Progress in Artificial Intelligence
https://doi.org/10.1007/s13748-024-00339-8

REGULAR PAPER

Optimizing sparse topologies via competitive joint unstructured neural networks

Federico A. Galatolo[1] · **Mario G. C. A. Cimino**[1]

Received: 2 November 2022 / Accepted: 4 August 2024

Abstract
A major research problem of artificial neural networks (NNs) is to reduce the number of model parameters. The available approaches are pruning methods, consisting of removing connections of a dense model, and natively sparse models, based on training sparse models using meta-heuristics to preserve their topological properties. In this paper, the limits of both approaches are discussed. A novel hybrid training approach is developed and experimented. The approach is based on a linear combination of sparse unstructured NNs, which are joint because they share connections. Such NNs dynamically compete during the optimization, since the less important networks are iteratively pruned until the most important network remains. The method, called Competitive Joint Unstructured NNs (CJUNNs), is formalized with an efficient derivation in tensor algebra, which has been implemented and publicly released. Experimental results show its effectiveness on benchmark datasets compared to structured pruning.

Keywords Artificial neural networks · Neural network pruning · Unstructured topology

1 Introduction and background

1.1 Introduction

In the last years, artificial neural networks (NNs) have been the object of great improvements and major milestones. However, the significant rise of number of parameters for increasing problem complexity remains a fundamental challenge for NN modeling. In the literature, a variety of solutions has been proposed to overcome this issue [1]. Overall, the available approaches can be divided into two major categories: *pruning methods* and *natively sparse models*. The first approach consists of pruning parameters of a dense model, which can be done iteratively, i.e., during or at the end of the training. The second approach consists of training sparse models using meta-heuristics to preserve topological properties [2]. Hybrid categories have been also proposed [3].

Pruning methods require an initial dense model, which can be complex and time-consuming. Natively sparse models require complex meta-heuristics to preserve their topological properties, and they can be difficult to optimize. We propose a new pruning method that can overcome the limitations of existing methods and provide a more efficient and effective solution.

In this paper, a novel hybrid approach is developed and experimented with, called *Competitive Joint Unstructured NN (CJUNN)*. It generates a sparse UNN model, via many Competing UNNs. The term *Joint* refers to the fact that the UNNs share a part of their weights. The term *Competing* relates to the importance assigned to each network when computing their combined output. The importance coefficients take part in the optimization and determine the pruning of the less important network during the training iterations until the most important network is finally generated. To design an efficient derivation in tensor algebra of this approach, able to formalize the optimization task via backpropagation and stochastic gradient descent, in this paper a proper tensorial representation is developed and publicly released. Experimental results carried out on benchmark data, show that CJUNN overcomes structured pruning approaches in terms of both accuracy and parametric simplicity.

The paper is structured as follows. The remainder of this section covers the background. Section 2 details the development of the proposed model. Experimental studies are

✉ Federico A. Galatolo
federico.galatolo@ing.unipi.it

[1] Department of Information Engineering, University of Pisa, Largo Lazzarino 1, 56122 Pisa, Italy

Published online: 16 September 2024

illustrated and discussed in Sect. 3. Finally, conclusions and future work are drawn in Sect. 4.

1.2 Mathematical preliminaries

To formalize the architectural solutions, let us define the basic concepts and notations. In structured architectures, a layer of neurons with i inputs and o outputs can be described by: (i) its weight matrix $\boldsymbol{W} \in \mathbb{R}^{i \times o}$, whose element $\boldsymbol{W}_{ij}$ represents the weight from the i-th input neuron to the j-th output neuron; (ii) the bias vector $\boldsymbol{b} \in \mathbb{R}^o$, whose element b_j represents the bias of the j-th output neuron. By denoting with $\varphi(x)$ the activation function, for a given input $\boldsymbol{x} \in \mathbb{R}^o$ the layer output is computed as:

$$\boldsymbol{y} = \varphi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{1}$$

A Multi Layer Perceptron (MLP) is made of stacked NNs layers. The Stochastic Gradient Descent (SGD) algorithm, which is widely used to train NNs, is a stochastic approximation of the gradient descent optimization calculated over randomly selected subsets of data [4]. In supervised training settings, a set of desired input–output pairs $\hat{\mathbb{X}}$, $\hat{\mathbb{Y}}$ and an error function $\epsilon(\boldsymbol{y}, \boldsymbol{fy})$ are given. By denoting with p the parameters of an NN, and with $f(x)$ its input–output function, then in SGD, the mean gradient of the error function with respect to the NN parameters is given in Eq. 2. It is worth noting that SGD samples Eq. 2, which is actually infeasible to compute.

$$\Delta_p E_{(\boldsymbol{fx}, \boldsymbol{fy}) \in (\hat{\mathbb{X}}, \hat{\mathbb{Y}})}[\epsilon(f(\boldsymbol{fx}), \boldsymbol{fy})] \tag{2}$$

To compute the minimum-error gradient, the *backpropagation* algorithm is used [5]. The parameters are then updated following the trajectory determined by this algorithm.

