![\mathcal{N}/(\mathcal{R}_0 \cup \mathcal{R}_1)]\!]_k \circ [\![\mathcal{N}/(\mathcal{R}_0 \cup \mathcal{R}_1)]\!]_{k-1} \circ \\ & \quad \cdots \circ [\![\mathcal{N}/(\mathcal{R}_0 \cup \mathcal{R}_1)]\!]_2 \circ [\![\mathcal{N}/(\mathcal{R}_0 \cup \mathcal{R}_1)]\!]_1 \circ [\![\mathcal{N}/(\mathcal{R}_0 \cup \mathcal{R}_1)]\!]_0 \end{aligned} \tag{19}$$

Repeating such process starting from (19) and recalling that $\mathcal{R} = \bigcup_{i\in[k]} \mathcal{R}_i$, we end up with (20):

$$[\![\mathcal{N}/\mathcal{R}]\!]_k \circ [\![\mathcal{N}/\mathcal{R}]\!]_{k-1} \circ \cdots \circ [\![\mathcal{N}/\mathcal{R}]\!]_2 \circ [\![\mathcal{N}/\mathcal{R}]\!]_1 \circ [\![\mathcal{N}/\mathcal{R}]\!]_0 \tag{20}$$

which is exactly $[\![\mathcal{N}/\mathcal{R}]\!]$, hence proving (15). □

Example 3. Fig. 3 shows how the pruning technique works on two nodes s_1, s_2. In particular, s_2 input weights are proportional to s_1's. The algorithm proceeds in two steps. Firstly, s_2 is deleted together with all its input and output edges. Secondly, the weight from s_1 to t is modified by adding $\rho W_{\ell+1}(s_2, t)$.

The maximum proportional exact lumpability over $\mathcal{N}$ together with the reduced network can be efficiently computed by proceeding top-down from layer 1 to $k-1$. Since the network is acyclic, each layer is influenced only by the previous one. Hence, the computation is linear with respect to the number of edges of the network.

Before proving the overall result about the complexity of the algorithm, we prove that a *greedy technique* is enough to get a maximum proportional exact lumpability.

Lemma 3. *Let $\ell, \ell+1 \in (k)$ be two layers. Let $\mathcal{R}$ be a proportional exact lumpability such that $t_1, t_2 \in S_{\ell+1}$ and $(t_1, t_2) \in \mathcal{R}$. Let $s_1, s_2 \in S_\ell$ such that $(s_1, s_2) \notin \mathcal{R}$. Let $\mathcal{R}'$ be defined by the following Eq. (21)*

$$\mathcal{R}' = \left(\mathcal{R} \cup \{(s_1, s_2)\}\right)^{tr} \tag{21}$$

where we denote the transitive closure of a relation with $(\cdot)^{tr}$. It holds that:

- $\rho_{\ell+1}(t_1)b_{\ell+1}(t_1) = \rho_{\ell+1}(t_2)b_{\ell+1}(t_2)$
- $\forall S \in S_\ell/\mathcal{R}'_\ell : \rho_{\ell+1}(t_1)\sum_{s\in S} W_{\ell+1}(s, t_1) = \rho_{\ell+1}(t_2)\sum_{s\in S} W_{\ell+1}(s, t_2)$

Proof. Let s_1, s_2, t_1, t_2 be defined as in the statement of the Lemma. Since $(t_1, t_2) \in \mathcal{R}$ and $(s_1, s_2) \notin \mathcal{R}$, we have:

- $\rho_{\ell+1}(t_1)b_{\ell+1}(t_1) = \rho_{\ell+1}(t_2)b_{\ell+1}(t_2)$
- $\rho_{\ell+1}(t_1)\sum_{s\in[s_1]} W_{\ell+1}(s, t_1) = \rho_{\ell+1}(t_2)\sum_{s\in[s_1]} W_{\ell+1}(s, t_2)$
- $\rho_{\ell+1}(t_1)\sum_{s\in[s_2]} W_{\ell+1}(s, t_1) = \rho_{\ell+1}(t_2)\sum_{s\in[s_2]} W_{\ell+1}(s, t_2)$

Let us now consider $\mathcal{R}'$ instead of $\mathcal{R}$, where s_1 and s_2 have been put in the same equivalence class. Since the equivalence classes of $\mathcal{R}'$ are union of classes of $\mathcal{R}$, we get that:

- $\rho_{\ell+1}(t_1)b_{\ell+1}(t_1) = \rho_{\ell+1}(t_2)b_{\ell+1}(t_2)$
- $\forall S \in S_\ell/\mathcal{R}'_\ell : \rho_{\ell+1}(t_1)\sum_{s\in S} W_{\ell+1}(s, t_1) = \rho_{\ell+1}(t_2)\sum_{s\in S} W_{\ell+1}(s, t_2)$ □

The overall meaning of this last Lemma is that during the computation of $\mathcal{R}$, when two nodes are declared equivalent, we do not have to re-check their equivalence when some other layer is reduced.

Thanks to this last result, we obtain the following theorem.

Theorem 2. *Let $\mathcal{N}$ be a NN. There exists a unique maximum proportional exact lumpability $\mathcal{R}$ over $\mathcal{N}$. Moreover, $\mathcal{R}$ and $\mathcal{N}/\mathcal{R}$ can be computed in linear time with respect to the size of $\mathcal{N}$, i.e., in time $\Theta\left(\sum_{\ell\in(k)} |S_{\ell-1} \times S_\ell|\right)$.*

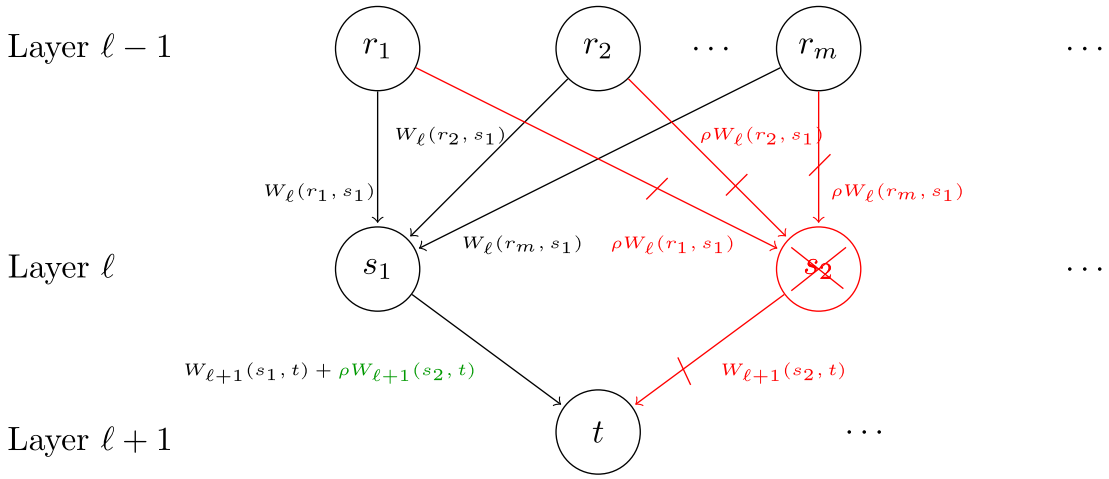


Fig. 3. Pruning one node and updating the network.