2 Literature analysis

In this paper, our focus lies primarily on sparsification and pruning techniques for model compression. Sparsification and pruning offer compelling solutions to improve computational and memory efficiency while maintaining or even enhancing the performance of deep neural networks.

Sparsification techniques aim to reduce the representational complexity of models by selectively utilizing only a subset of dimensions in high-dimensional parameter spaces. By zeroing out certain model parameters, sparsification effectively reduces the overall size of the network, leading to more efficient models. This approach allows for significant savings in terms of memory requirements and computational resources.

Pruning, on the other hand, focuses on removing redundant or insignificant connections within dense neural networks without significantly impacting overall performance. Through a process of ranking model parameters based on their importance or relevance, pruning identifies and eliminates the least crucial connections. By selectively pruning these connections, the resulting pruned network maintains its functionality while further reducing its size and computational requirements.

Both sparsification and pruning techniques offer valuable means for model compression, enabling users to achieve significant reductions in the number of parameters and computations without sacrificing the network's overall performance.

In this paper, we propose the Competitive Joint Unstructured Neural Networks (CJUNNs) as a novel approach to address the problem of reducing the number of model parameters in Artificial Neural Networks (NNs). To establish the effectiveness of CJUNNs, we compare our model specifically with two prominent techniques in the field: dense neural network pruning and lottery ticket pruning. Here, we provide a rationale for selecting these two approaches for comparison, highlighting the distinct characteristics and advantages of CJUNNs.

Dense neural network pruning methods have gained significant attention due to their ability to reduce model complexity by removing connections in a dense neural network. These methods typically rely on heuristics to identify and remove less important connections based on weight magnitudes, gradients, or other criteria. By comparing CJUNNs with dense neural network pruning, we aim to investigate whether our proposed method can achieve comparable or superior performance in terms of model compression while maintaining competitive accuracy levels. Furthermore, by providing a direct comparison, we can assess the effectiveness of CJUNNs in addressing the limitations associated with dense pruning methods, such as potential loss of network connectivity or the need for retraining after pruning.

Lottery ticket pruning has emerged as an approach that identifies sparse subnetworks within an initially dense neural network. It demonstrates that training a sparse network from scratch can achieve comparable accuracy to the original dense network, provided that the sparse network has the right initialization. Comparing CJUNNs with lottery ticket pruning enables us to evaluate the performance of our proposed method against an existing state-of-the-art technique that focuses on training sparse models from the ground up. By doing so, we can assess whether CJUNNs offer advantages in terms of training efficiency and the ability to maintain accuracy during training compared to the lottery ticket pruning approach.

An important contribution of CJUNN lies in its capacity to generate a sparsely connected Unstructured Neural Network (UNN) model through a competitive process involving multiple networks in competition. The sharing of connec-

tions facilitates dynamic competition during the optimization process, wherein less significant networks are progressively pruned, leading to the emergence of the most pivotal network. By amalgamating the strengths inherent in sparse unstructured networks and competition-based pruning, CJUNN presents a more efficient and effective solution for reducing the parameter count in NNs.

Another noteworthy contribution of CJUNN entails the development of an efficient derivation in tensor algebra to formalize the optimization task through the utilization of backpropagation and stochastic gradient descent. This tensorial representation not only expedites the implementation of CJUNN but also enables the model to scale effortlessly by leveraging parallel computing platforms.

By contrasting CJUNNs with both dense neural network pruning and lottery ticket pruning, we aim to comprehensively evaluate the unique characteristics and advantages of our proposed method. Through this comparative analysis, we can provide insights into the strengths and limitations of CJUNNs.

2.1 Dense neural networks pruning

NN pruning is a method based on removing superfluous connections without a significant impact on the overall performance. To carry out the pruning after the training process, the parameters of the model are ranked by their L1 norm, and a target level of pruning p is set. By denoting with N the number of parameters, the $N \cdot p$ parameters with the lowest rank then are pruned. It has been shown that dense overparameterized models can be pruned this way without any considerable decrease in performance [6].