Algorithm 1 Computes the maximal proportional exact lumpability on $\mathcal{N}$

1: **function** LumpNeuralNetwork($\mathcal{N} = (k, \{S_\ell\}_{\ell\in[k]}, \{W_\ell\}_{\ell\in(k]}, \{b_\ell\}_{\ell\in(k]}, \{A_\ell\}_{\ell\in(k]})$)
2: $S'_0 \leftarrow \{S_i = [s_i] \ : \ s_i \in S_0\}$
3: **for all** $S_i = [s_i] \in S'_0$ **do**
4: **for all** $\bar{s}_j \in S_1$ **do**
5: $O(S_i, \bar{s}_j) \leftarrow W_1(s_i, \bar{s}_j)$
6: $\tilde{W}_1(S_i, \{\bar{s}_j\}) \leftarrow W_1(s_i, \bar{s}_j)$
7: **end for**
8: **end for**
9: **for all** $l \in (k]$ **do**
10: $S'_l \leftarrow \varnothing$
11: **while** $S_l \neq \varnothing$ **do**
12: $s \leftarrow$ Pick(S_l)
13: $C \leftarrow \{s\}$
14: $b'_l(C) \leftarrow b_l(s), A'_l(C) \leftarrow A_l(s)$
15: **for all** $S' \in S'_{l-1}$ **do**
16: $W'_l(S', C) \leftarrow \tilde{W}_l(S', \{s\})$
17: **end for**
18: **for all** $s' \in S_l \setminus \{s\}$ **do**
19: $\rho_{s'} \leftarrow b_l(s)/b_l(s')$
20: **for all** $S' \in S'_{l-1}$ **do**
21: **if** $\rho_{s'} * O(S', s') \neq O(S', s)$ **then**
22: GO TO Line 18
23: **end if**
24: **end for**
25: $C \leftarrow C \cup \{s'\}$
26: $S \leftarrow S \setminus \{s'\}$
27: $R(s') \leftarrow \rho_{s'}$
28: **end for**
29: $S'_l = S'_l \cup \{C\}$
30: **end while**
31: **for all** $C \in S'_l, s' \in S_{l+1}$ **do**
32: $O(C, s') \leftarrow \sum_{r\in C} W_{l+1}(r, s')$
33: $\tilde{W}_l(C, \{s'\}) \leftarrow \sum_{r\in C} R(r) W_{l+1}(r, s')$
34: **end for**
35: **end for**
36: **return** $\mathcal{N} \setminus \mathcal{R} = \left(k, \{S'_\ell\}_{\ell\in[k]}, \{W'_\ell\}_{\ell\in(k]}, \{b'_\ell\}_{\ell\in(k]}, \{A'_\ell\}_{\ell\in(k]}\right)$
37: **end function**

Let $\mathcal{N} = (k, \{S_\ell\}_{\ell\in[k]}, \{W_\ell\}_{\ell\in(k]}, \{b_\ell\}_{\ell\in(k]}, \{A_\ell\}_{\ell\in(k]})$ be a Neural Network. `LumpNeuralNetwork`($\mathcal{N}$) defined in Algorithm 1 computes the maximum proportional exact lumpability $\mathcal{R}$ over $\mathcal{N}$.

Before unraveling the technical details, we introduce a small example to describe the idea behind the algorithm internals.

Suppose we want to find the equivalence classes in layer l, with $1 \leq l \leq k$. For sake of readability, assume $S'_{l-1} = \{C_1, C_2\}$—layer $l-1$ has been previously split in two equivalence classes. Pick one element s from S_l. Roughly speaking, s is the representative of the *next* equivalence class we will create. We want to find all the elements $s' \in S_l \setminus \{s\}$ such that $s\mathcal{R}s'$. Using Definition 7, s and s' are equivalent according to $\mathcal{R}$ if and only if:

- $b_l(s)$ is proportional to $b_l(s')$
- for both C_1 and C_2—the two equivalence classes identified in S'_{l-1}—the sum of the weights from $C_1(C_2)$ to s is proportional to the sum of the weights from $C_1(C_2)$ to s'.

To identify all these s', we start by noticing that $\rho_l(s)$ can be set to 1. Exploiting this fact, we can compute $\rho_l(s')$—from Definition 7—as $b_l(s)/b_l(s')$. Using such value, what is left to check is whether the weights from C_i to s are proportional to the weights from C_i to s'. If this condition holds, then s and s' are equivalent. Otherwise, they are not. Once s is checked against all the elements inside $S_l \setminus \{s\}$, the equivalence class $[s]$ is complete. We iterate the procedure starting with the choice of s until all elements of S_l have been put in some equivalence class.

This very idea has been exploited to devise Algorithm 1. The for loop at lines 3–8 handles the first layer of the Neural Network. Each element is treated as a singleton equivalence class.

For loop 9–35 takes care of splitting all the other layers, one per time. Let S_l be the current layer. It is split completely inside for loop 11–30. We denote with C the current equivalence class, which at the beginning contains just s—notice that any strategy to pick s from S_l at line 12 can be adopted.

The for loop 18–28 is devoted to find all the elements left in S_l that are equivalent to s. This goal is accomplished by first iterating through all elements $s' \in S_l$. For each one of them, we scan all the equivalence classes S' from the layer $l-1$—For loop 20–24. At line 21 we check if the sum of the weights from S' to s—stored in $O(S', s)$—is proportional to the sum of the weights from S' to s'—stored in $O(S', s')$. If there exists one class S' such that this condition does not hold, then s and s' are not equivalent and we move to the next s'. If no such S' is found, then s' is added to C and we remove it from S_l.

Variable O is updated in lines 31–34. On the other hand, the variable $\tilde{W}$ is used to store the weighted sum of the rates, so that we can correctly populate W'_l at each iteration. The function returns the lumped neural network $\mathcal{N} \setminus \mathcal{R}$.

We now investigate the complexity of the proposed algorithm.

We focus on one specific iteration of loop 9–35.

The complexity of while loop 11–30 is composed of: the number of iterations performed and the complexity of each one of them. The while loop is performed $\mathcal{O}(|S_l|)$ times, with we worst case reached when all the equivalence classes are singleton. For what concerns the complexity of each iteration, there are two inner for loops. The former, lines 15–17, has a $\mathcal{O}(|S'_{l-1}|)$ running time. The latter, lines 18–28, has complexity $\mathcal{O}(|S_l|) * \mathcal{O}(|S'_{l-1}|) = \mathcal{O}(|S_l| * |S'_{l-1}|)$. Hence, the overall complexity of one iteration of while loop 11–30 is $\mathcal{O}(|S'_{l-1}|) + \mathcal{O}(|S_l| * |S'_{l-1}|) = \mathcal{O}(|S_l| * |S'_{l-1}|)$.

Recalling that the while loop has to be performed $\mathcal{O}(|S_l|)$ times, we obtain $\mathcal{O}(|S_l|^2 * |S'_{l-1}|)$.

The semantical equivalence between $\mathcal{N}$ and $\mathcal{N} \setminus \mathcal{R}$ has been proven in Theorem 1. We want the reader to focus on the fact that the only assumption made in such a theorem is that $\mathcal{R}$ is a proportional lumpability. This is because the main *equivalence* result is proven in Lemma 1. In such Lemma, proved that the (Leaky)ReLU is amenable to be used to compute a Proportional reduced NN (as for Definition 7) by exploiting the following property:

$$\forall y \in \mathbb{R}\, \forall r \in \mathbb{R}_{>0} \;\; ReLU(r * y) = r * ReLU(y)\,. \tag{22}$$

Hence, in some sense, Theorem 1 is unaware of which activation function f is used as long as a result like Lemma 1 can be proved for f.

In order to guarantee the correctness of the removal on all possible inputs, as stated in Theorem 1, it is not possible to exploit less restrictive relationships than proportionality. This fact can also be formally proved, under the hypothesis that the input set is sufficiently rich. However, one could ask what happens if we move from a simple proportionality relation to a linear dependence. For instance, let us consider what happens if in Definition 6 we relax the two equations by considering that s_1 is a linear combination of s_2 and s_3, *i.e.*:

- $\rho_\ell(s_1) b_\ell(s_1) = \rho_\ell(s_2) b_\ell(s_2) + \rho_\ell(s_3) b_\ell(s_3)$,
- $\rho_\ell(s_1) \sum_{r \in S'} W_\ell(r, s_1) = \rho_\ell(s_2) \sum_{r \in S'} W_\ell(r, s_2) + \rho_\ell(s_3) \sum_{r \in S'} W_\ell(r, s_3)$.

In this case, we could eliminate s_1 by including its contribution on the outgoing edges of both s_2 and s_3. Unfortunately, the behavior of the network is preserved only for those input values $x_1, x_2, \dots, x_m$ which ensures that $\sum_{j=1}^m W_\ell(r_j, s_2) x_j + b_\ell(s_2)$ and $\sum_{j=1}^m W_\ell(r_j, s_3) x_j + b_\ell(s_3)$ have the same sign, since

- $\forall y_1, y_2 \in \mathbb{R}, \;\; \forall n_1, n_2 \in \mathbb{R}_{>0}$,
- $ReLU(n_1 * y_1 + n_2 * y_2) = n_1 * ReLU(y_1) + n_2 * ReLU(y_2)$ iff $y_1 * y_2 \geq 0$.

In other terms, our analysis points out that reduction techniques based on linear combinations of neurons can be exploited without retraining the network only when strong hypotheses on the sign of the neurons hold.

For better understanding, a graphic representation of the problem can be seen in Fig. 4. In this example, we consider all ratios ρ as unitary and we ignore the bias. Let us consider $W(r, s_3)$ (in red), the incoming weights of a fully connected node s_3, as a linear combination of the weights $W(r, s_1)$ and $W(r, s_2)$ of two different nodes in the same layer. If the vectors $W(r, s_1)$ and $W(r, s_2)$ have at least one component with a different sign (as in the example) there is a chance that also their linear combination will have negative components. In this case, the sign of the scalar product between the vector $W(r, s_3)$ and an input x depends on the angle θ between them. If $cos(\theta)$ is negative, then the result will also be negative. In this case, the update of the output of s_1 will cause the network to exhibit an approximate behavior. On the other hand, when $W(r, s_3) * x$ is positive, it will be the output of unit s_2 to introduce an error. This can be seen in Figs. 4(a) and 4(b), which show how the output of s_1, s_2 and s_3 change according to the angle they form with the inputs x_1 (in green) and x_2 (in blue).