Another pruning approach proposed in the literature is based on the so-called *lottery ticket hypothesis*: dense, randomly initialized, feed-forward networks contain subnetworks that, when trained in isolation, achieve test accuracy comparable to the original network in a similar number of iterations [7]. The related approach consists of the following steps: (i) train a dense network; (ii) set a target level of pruning p; (iii) rank the model parameters by their L1 norm; (iv) find a mask m that masks $N \cdot p$ parameters; (v) restore the parameters to their initial state; (vi) train again the model masking the pruned parameters via m.

A different approach consists of using *gate variables* representing the probability of certain weights to be pruned [8]. The *gate variables* approach consists of defining, for each layer, a matrix $\boldsymbol{G}$ with the same size of $\boldsymbol{W}$ where $\boldsymbol{G}_{i,j}$ represents the probability of the weight $\boldsymbol{W}_{i,j}$ to be pruned. For each forward pass, the binary mask matrix $\boldsymbol{G_S}$ is computed, where each element $\boldsymbol{G_S}_{i,j}$ is treated as a random sample of the Bernoulli random variable with $p = \boldsymbol{G}_{i,j}$. Finally, the masked weight matrix $\boldsymbol{W}_s$ is used instead of the dense weight matrix $\boldsymbol{W}$:

$$\boldsymbol{W}_s = \boldsymbol{W} \cdot \boldsymbol{G_S} \tag{3}$$

To promote the sparsity and to avoid the gate values from converging to 0.5 the authors also suggested using l_1 and l_2 regularization. This approach can approximately reach a compression rate of 10 to 20 without compromising the performances of some widespread models. The downside is that backpropagation cannot be used to train $\boldsymbol{G}$: for this reason, the authors use Monte Carlo methods.

Another method to prune a dense model was proposed by NVIDIA [9]. The core concept of their approach revolves around approximating the significance of each parameter and subsequently pruning those that hold lesser importance. Given a dataset $\mathbb{D}$ and an error function E, the authors define the importance I_m of a parameter w_m as the squared difference of the error of the network with and without that parameter:

$$I_m = (E(\mathbb{D}, \boldsymbol{W}) - E(\mathbb{D}, \boldsymbol{W}|w_m = 0))^2 \tag{4}$$

A direct computation of I_m is infeasible because it requires evaluating the network many times as the number of elements of $\boldsymbol{W}$. For this reason, in the paper, the authors suggest using the first-order Taylor expansion of I_m:

$$I_m{}^{(1)} = \left(\frac{\partial E(\mathbb{D}, \boldsymbol{W})}{\partial w_m} w_m \right)^2 \tag{5}$$

This formulation is very convenient because the gradient $\frac{\partial E(\mathbb{D}, \boldsymbol{W})}{\partial w_m}$ is available from the backpropagation. The authors showed how this first-order approximation is highly correlated to the actual importance values; they achieved up to 40% FLOPS reduction and up to 30% parameters pruning, without any significant increase in the error rate of the networks.

Finally, a novel approach to network pruning was proposed by Tanaka et al. [10]. Focusing on identifying highly sparse trainable subnetworks at initialization without the need for training or examining the data. The authors address the limitations of existing pruning algorithms at initialization, specifically the issue of layer collapse, which occurs when an entire layer is pruned prematurely, making the network untrainable. To overcome this challenge, they present a theory-driven algorithm called Iterative Synaptic Flow Pruning (SynFlow).

SynFlow is motivated by a conservation law that explains the mechanism behind layer collapse and offers insights into its avoidance. The algorithm preserves the total flow of synaptic strengths throughout the network at initialization while imposing a sparsity constraint. Remarkably, SynFlow operates independently of the training data and consistently competes with or surpasses state-of-the-art pruning algorithms at initialization across various models (such as VGG

and ResNet), datasets (including CIFAR-10/100 and Tiny ImageNet), and sparsity constraints (up to 99.99 percent).

The data-agnostic nature of SynFlow challenges the prevailing paradigm that emphasizes the use of data to quantify the importance of synapses during initialization. Instead, it demonstrates that highly sparse trainable subnetworks can be identified without relying on the training process or data analysis.