We now move to the study of *quasi* equivalent nodes. In particular, we relax Definition 6 so that it takes into account a small *error*, in the same spirit of Casagrande, Dreossi, and Piazza (2012). What we get is that the relation between the incoming weights becomes:

$$\rho_\ell(s_1) \sum_{r \in S'} W_\ell(r, s_1) = \rho_\ell(s_2) \sum_{r \in S'} W_\ell(r, s_2) + \epsilon_{S'} \tag{23}$$

while, on the other hand, the relation between the biases:

$$\rho_\ell(s_1) b_\ell(s_1) = \rho_\ell(s_2) b_\ell(s_2) + \epsilon_{S'} \tag{24}$$

With a small re-arrangement of Eqs. (23) and (24), we obtain:

$$\sum_{r \in S'} W_\ell(r, s_1) = \frac{\rho_\ell(s_2)}{\rho_\ell(s_1)} \sum_{r \in S'} W_\ell(r, s_2) + \frac{\epsilon_{S'}}{\rho_\ell(s_1)} \tag{25}$$

$$b_\ell(s_1) = \frac{\rho_\ell(s_2)}{\rho_\ell(s_1)} b_\ell(s_2) + \frac{\epsilon_{S'}}{\rho_\ell(s_1)} \tag{26}$$

that can be analyzed easily. In particular, from Eqs. (25) and (26) we can see that the error that can be introduced is strictly related to $\rho_\ell(s_1)$. In the case that $\rho_\ell(s_1)$ is smaller than 1, then the error will drastically increase. On the other hand, if $\rho_\ell(s_1)$ is much greater than 1, the introduced error will be small.

Moreover, due to the layered nature of neural networks, the introduction of an error in a single layer propagates throughout the whole network.

Again, we provide a visualization of the weights to prune in Fig. 5. Similarly to Fig. 4, we consider the red vector $W(r, s_3)$ as the weights coming into unit s_3. The vector is the target of our pruning method as it can be obtained as a linear combination of the weights of units s_1 (blue) and s_2 (green). In our setup, we also allow to remove a quasilinear vector, which implies any vector falling within a cylinder with a ray equal to ϵ.

5. Experimental results

We now present the experiments we carried on to test the feasibility of our approach, and in particular how the performance degrade when we relax some of the constraints imposed by our compression technique.

5.1. Simulation design

In this section, we explore different scenarios where our method can be effectively implemented. We have already demonstrated that, within specific constraints, it is possible to remove nodes from a fully connected layer without altering the output of the network. Specifically, we have shown that it is possible to prune a unit if its incoming weights are a linear combination of the weights entering at least one other unit.

The sole limitation of our approach concerns the signs of the weights. When the linear combination includes discordant signs, some information is lost after the ReLU activation. In other words, the output of the node to prune must exhibit the same sign as the sign of the output of other nodes whose weights are a linear combination of the weights of the target node.

To gain a deeper understanding of both the potential and constraints of our technique, we focus on pruning a single dense layer and observe how performances drop under different conditions. Our pruning method requires prior knowledge of which nodes present weights that can be linearly combined to obtain the weights of the target node, as well as their coefficients. To evade the NP-complete problem of finding such elements, we synthetically construct the layer to be pruned, enabling us to determine these parameters in advance.

After the training phase, we freeze some of the weights of the layer and manually overwrite the remaining ones as linear combinations of some of the frozen ones. After re-training the network, the accuracy is easily recovered, allowing us to proceed with the pruning of this layer.

We test our method under three different scenarios:

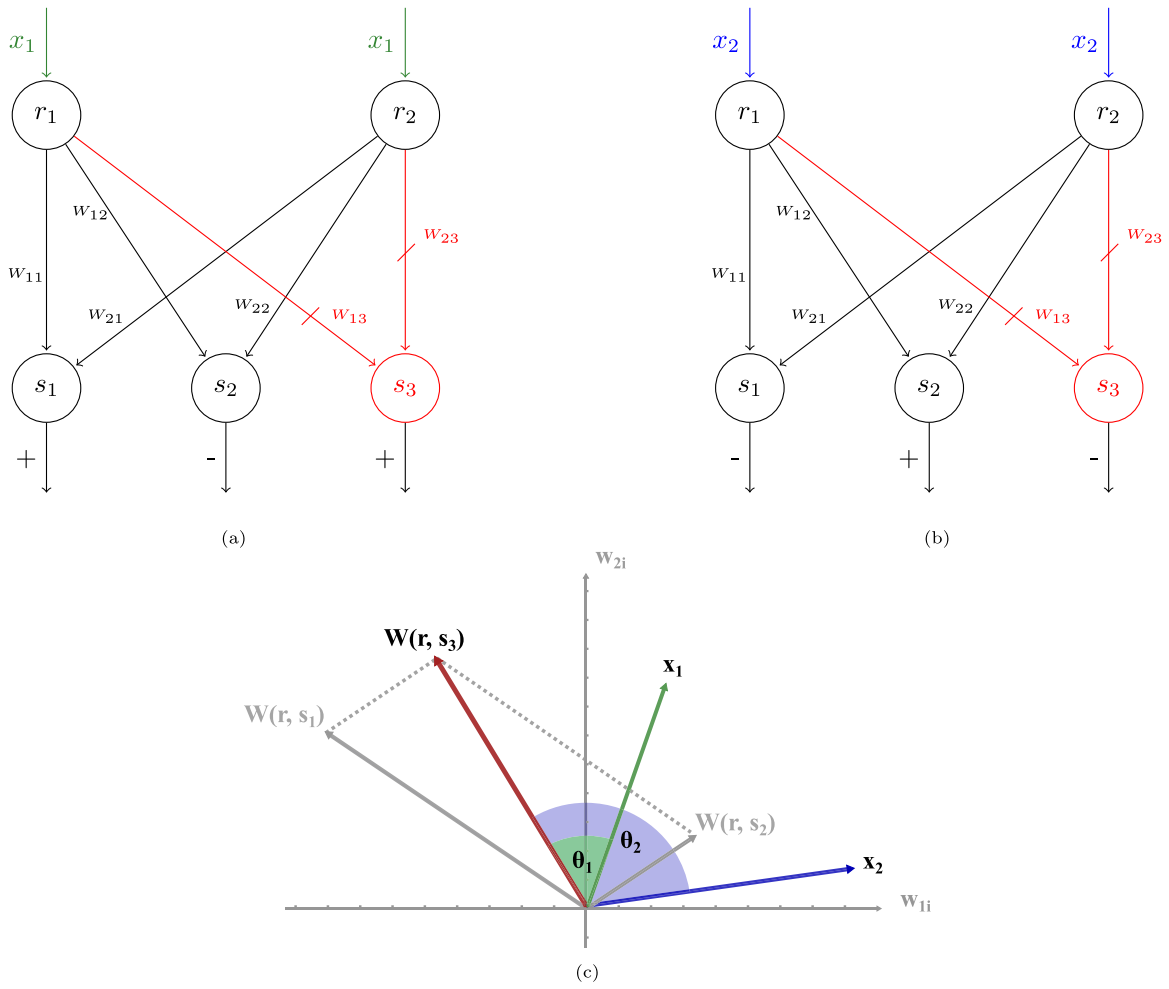


Fig. 4. Visual representation of how different inputs (here x_1 and x_2) can change the sign of the output of a unit. In this example, $W(r, s_3)$ are the weights incoming to the unit s_3, and they can be represented as a linear combination of the weights incoming to nodes s_1 and s_2.

1. All the weights in the layer are positive, and the weights are synthetically configured as linear combinations.
2. All the weights in the layer are positive, and the weights of a node within the layer are *quasi*-linear combinations of the weights of other nodes in the same layer, with a small error ϵ.
3. We do not impose any constraints on the weights' sign, and the weights of the nodes to be pruned are overwritten as linear combinations of a fixed subset of weights.

For the first two setups, we enforce strict positivity of all weights to prevent conflicting signs. This constraint is stronger than what is strictly required for our pruning method to function correctly, but it ensures uniform sign outputs. Regarding the first point, as demonstrated, we have verified that pruning all redundant nodes and adjusting the outputs of the remaining nodes according to our method do not affect the network's output at all.

More interestingly, we examine how our technique performs when we allow a degree of flexibility in weight properties. To this end, we explore two different scenarios. In the former, we analyze the performance degradation of our technique when we introduce an approximation parameter, denoted as ϵ. In the latter, we investigate a real-world scenario by dealing with weights of discordant signs.

To evaluate the robustness of our method, we set up a series of experiments where we implemented the neural network pruning by lumping. In particular, we want to show how accuracy is affected when the weights of the node to prune are not merely proportional to those of another node in the same layer. Instead, these weights are a linear combination of the weights of two or more other nodes.

Our experiments aim to analyze the performance of our pruning method under relaxed constraints, rather than comparing it with existing heuristic methods. We apply the pruning technique to one of the fully connected layers in three different model architectures, trained on two benchmark datasets.

5.2. Datasets

For our analysis, we use two benchmark datasets. The first is the MNIST dataset comprising 7,000 grayscale images of handwritten digits, each measuring 28 × 28 pixels and categorized into 10 classes. The second dataset is a benchmark dataset from Kaggle called Zillow's Home Value Prediction (Zestimate)[1], from which we extracted a smaller dataset comprising: Lot Area (in sq ft), Overall Quality (scale from 1 to 10), Overall Condition (scale from 1 to 10), Total Basement Area (in sq ft), Number of Full Bathrooms, Number of Half Bathrooms, Number of Bedrooms above ground, Total Number of Rooms above ground, Number of Fireplaces, Garage Area (in sq ft). The last column is a boolean value set to 1 only if the house price is above the median. In our experiment, we refer to this dataset as the *Housing* dataset.

[1] https://www.kaggle.com/c/zillow-prize-1/overview

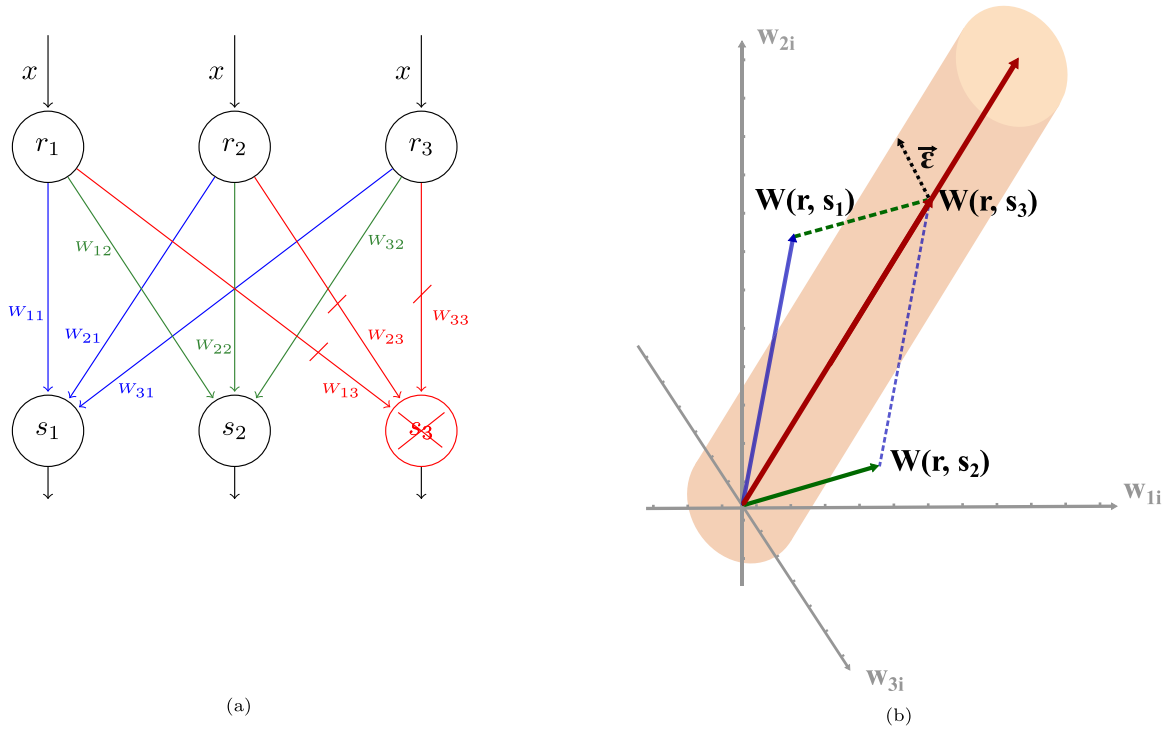


Fig. 5. Graphical representation of quasilinearity. The red vector represents the weights of the node to prune, as a linear combination of the weights of two different nodes in the same layer (green and blue). Allowing the weights to differ by a vector of small values ϵ means that all the vectors falling in the sphere can still be considered as linear combinations of the green and blue vectors of weights.

5.3. Architectures

We conduct our experiments on the MNIST dataset with two different neural networks. The first model is a *Convolutional Neural Network (CNN)* comprising two convolutional blocks, each with 32 3×3 filters, followed by a max-pooling layer. After a flatten layer, it consists of three fully connected layers (*fc*), ending with the softmax layer made of 10 units. While the layer to prune has a fixed length of 128 units, we change the number of units in the previous layer to study how the method behaves under different pruning ratios.

The reason for this choice depends on the rank of the weights matrix: as the weights are made of floating point numbers it is unreasonable to think that the rank of the matrix can be lower than the smaller dimension. At the same time, whenever the number of weights is larger than the number of their components, we can always describe a weight in the matrix as a linear combination of the other weights.

The second architecture is a dense *autoencoder (AE)*, where we focus on compressing the fully connected layer before the output layer. After the input layer, the network comprises two fully connected layers for the encoder, of 128 and 16 units respectively, followed by two layers for the decoder, where the first 128 units fully connected layer precedes the final output layer. While the inputs given to the CNN are the 28×28 images, the autoencoder takes a flattened version of the digits and returns a reconstructed output of the same dimensionality. Once again, we prune up to 96 weights in the second last layer.

Finally, we also test our pruning method on the housing dataset, where we implement a simple *Dense* network made of only two hidden fully connected layers (made of 16 and 128 units, respectively), where the second one is the target of our pruning. The output layer consists of a single node, whose role is to compute if the house price is above the median.

As required by our method, all networks use ReLU activations on the layer to prune, and we also exploit ReLU in all the other layers, except for the output one.

5.4. Formulations of the performance metrics

We evaluate the performance of our method by measuring the accuracy for the CNN and the Dense network, while we report the reconstruction loss for the autoencoder. The accuracy (Eq. (27)) is a ratio between the correctly predicted test samples over all the predictions:

$$accuracy = \frac{truepositives + truenegatives}{truepositives + truenegatives + falsepositives + falsenegatives} \tag{27}$$

In our experiments, we can consider the accuracy as the performance metric, instead of the F1-score, since the MNIST dataset is balanced. We also report the accuracy of our Housing dataset, as we are simply predicting a binary value.

To assess the performance of the autoencoder, we use the reconstruction error, or the mean average error (Eq. (28)) which is the error between the actual input value y and the predicted value $\hat{y}$ averaged over the total number of predictions n.

$$MAE = \frac{1}{n} \sum |y - \hat{y}| \tag{28}$$

5.5. Reproducibility and parameters assignment

We use a NVIDIA GeForce RTX 3090 and Tensorflow 2.13 for our experiments. We designed three simple architectures to test our technique. As the target layer of our pruning has to be (i) fully connected and (ii) with ReLU activation, we included at least one of such layers before the output layer. Moreover, we assigned a large number of nodes to the pruned layer to report the performance drop when we removed a large number of nodes. The other previous layers are either fully connected with a limited number of nodes (in the case of the autoencoder and Dense network), or standard convolutional blocks composed of convolutional and pooling layers. Given the nature of

the dataset we exploited we did not delve into searching the best parameters for such architectures, but we just set them to achieve performance at least close to the state of the art. We also used a fixed number of epochs to train the models because our goal is to investigate the performance drop under different conditions, rather than looking for the best model. Finally, we used ReLU activations and Adam optimizer since they currently are the best combination in the state of the art for simple architectures such as ours. All tests have been repeated 20 times to display average and variance.

5.6. Experimental setup

The setup is the same for all networks: the models are trained for a fixed number of epochs on the MNIST and Housing datasets to get high accuracy for the CNN and the Dense network while looking for the small reconstruction loss for the autoencoder. After the training phase, we modify the weights of the fully connected layer to compress (the second last one for all models) to test the behavior of the network under the three scenarios we already defined. Specifically, we aim to avoid calculating the coefficients of the linear combinations, so we select a subset of weights and manually overwrite the remaining ones as linear combinations of the fixed weights with a random component. While this synthetically crafted layer allows us to test our hypothesis in a faster way, it is possible to find these linear combinations without imposing constraints on the weights.