2.2 Natively sparse models

Another approach to reduce the number of parameters of NNs is to use natively sparse models. The main issue with using sparse models is how to ensure both high sparsity and connectivity. One of the most recent architectures facing this problem is based on the so-called *X-Nets* [11]. The *X-Nets* model the connections between neurons in a NN as an *expander graph*. The expander graph is a well-known sparse graph model that has been proven to ensure high connectivity between its nodes. In their research work, the authors experimentally proved that the resulting NNs maintain the same properties of expander graphs. The authors also showed that the sparse models outperform the classical counterpart by up to 4% in accuracy, while having a reduction factor of number of parameters of more than 10.

In another research work, it has been shown that it is possible to achieve the same sparsity and connectivity of the expander graphs without relying on them. Authors from MIT proposed a novel algorithm called *RadiX-Nets* [12]. *RadiX-Nets* are topologically very different from *X-Nets*, and do not rely on an underlying expander graph. The proposed algorithm can create NNs with the same topological properties. The authors of *RadiX-Nets* mathematically proved their claims. The main advantage of natively sparse models is that they constrain the weight matrices to sparse connectivity patterns before training. Therefore, it is possible to exploit memory and runtime efficiently in the training phase. In contrast, with pruning techniques it is necessary to train the dense model, inherently limiting the size of that can be compressed. However, the corpus of research on natively sparse models is not currently mature and homogeneous on the subject [12]. From one side, the collective body of research in this field mutually corroborates the assertion that sparse neural networks can train to the same arbitrary degree of precision as their dense counterparts. However, while the reduced training time of sparse neural nets can be attributed to having fewer parameters, there is no clear reason why sparse networks should demonstrate the same expressive power as dense counterparts. Moreover, there is still a lack of consensus on what is meant when discussing the concept of expressive power, when describing the abilities and limitations of neural networks rigorously. As a consequence, researchers in the field propose some conjecture based on the experimental findings, which are intended to prove to direct future research [12]. In this paper, a hybrid model is proposed, in which natively sparse networks are also pruned. The fundamental problem is how to generate sparse networks without biasing the network. Concerning this problem, the proposed approach is based on a combination of multiple sparse networks, which are progressively pruned by their contributions to the output. This self-organized competition is the basis of the proposed method.

In a related study by another group of researchers, a novel technique called Sparse Unbalanced GAN (STU-GAN) is introduced to address the challenges associated with training sparse generative adversarial networks (GANs) from scratch. Training GANs is a notoriously difficult and unstable process, thus the techniques employed to train natively sparse GANs can provide valuable insights into naturally sparse model training. The technique proposed by [13] aims to achieve superior training efficiency while maintaining competitive performance, without relying on pre-training or dense training steps, which are resource-intensive and may limit the applicability of GANs in resource-limited scenarios. The authors propose a sparse-to-sparse training procedure, wherein the GAN is initially initialized with a highly sparse generator and a denser discriminator. Throughout the training process, the parameter space of the sparse generator is dynamically explored to enhance its capacity progressively, while adhering to a fixed small parameter budget. This approach significantly improves the expressibility of the sparse generator, mitigating the training instability typically observed in sparse unbalanced GANs.

The authors conducted extensive experiments using state-of-the-art GAN architectures, such as BigGAN and SNGAN, on datasets like CIFAR-10 and ImageNet. The results demonstrated the effectiveness of the proposed STU-GAN technique. Notably, STU-GAN surpassed the performance of dense BigGAN on CIFAR-10 using only an 80% sparse generator and a 70% sparse discriminator. This achievement highlights the potential of training sparse GANs from scratch as a more efficient alternative to relying on parameters inherited from pre-trained GANs. The end-to-end trainability of STU-GAN allows for streamlined training and inference processes, addressing the computational and memory requirements associated with GANs.

The findings of this study are in line with the growing interest in exploring methods to reduce the number of parameters in neural networks. The use of native sparsity in GAN models, as demonstrated by STU-GAN, offers the advantage of constraining weight matrices to sparse connectivity patterns before training. This constraint enables efficient utilization of memory and runtime during the training phase. The findings presented in this paper provide valuable insights and directions for future research endeavors in this field, shedding light on the advantages of combining native sparsity and

pruning techniques to generate hybrid models with improved efficiency and performance.

2.3 Hybrid models

To create unstructured sparse NN models that do not rely on backpropagation, recently the *Mesh Neural Networks* (MNNs) have been proposed [3]. MNNs are NN models in which neurons can be connected in unstructured topology. An MNN with i inputs, h hidden nodes and o outputs is defined over an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{(i+h+o)\times(i+h+o)}$, in which each element $\boldsymbol{A}_{ij}$ represents the connection weight between node i and j and a state transition function $\boldsymbol{s_t} = \varphi(\boldsymbol{s_{t-1}A})$. The state $S_t \in \mathbb{R}^{(i+h+o)}$ represents the output states of the nodes at the time t. In MNNs the gradient of the current state with respect to the adjacency matrix $\Delta_{\boldsymbol{A}}\boldsymbol{s_t}$ can be directly computed in a forward pass using the Forward-Only Propagation (FOP) algorithm. MNNs showed state-of-the-art results on benchmark datasets using a low number of neurons. A *lottery-ticket* pruning method has been also applied to MNNs: pruned MNNs showed to achieve state-of-the-art MLP performances on benchmark datasets, such as MNIST or Fashion-MNIST, while removing up to 85% of the weights. A drawback of MNNs is that they need large adjacency matrix operations that must be supported by a framework implementation that efficiently exploits the hardware resources (via memory caching and highly parallel computation). Currently, the experimentation of MNNs on very large datasets can be carried out on specific hardware. For this reason, in this paper, a different hybrid approach is developed, which is based on explicit topology representation in terms of connection matrix.

Our proposed method combines the advantages of both pruning methods and natively sparse models and dynamically competes among the sparse unstructured NNs during optimization to create an optimized model with reduced number of parameters. This work contributes to the existing literature and practice by proposing a novel hybrid training approach that addresses the limitations of existing methods and provides an effective solution for reducing the number of model parameters. In the next sections is also presented an efficient derivation in tensor algebra for the CJUNN method. Finally, the experimental results of the proposed method demonstrate its effectiveness on benchmark datasets and show that it outperforms structured pruning methods, providing further evidence of its contribution to the field.

3 Development of the proposed model

The proposed Competitive Joint Unstructured Neural Networks (CJUNNs) is an unstructured model, in which neurons are allowed to be connected in any topology. Differently from MNNs, instead of relying on an implicit topology defined over an adjacency matrix and on a state transition function, in CJUNNs, the topology is explicitly defined in a connection matrix. Unlike natively sparse models, in which an explicit criterion is established at design time for sparsity, in CJUNNs the sparse structure is not defined a priori: the model is initialized with n parallel sparse networks with shared weights, whose outputs are linearly combined through their current relative importance. During the training process, two mechanisms provide both the exploitation of the best networks and exploration of the search space: (i) network competition: at each early stop trigger, the less important network is pruned; (ii) topological mutation: topological variation of the low-entropy neurons. In contrast to [9] the importance of a whole network is defined instead of a single weight, and instead of relying on a definition of importance based on the error function the proposed competition mechanism is based on learnable importance coefficients α. Diversely from [11] and [12] the network topology is not based on an auxiliary graph model, but local topological mutations are stochastically triggered on the low-entropy neurons.

3.1 Forward propagation model

This section formally defines the forward propagation model of the CJUNN-based architecture. Let us denote by i the number of inputs, h the number of hidden nodes, o the number of outputs, n the number of networks, and m the maximum number of incoming connections. Let us define a weight matrix $\boldsymbol{W} \in \mathbb{R}^{(i+h)\times(h+o)}$, a connection tensor $\mathsf{C} \in \mathbb{R}^{n\times(h+o)\times m}$ and the network importance coefficients vector $\boldsymbol{\alpha} \in \mathbb{R}^n$. Each element $\boldsymbol{W}_{i,j}$ of the weight matrix represents the weight of the connection between the i-th neuron and the j-th neuron. Each vector $\mathsf{C}_{i,j,:} \in \mathbb{R}^m$ of the connection tensor represents the indices of the incoming neurons connected to the neuron j in the network i. It is also important to notice that two additional virtual input neurons are added: the *zero-bias* neuron, whose output value is fixed to zero, and the *one-bias* neuron, whose output value is fixed to one. The *zero-bias* neuron acts as a non-connection virtual node, whereas the *one-bias* neuron, when multiplied by its corresponding weight, acts as a classical linear bias. The neuron output state is updated following the natural order of C. As a consequence, a neuron η can have as incoming connections the neurons from 0 to $\eta - 1$. This constraint does not limit the network structure, because it is well known that recurrent nodes can be unfolded and transformed into forward-only nodes, with a new forward step working per each instant of time of the finite response [14].