The initial training of the network ensures that the subset of fixed weights approximates feasible values. The number of nodes we preserve equals the number of units in the previous layer. Since the weights in the pruned layer are synthetically forced to be linearly dependent or quasi-linear dependent, we set the number of weights used as generators equal to the rank of the matrix of vectors. To clarify, while pruning the nodes in the 128 units layer, given the previous layer has 32 units, we examine the 32×128 matrix of weights and we fix the initial 32 weights. The remaining 96 weights are overwritten as pseudo-random linear combinations of the 32 fixed weights. After this step, the layer is frozen and a rapid fine-tuning of the network adjusts the other weights to the new configuration. The resulting network allows us to apply our method to remove up to 96 of the units in this specific layer, with the number changing according to the number of units in the previous layer, and depending on the different architectures we test.

To analyze the first two scenarios, we force the weights of this layer to be positive during the training phase. Specifically, for the second scenario, we draw a vector of random values between $-\epsilon$ and ϵ to add after the linear combination. Finally, the same process is repeated for the third scenario without forcing the weights to be positive.

As mentioned earlier, during the first round of experiments, we confirmed that when the weights in the layer to prune have all the same sign, our method successfully removes the redundant units without introducing any performance loss. More interestingly, we aim to explore two scenarios: how the accuracy changes when the weights of the node to prune are quasi-linear combinations, and second, what happens when the signs of the weights can be discordant. In both cases, the updating step of our algorithm introduces an error that causes the network to deviate from its initial behavior.

Even if we only consider one layer as the target of our pruning approach, we proved in Theorem 1 that it can be applied to any fully connected hidden layer, and in particular that it can be applied iteratively on multiple dense adjacent layers while maintaining the semantic of the network (see Lemma 2).

5.7. Lumping a CNN

When a layer is configured with strictly positive weights and exploits ReLU activation, it cannot exhibit the non-linearity essential for the network to learn. Nevertheless, this setup can still be valuable for studying how the method behaves in the presence of nearly linear dependence. Specifically, we introduce an error ϵ, which depends on the magnitude of the weights in the layer after the initial training. Next, the weights in the layer are then overwritten as illustrated in Fig. 5. We test our method under different parameter configurations, in particular by using 16, 32, and 64 units in the layer before the one target for our pruning, and different values of ϵ. We also analyze the relationship between the number of weights contributing to the quasi-linear combination.

Fig. 6 shows how the accuracy decreases while pruning a layer of 128 nodes when the previous one contains either 16 units (top row) or 32 units (bottom row). We can prune up to 112 and 96 weights in the target layer. To assign a reasonable number to ϵ, we first compute the average magnitude of the weights in the layer, and then select ϵ as the 5%, 10% and 15% of the average magnitude. This epsilon represents the ray of the cylinder where we allow the weights to span. From our tests, it is clear that if the weights of the node to remove are dependent on a larger number of units, then removing such unit affects less the performance of the network, even when ϵ is up to 15% of the average magnitude of the weights.

Unfortunately, we witness the opposite behavior in the second scenario. Results are shown in Fig. 7. When the weights can be of opposite sign, the updating step introduces an error that diverges. Even if the effect seems to be mitigated when the previous layer presents a larger number of units, the cause is most probably related to the redundancy of the network with respect to the difficulty of the task, rather than an actual improvement. This can be seen by comparing the random pruning (in black), with the increasing number of vectors used for the linear combinations. Even though the results in this particular scenario do not seem promising, it is important to notice that our algorithm can be largely improved by designing a more intelligent strategy to remove the weights, taking into account how many components are discordant and which units are more or less relevant to the output.

5.8. Lumping an autoencoder

We conducted identical experiments using the autoencoder, with summarized results shown in Fig. 8. While the CNN architecture appeared less affected by our synthetic setup, the training and recovery phases for the autoencoder demanded 100 and 350 epochs, respectively.

Even if forcing the weights to be positive in the designed layer does not seem to affect significantly the initial performance of the model, the autoencoder cannot completely recover from manually setting up the weights as linear combinations, and this can be seen in the plot from the slight difference in the initial loss of the model. However, the marginal difference observed is negligible. Our primary focus lies in observing how the reconstruction loss deteriorates as we systematically remove nodes from our designed layer.

Similarly to the previous setup, the black line represents the pruning of positive random independent nodes, while the magenta line displays the pruning of an AE where the layer is not forced to have positive weights. The other results show the performance of our pruning by lumping. As expected, the loss of the model is not affected when we apply our pruning by lumping on positive weights. We still can observe an acceptable behavior when we introduce ϵ as 5% of the average magnitude of the weights in the layer. Finally, when we apply lumping on discordant signs, we obtain results consistent with those achieved in the CNN.

Fig. 10 displays a series of images depicting the reconstructed images and their degradation as we remove 0, 25, 50, 75, or 100% of the prunable nodes from our designed layer. This corresponds to removing 0, 28, 56, 84, and 112 units out of the 128 units initially present. Notably, the images degrade rapidly with the pruning of independent weights compared to leveraging our lumping method, even when allowing for an error margin of up to 15% of the average weight magnitude within the layer.

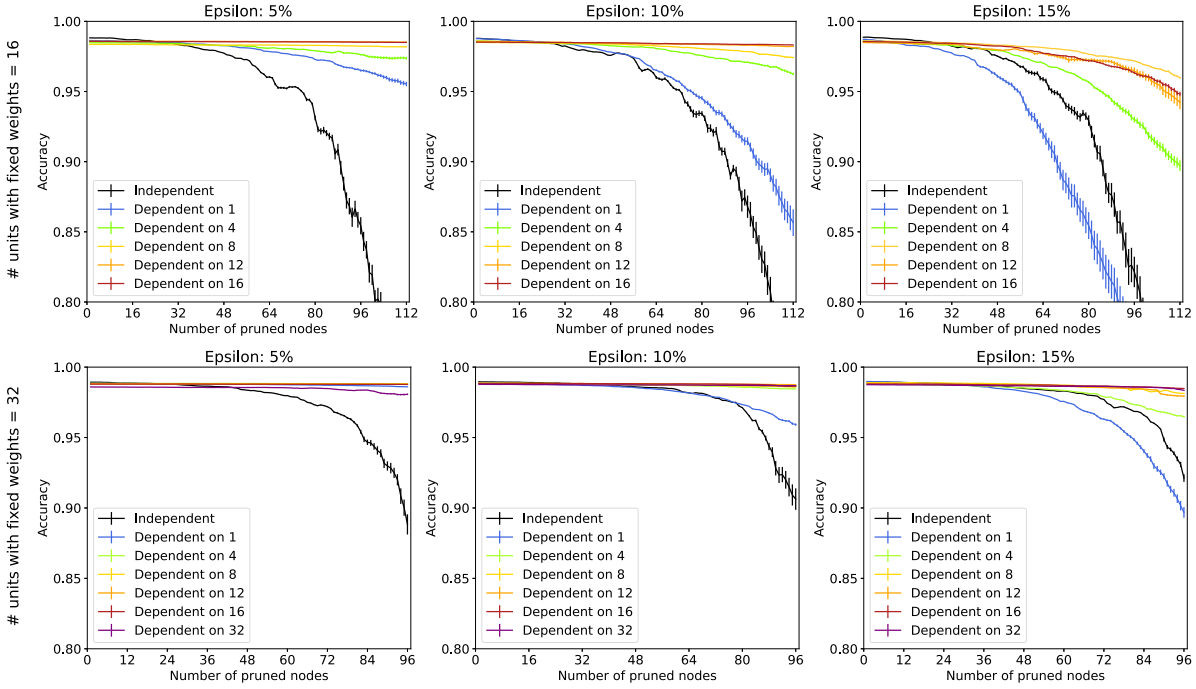


Fig. 6. Average accuracy of pruning CNN (trained on MNIST) when changing the number of input units and the value of ϵ.

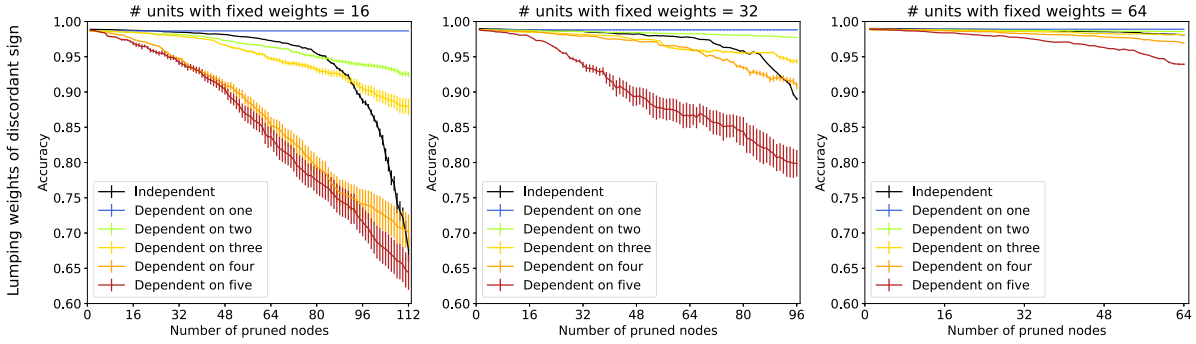


Fig. 7. Pruning CNN trained on MNIST. The plots show how the accuracy changes when pruning nodes whose vectors are linear combinations of one or more other nodes' weights when their outputs might have discord signs.

By applying our lumping on the target layer, we can compress the model from 205856 parameters to 116144 parameters overall, removing almost half of the parameters. Notice that, since our pruning by lumping is a node pruning method, this also results in greatly decreasing the number of FLOPs performed at inference time.

5.9. Lumping a dense network

The final series of experiments aims to validate the robustness of our analysis. For this reason, we test a different dataset (house price prediction) on a simple dense architecture. The results are summarized in Fig. 9. The results align consistently with those obtained from experiments conducted on the MNIST dataset. As we have already seen, lumping positive weights demonstrates minimal impact on the overall accuracy, even for *epsilon* up to 20% of the average weight magnitude in the layer. However, in contrast to expectations, the purple line representing the consolidation of positive weights exhibits unexpected perturbations, as compared to Fig. 8. Our investigation attributes this phenomenon to the version of the Tensorflow library. Specifically, we observed that altering the version of the library can slightly influence our results.

Finally, we can notice how the accuracy drops very fast when applying lumping to weights with discordant signs, however, since we set the number of dependant weights to 16, the results are consistent with those presented in Fig. 7.

6. Discussion

In the previous section, we explored the efficiency of our pruning method when the conditions are more similar to a real-world case. Indeed, even if our technique allows us to produce a compressed version of a network that produces the same exact output, we impose very hard constraints on the nature of the weights. In a more realistic setup, we assume the weights to be discordant, and we allow the linear combinations to include an error. From our experiments, we discovered a relationship between the number of units in the previous layer and the drop in accuracy. Specifically, our method has proved to be more robust when the weights have a higher number of dimensions. The reason behind this behavior can most likely be associated with the number of redundant nodes, as more complex networks probably included units almost irrelevant to the final output.

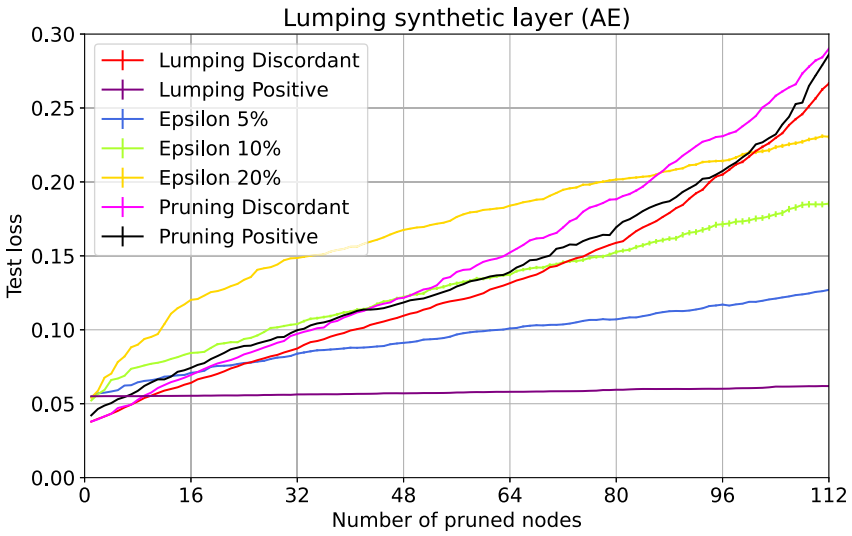


Fig. 8. Error plot of pruning via lumping applied to the autoencoder trained on MNIST.

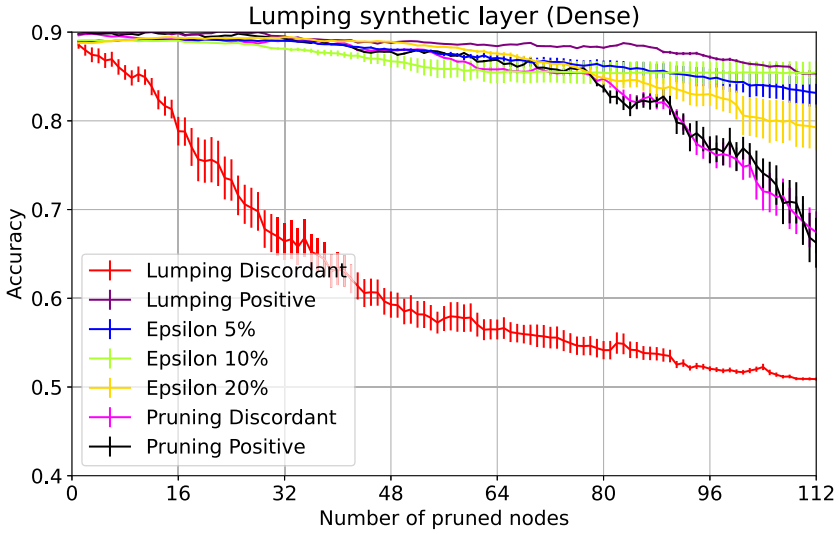


Fig. 9. Error plot of pruning via lumping applied to the Dense network for house price prediction.

Our pruning technique shares some similarities with low-rank approximations, but while the latter forces the matrix of weights to have a low rank during training time, we propose a post-training algorithm.

Another constraint we impose is the increasing number of nodes of the final layers, the target of our pruning. While the funnel-like architecture required for our pruning method to work the best is an unusual case for CNNs, autoencoders are the perfect target. In this case, our pruning method can potentially collapse all fully connected layers in the decoder except for the output layer (see Fig. 11).

In the future, we would like to approach an even more realistic scenario, by statistically analyzing the typical distribution of the weights in benchmark networks. This would allow us to modify our algorithm to target real weights, possibly considering removing the first less relevant nodes. Another technique that can be efficiently combined with our pruning is quantization. Finding linear dependence in a quantized network would be a much easier task, and especially for autoencoder architectures would achieve a significant degree of compression.

The actual implementation of these improvements would also allow us to compare our technique with baseline methods. A good candidate to prove the superiority of the proposed approach is the non-parametric Friedman test (Zamri, Azhar, Mansor, Alway, & Kasihmuddin, 2022; Zamri et al., 2024).

7. Conclusion

We propose a node pruning method that allows us to compress units in fully connected layers of a neural network without needing the original dataset or to re-train the model. By associating the nodes of a network with the states of a Markov chain, we can take advantage of state aggregation techniques distinctive to this area. In particular, we apply proportional lumpability on the layer, and we can provide a smaller version of the network that exhibits the same behavior, exactly as it works in lumping Markov Chains. Our strong requirements on the properties of the nodes led us to test our method also under relaxed constraints, to test its applicability in real-case scenarios. Even if the performance of the compressed version significantly drops when the requirement on the sign of the outputs is not met, our method shows high tolerance to quasi-linearity of the weights, which hints at its compatibility with other compression methods, such as low-rank tensor approximation. Regardless of the limitations of our method,