Figure 1 shows a representative example of a CJUNN-based model with its connection tensor. In particular, two competitive joint unstructured networks are represented in Fig. 1a called CJUNN_1 and CJUNN_2. Shared connections

are represented with a solid line, whereas independent connections are represented with dashed and dotted lines, for CJUNN_1 and CJUNN_2, respectively. The three input nodes i_0, i_1, and i_2 are enclosed by squares. In particular, the *one-bias* and the *zero-bias* special nodes are represented by two corresponding squares, enclosing the value 1 and empty, respectively. The other nodes are represented by circles.

A bold incremental number is also represented on the top-right of each node: it is an absolute ordering of all nodes serving as a tensor index: first the input nodes, second the special input nodes, third hidden nodes, and finally, the output nodes. Due to the unstructured topology, the ordering between hidden nodes is not based on layers: a first group of hidden nodes is fed only by input nodes, a second group is fed also by nodes of the first group, and so on. In general, an existing connection from node i to node j establishes an ordering i-then-j. This unstructured architecture generalizes, and then can also represent the conventional multi-layer perceptron.

Figure 1b shows the correspondent connection tensor, made by two matrices: the front matrix for CJUNN_1, and the back matrix for CJUNN_2. Each matrix row represents a non-input node with its input connections. In particular, the first row of the CJUNN_1 matrix refers to node n_0, which is fed by nodes 0, 1, and 2, following the solid and dashed connections arriving at n_0. Since node 3 is not connected to n_0, it is connected to node 4 (non-connection). As a consequence, the first row is "0 1 2 4". The second row refers to n_1, which is fed by nodes 1,2, and 3. Again, missing connections are represented by node 4 at the end. The third row refers to o_0, i.e., the first output node, which is also the third non-input node, and then also represented as n_2 in brackets. n_2 is fed by nodes 5 and 6, followed by the conventional non-connections "4". Finally, the fourth row refers to n_3, which is fed by 3 and 6. Similarly, considering the CJUNN_2 matrix (i.e., solid and dotted connections), n_0 is fed by 0, 1, 3, n_1 is fed by 0, 2, n_2 is fed by 5, 6, and n_3 is fed by 3, 5, 6.

Let us denote by $G(\mathsf{X}, \mathsf{Y})$ the *gather* operator that uses the elements of the tensor Y as indices to select the elements from the tensor X. Let us denote by $E(\mathsf{X}, n)$ the *expand* operator that replicates n times the X tensor along a new dimension. The output of the CJUNN-based network is computed as follows: given an input $\boldsymbol{x}$, and initializing the current hidden state $\boldsymbol{H} \in \mathbb{R}^{n\times(h+o)}$ to 0, update the state of each neuron η in order from the one with the lowest index to the one with the highest $\eta \in \{1, 2, \cdots, h+o\}$.

For each neuron η the presynaptic state $\boldsymbol{T}$ is computed by stacking the current hidden state $\boldsymbol{H}$ and the expanded input values $E(\boldsymbol{x}, n)$ as follows:

$$\boldsymbol{T} = [E(\boldsymbol{x}, n), \boldsymbol{H}] \tag{6}$$

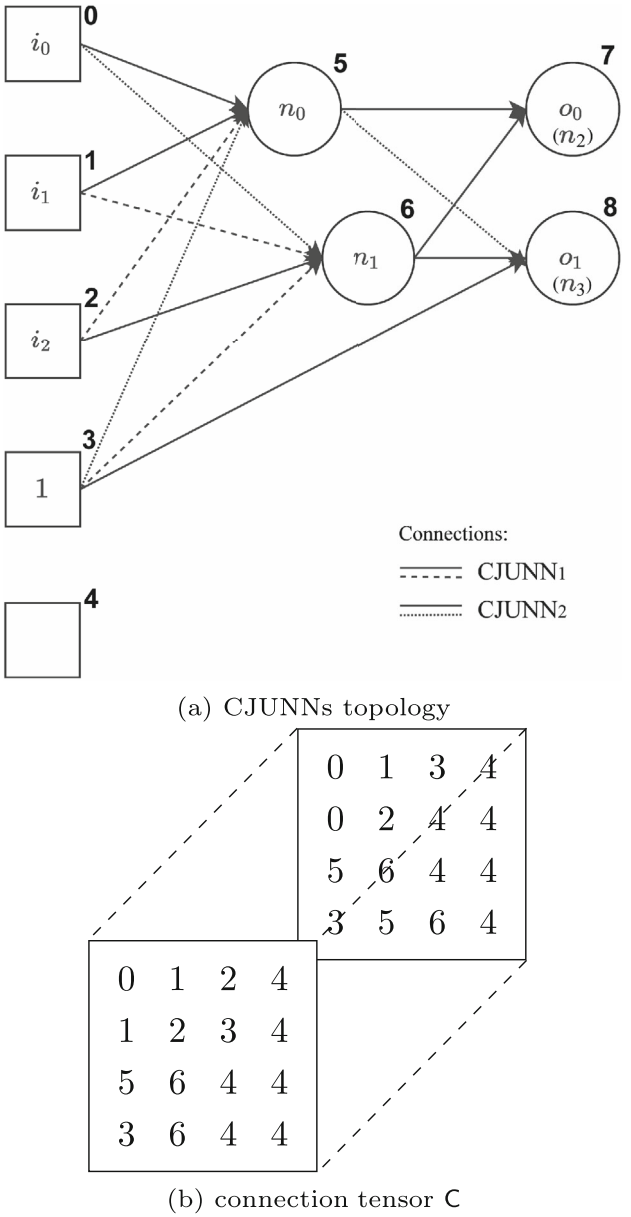

Fig. 1 Example of a CJUNN-based model

Then the indices of the input neurons to the neuron η are selected for all the networks, as $\boldsymbol{II} = \boldsymbol{C}_{:,\eta,:}$. Such input indices $\boldsymbol{II}$ are used to select the corresponding input values from the presynaptic state:

$$\boldsymbol{IV} = G(\boldsymbol{T}, \boldsymbol{II}) \tag{7}$$

Similarly, the connection weights between the selected neurons and the neuron η resulting in $\boldsymbol{IW}_{i,j} = \boldsymbol{W}_{j,\eta}$ $(i, j) \in \boldsymbol{II}$ are selected. Then, the postsynaptic state of the neuron η is selected for all the networks:

$$\boldsymbol{ov}_i = \varphi\left(\sum_{j=1}^{m} \boldsymbol{IV}_{i,j}\boldsymbol{IW}_{i,j}\right) \tag{8}$$

The hidden state matrix $\boldsymbol{H}$ is then updated with the computed values $\boldsymbol{H}_{:,\eta} = \boldsymbol{ov}$. After repeating this process for all the neurons η, all the network's hidden states are linearly combined:

$$\boldsymbol{o}_i = \sum_{j=1}^{n} \boldsymbol{\alpha}_j \boldsymbol{H}_{j,i} \tag{9}$$

and finally, the last o neurons are selected from the output state $\boldsymbol{o}$. The overall forward propagation process is schematized in Algorithm 1.

To investigate the topologies resulting from the proposed approach, we can estimate the deepest and average paths of dense CJUNNs. Specifically, the maximum path length of a network is equal to the number of hidden nodes plus one. Since the forward loop loops from 1 to $h+o$, the deepest path is characterized as follows: (i) start from one input neuron; (ii) connect to the first hidden neuron; (iii) connect the first hidden neuron to the second hidden neuron; (iv) and so on, until the last hidden neuron; (v) finally connect to one output neuron. It is important to note that the loop segment from h to $h+o$ computes the outputs of the output neurons. Since output neurons do not connect between themselves, this segment does not contribute to an increase in the maximum path length. Consequently, the maximum path length of a dense CJUNN $\widehat{PL}_{\max}$ is:

$$\widehat{PL}_{\max} = h + 1 \tag{10}$$

To compute the minimum path, let us consider that the output neurons can be directly connected to the input neurons. As a consequence, the minimum path length is 1.

In a dense fully connected UNN there exist $i \cdot o$ possible paths of length 1, made by the direct connections from one input to one output neuron. Moreover, there exist $i \cdot h \cdot o$ possible paths of length 2, originating from one input neuron, connecting to one hidden neuron, and then to one output neuron. Furthermore, for paths of length 3, the sequence begins with one input neuron, progresses to a hidden neuron, then moves to another hidden neuron with a higher index than the previous one, and finally reaches an output neuron. In general, for a given path length l, the number of potential paths connecting any input neuron to any output neuron is $i \cdot \binom{h}{l-1} \cdot o$. This formula arises from the model's connectivity constraints, which require that each path must progress sequentially through neurons, advancing from those with lower indices to those with higher indices.