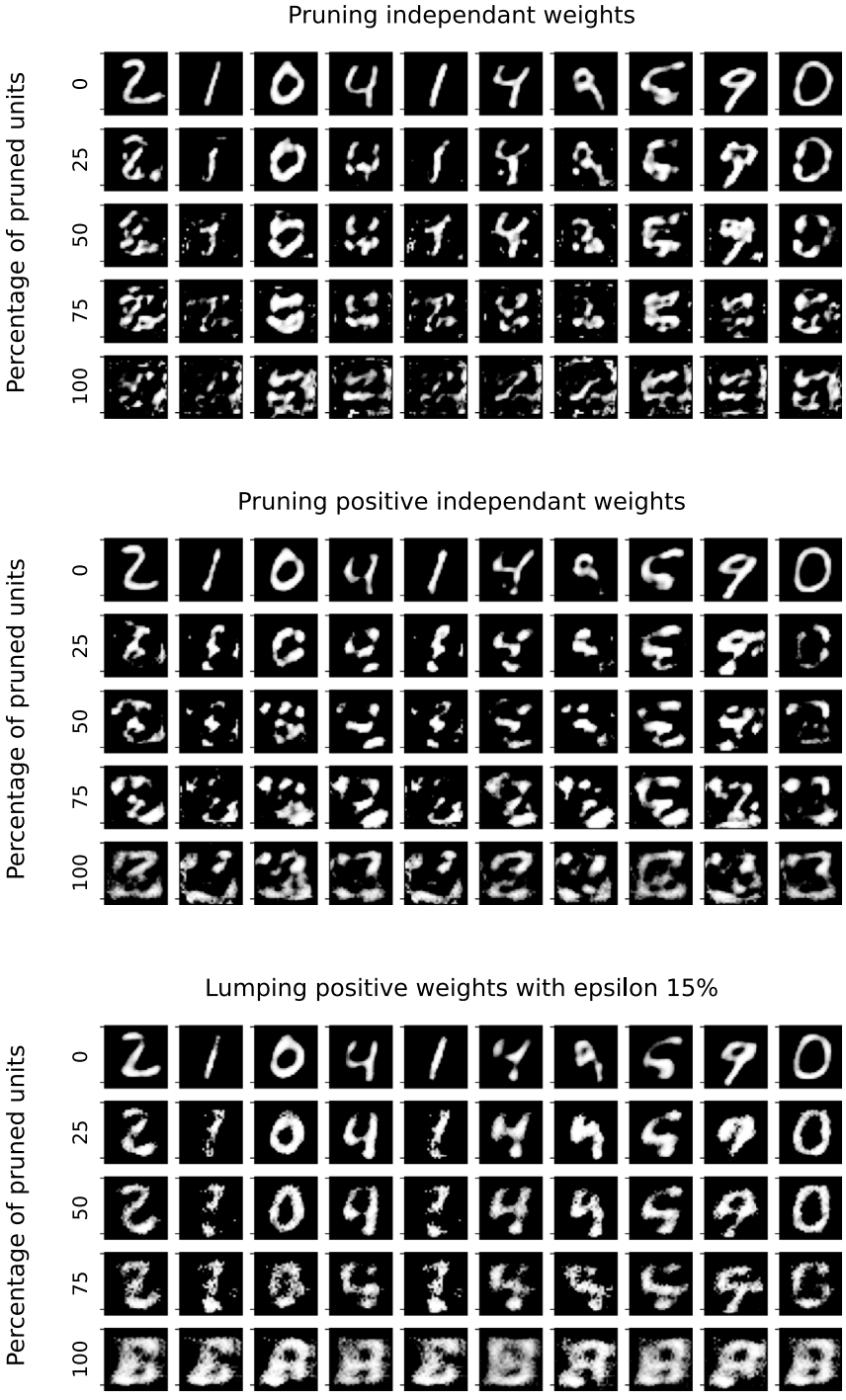


Fig. 10. Reconstructed images after pruning the autoencoder architecture selecting random nodes (top and middle) compared to our approximated pruning method (bottom).

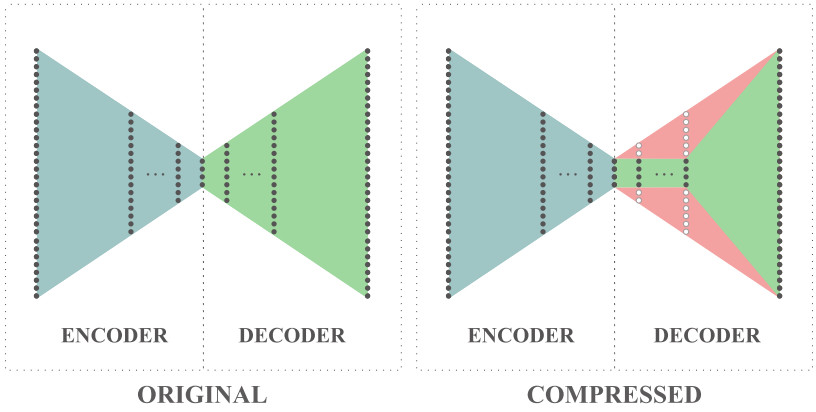


Fig. 11. Visualization of our pruning method applied to a simple autoencoder.

this work opens the door to a new research field where the aggregation techniques typical of performance evaluation are adopted in network compression, usually explored only by the machine learning community.

CRediT authorship contribution statement

Dalila Ressi: Investigation, Methodology, Supervision, Visualization, Writing – original draft, Conceptualization. **Riccardo Romanello:** Conceptualization, Formal analysis, Writing – original draft, Writing – review & editing. **Sabina Rossi:** Conceptualization, Formal analysis, Methodology, Supervision, Writing – original draft, Writing – review & editing. **Carla Piazza:** Conceptualization, Formal analysis, Methodology, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Dalila Ressi, Carla Piazza and Sabina Rossi reports financial support was provided by Government of Italy Ministry of Education University and Research.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work has been partially supported by the Project PRIN 2020 “Nirvana - Noninterference and Reversibility Analysis in Private Blockchains”, and by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

Abrar, S., & Samad, M. D. (2022). Perturbation of deep autoencoder weights for model compression and classification of tabular data. *Neural Networks*, *156*, 160–169.

Alzetta, G., Marin, A., Piazza, C., & Rossi, S. (2018). Lumping-based equivalences in Markovian automata: Algorithms and applications to product-form analyses. *Information and Computation*, *260*, 99–125.

Anticoli, L., Piazza, C., Taglialegne, L., & Zuliani, P. (2016). Towards quantum programs verification: from quipper circuits to qpmc. In *Reversible computation: 8th international conference, RC 2016, bologna, Italy, July (2016) 7-8, proceedings 8* (pp. 213–219). Springer.

Ashiquzzaman, A., Van Ma, L., Kim, S., Lee, D., Um, T.-W., & Kim, J. (2019). Compacting deep neural networks for light weight iot & scada based applications with node pruning. In *2019 international conference on artificial intelligence in information and communication* (pp. 082–085). IEEE.

B. Zoph, Q. V. Le (2017). Neural architecture search with reinforcement learning. In *Conference track proceedings of the 5th international conference on learning representations* (pp. 1–16). OpenReview.net.

Baker, B., Gupta, O., Naik, N., & Raskar, R. (2017). Designing neural network architectures using reinforcement learning. In *Conference track proceedings of the 5th international conference on learning representations*. OpenReview.net.

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., & Guttag, J. (2020). What is the state of neural network pruning? In *Proceedings of machine learning and systems: Vol. 2*, (pp. 129–146).

Bossi, A., Focardi, R., Macedonio, D., Piazza, C., & Rossi, S. (2004). Unwinding in information flow security. *Electronic Notes in Theoretical Computer Science*, *99*, 127–154.

Buchholz, P. (1994). Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, *31*, 59–75.

Bugliesi, M., Gallina, L., Marin, A., Rossi, S., & Hamadou, S. (2012). Interference-sensitive preorders for manets. In *Ninth international conference on quantitative evaluation of systems* (pp. 189–198). IEEE Computer Society.

Carroll, J. D., & Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika.*, *35*(3), 283–319.

Casagrande, A., Dreossi, T., & Piazza, C. (2012). Hybrid automata and ϵ-analysis on a neural oscillator. In E. Bartocci, & L. Bortolussi (Eds.), *EPTCS: Vol. 92, Proceedings first international workshop on hybrid systems and biology* (pp. 58–72). http://dx.doi.org/10.4204/EPTCS.92.5.

Dai, Z., Liu, H., Le, Q. V., & Tan, M. (2021). Coatnet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems*, *34*, 3965–3977.

Deng, L., Li, G., Han, S., Shi, L., & Xie, Y. (2020). Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, *108*(4), 485–532.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., & Fergus, R. (2014). Exploiting linear structure within convolutional networks for efficient evaluation. *Advances In Neural Information Processing Systems*, 1269–1277.

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, *20*(1), 1997–2017.

Eo, M., Kang, S., & Rhee, W. (2023). An effective low-rank compression with a joint rank selection followed by a compression-friendly training. *Neural Networks*, *161*, 165–177.

Frankle, J., & Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Conference track proceedings of the 7th international conference on learning representations* (pp. 1–42). OpenReview.net.

Gallina, L., Hamadou, S., Marin, A., & Rossi, S. (2011). A probabilistic energy-aware model for mobile ad-hoc networks. In *International conference on analytical and stochastic modeling techniques and applications* (pp. 316–330). Springer.

Grasedyck, L., Kressner, D., & Tobler, C. (2013). A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, *36*(1), 53–78.

Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *Conference track proceedings of the 4th international conference on learning representations*.

Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). Learning both weights and connections for efficient neural network. In *Annual conference on neural information processing systems 28: advances in neural information processing systems* (pp. 1135–1143).

Harshman, R. A., et al. (1970). Foundations of the parafac procedure: Models and conditions for an explanatory multimodal factor analysis.

He, J., Ding, Y., Zhang, M., & Li, D. (2022). Towards efficient network compression via few-shot slimming. *Neural Networks*, *147*, 113–125.

He, Y., Liu, P., Wang, Z., Hu, Z., & Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 4340–4349).

Hillston, J. (1994). *A compositional approach to performance modelling* (Ph.D. thesis), Department of Computer Science, University of Edinburgh.

Hillston, J., Marin, A., Piazza, C., & Rossi, S. (2013). Contextual lumpability. In *Proc. of valuetools 2013 conf.* (pp. 194–203). ACM Press.

Hillston, J., Marin, A., Piazza, C., & Rossi, S. (2021). Persistent stochastic non-interference. *Fundamenta Informaticae*, *181*(1), 1–35.

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

Hong, Q., Xiao, P., Fan, R., & Du, S. (2024). Memristive neural network circuit design based on locally competitive algorithm for sparse coding application. *Neurocomputing*, Article 127369.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., et al. (2019). Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 1314–1324).

Hu, H., Peng, R., Tai, Y.-W., & Tang, C.-K. (2016). Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5 mb model size. arXiv preprint arXiv:1602.07360.

Kemeny, J. G., & Snell, J. L. (1976). *Finite Markov chains*. Springer.

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., et al. (2020). Big transfer (bit): General visual representation learning. In *European conference on computer vision* (pp. 491–507). Springer.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, *25*.

LeCun, Y., Denker, J. S., & Solla, S. A. (1990). Optimal brain damage. *Advances In Neural Information Processing Systems*, 598–605.

Li, H., Yue, X., Wang, Z., Chai, Z., Wang, W., Tomiyama, H., et al. (2022). Optimizing the deep neural networks by layer-wise refined pruning and the acceleration on fpga. *Computational Intelligence and Neuroscience*.

Lin, M., Chen, Q., & Yan, S. (2014). Network in network. In *Conference track proceedings of the 2nd international conference on learning representations*.

Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., et al. (2020). Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 1529–1538).

Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., et al. (2019). Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 2790–2799).

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., & Zhang, C. (2017). Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision* (pp. 2736–2744).

Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G. G., & Tan, K. C. (2021). A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*.

Ma, N., Zhang, X., Zheng, H.-T., & Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision* (pp. 116–131).

Marin, A., Piazza, C., & Rossi, S. (2019). Proportional lumpability. In *Lecture notes in computer science*: *Vol. 11750*, *Formal modeling and analysis of timed systems - 17th international conference, FORMATs 2019, proceedings* (pp. 265–281). Springer.

Marin, A., Piazza, C., & Rossi, S. (2022). Proportional lumpability and proportional bisimilarity. *Acta Informatica*, *59*(2), 211–244.

Molchanov, P., Mallya, A., Tyree, S., Frosio, I., & Kautz, J. (2019). Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 11264–11272).

Mou, X., Tang, J., Lyu, Y., Zhang, Q., Yang, S., Xu, F., et al. (2021). Analog memristive synapse based on topotactic phase transition for high-performance neuromorphic computing and neural network pruning. *Science Advances*, *7*(29), Article eabh0648.

Novikov, A., Podoprikhin, D., Osokin, A., & Vetrov, D. P. (2015). Tensorizing neural networks. *Advances in Neural Information Processing Systems*, *28*.

Piazza, C., & Rossi, S. (2021). Reasoning about proportional lumpability. In *Lecture notes in computer science*: *Vol. 12846*, *QEST* (pp. 372–390). Springer.

Pistellato, M., Bergamasco, F., Bigaglia, G., Gasparetto, A., Albarelli, A., Boschetti, M., et al. (2023). Quantization-aware nn layers with high-throughput fpga implementation for edge ai. *Sensors*, *23*(10), 4667.

Prabhakar, P. (2022). Bisimulations for neural network reduction. In *International conference on verification, model checking, and abstract interpretation* (pp. 285–300). Springer.

Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision* (pp. 525–542). Springer.

Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Chen, X., et al. (2021). A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys*, *54*(4), 1–34.

Ressi, D., Pistellato, M., Albarelli, A., & Bergamasco, F. (2022). A relevance-based cnn trimming method for low-resources embedded vision. In *International conference of the Italian association for artificial intelligence* (pp. 297–309). Springer.

Ressi, D., Romanello, R., Piazza, C., & Rossi, S. (2022). Neural networks reduction via lumping. In *International conference of the Italian association for artificial intelligence* (pp. 75–90). Springer.

Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., & Chen, L. (2018a). Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE conference on computer vision and pattern recognition* (pp. 4510–4520). Computer Vision Foundation / IEEE Computer Society.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018b). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4510–4520).

Schweitzer, P. (1984). Aggregation methods for large Markov chains. In *Proc. of the international workshop on computer performance and reliability* (pp. 275–286). North Holland.

Shang, R., Li, W., Zhu, S., Jiao, L., & Li, Y. (2023). Multi-teacher knowledge distillation based on joint guidance of probe and adaptive corrector. *Neural Networks*, *164*, 345–356.

Shimoda, M., Sada, Y., & Nakahara, H. (2019). Filter-wise pruning approach to fpga implementation of fully convolutional network for semantic segmentation. In *Applied reconfigurable computing: 15th international symposium, ARC 2019, darmstadt, Germany, April (2019) 9–11, proceedings 15* (pp. 371–386). Springer.

Sproston, J., & Donatelli, S. (2004). Backward stochastic bisimulation in csl model checking. In *First international conference on the quantitative evaluation of systems, 2004. QEST 2004. proceedings* (pp. 220–229). IEEE.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1–9).

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2818–2826).

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105–6114). PMLR.

Tan, M., & Le, Q. V. (2021). Efficientnetv2: Smaller models and faster training. *Vol. 139*, In *Proceedings of the 38th international conference on machine learning* (pp. 10096–10106). PMLR.

Tan, C. M. J., & Motani, M. (2020). Dropnet: Reducing neural network complexity via iterative pruning. In *International conference on machine learning* (pp. 9356–9366). PMLR.

Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika.*, *31*(3), 279–311.

Wang, N., Liu, T., Wang, J., Liu, Q., Alibhai, S., & He, X. (2022). Locality-based transfer learning on compression autoencoder for efficient scientific data lossy compression. *Journal of Network and Computer Applications*, *205*, Article 103452.

Wang, Z., Xie, X., & Shi, G. (2021). Rfpruning: A retraining-free pruning method for accelerating convolutional neural networks. *Applied Soft Computing*, *113*, Article 107860.

Wen, S., Wei, H., Yan, Z., Guo, Z., Yang, Y., Huang, T., et al. (2019). Memristor-based design of sparse compact convolutional neural network. *IEEE Transactions on Network Science and Engineering*, *7*(3), 1431–1440.

Xiao, L., Bahri, Y., Sohl-Dickstein, J., Schoenholz, S., & Pennington, J. (2018). Dynamical isometry and a mean field theory of cnns: How to train 10 000-layer vanilla convolutional neural networks. In *International conference on machine learning* (pp. 5393–5402).

Yu, R., Li, A., Chen, C.-F., Lai, J.-H., Morariu, V. I., Han, X., et al. (2018). Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 9194–9203).

Yu, J., Wang, Z., Vasudevan, V., Yeung, L., Seyedhosseini, M., & Wu, Y. (2022). Coca: Contrastive captioners are image-text foundation models. *Transaction on Machine Learning Research*, *2022*, 1–20.

Zamri, N. E., Azhar, S. A., Mansor, M. A., Alway, A., & Kasihmuddin, M. S. M. (2022). *Applied Soft Computing*, *126*, Article 109312.

Zamri, N. E., Mansor, M. A., Kasihmuddin, M. S. M., Sidik, S. S., Alway, A., Romli, N. A., et al. (2024). A modified reverse-based analysis logic mining model with weighted random 2 satisfiability logic in discrete hopfield neural network and multi-objective training of modified niched genetic algorithm. *Expert Systems with Applications*, *240*, Article 122307.

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 6848–6856).

Zhou, C., Wang, H., Zhou, S., Yu, Z., Bandara, D., & Bu, J. (2023). Hierarchical knowledge propagation and distillation for few-shot learning. *Neural Networks*, *167*, 615–625.