Finally we can compute the Average Path Length $\widehat{PL}_{\text{avg}}$ of a dense CJUNN as follows:

$$\widehat{PL}_{\text{avg}} = \frac{\sum_{l=1}^{h+1} i\binom{h}{l-1}ol}{\sum_{l=1}^{h+1} i\binom{h}{l-1}o} = \frac{h}{2} + 1 \tag{11}$$

The justification of Eq. 11 is provided in Appendix A. It is important to notice that Eqs. 11 and 10 assume a dense CJUNN; the training process determines the exact topology with specific maximum and average path lengths, i.e., $PL_{\max}$ and PL_{avg}.

The interested reader is referred to Sect. 4 for the experimental values of both parameters, calculated for non-dense and dense CJUNNs.

The next section formalizes the training model at the algorithmic level. Both the forward model and the training model are executed within the context of a deep learning framework, utilizing an underlying representational model referred to as a computational graph. The computational graph serves as an efficient operational representation of mathematical expressions. It is constructed as a directed graph in which the nodes correspond to distinct mathematical operations [15]. The training model employs the backpropagation algorithm, which exploits the chain rule of differential calculus. This algorithm calculates the error gradients by expressing them as summations of products of local gradients along diverse pathways from a given node to the output.

Algorithm 1: Pseudo-code of the CJUNN forward propagation function

Function FwProp($\boldsymbol{x}$, $\boldsymbol{W}$, C, $\boldsymbol{\alpha}$) **is**
  $\boldsymbol{H} \leftarrow 0$ // $\mathbb{R}^{n\times(h+o)}$
  for η ***in*** $\{1, 2, \cdots, h+o\}$ **do**
    $\boldsymbol{T} \leftarrow [E(\boldsymbol{x}, n), \boldsymbol{H}]$
    // $\mathbb{R}^{n\times(i+h+o)}$
    $\boldsymbol{II} \leftarrow \mathsf{C}_{:,\eta,:}$ // $\mathbb{R}^{n\times m}$
    $\boldsymbol{IV} \leftarrow G(\boldsymbol{T}, \boldsymbol{II})$
    // $\mathbb{R}^{n\times m}$
    $\boldsymbol{IW}_{i,j} \leftarrow \boldsymbol{W}_{j,\eta}\ (i,j) \in \boldsymbol{II}$
    // $\mathbb{R}^{n\times m}$
    $\boldsymbol{ov}_i \leftarrow \varphi(\sum_{j=1}^{m} \boldsymbol{IV}_{i,j}\boldsymbol{IW}_{i,j})$
    // $\mathbb{R}^{n}$
    $\boldsymbol{H}_{:,\eta} \leftarrow \boldsymbol{ov}$ // $\mathbb{R}^{n\times(h+o)}$
  end
  $\boldsymbol{o}_i \leftarrow \sum_{j=1}^{n} \boldsymbol{\alpha}_j \boldsymbol{H}_{j,i}$
  // $\mathbb{R}^{h+o}$
  return $\boldsymbol{o}_{o:h+o}$, $\boldsymbol{H}$
end

3.2 Training model

As anticipated in the introductory section, two mechanisms of the training model favor the exploitation of the best networks and the exploration of the search space: (i) network competition: at each early stop trigger the less important network is pruned; (ii) topological mutation: topological variation of the low-entropy neurons. In this section, the important aspects of the training process are formalized.

To train the CJUNN-based model, the dataset is split into training, validation, and testing. The validation set is not only used to perform early-stopping and hyperparameters optimization but also to carry out the network competition mechanism.

Given an input–output pair from the training dataset ($\boldsymbol{fx}$, $\boldsymbol{fy}$) the backpropagation algorithm is used to compute the gradient of the error function $E(\boldsymbol{y}, \boldsymbol{fy})$ with respect to the parameters which are updated according to the Stochastic Gradient Descent (SGD) optimization algorithm:

$$\nabla_{\boldsymbol{W},\boldsymbol{\alpha}} E(\textsc{FwProp}(\boldsymbol{fx}, \boldsymbol{W}, \mathsf{C}, \boldsymbol{\alpha}), \hat{y}) \tag{12}$$

As previously discussed, the network competition is based on the network importance coefficients $\boldsymbol{\alpha}$, to which the softmax normalization function is applied:

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{13}$$

This avoids $\boldsymbol{\alpha}$ elements converging to $\frac{1}{n}$ [8]. The removal of the least important CJUNN, associated to the lowest $\boldsymbol{\alpha}$, is carried out when the error on the validation set stops decreasing. The overall training process ends when only one CJUNN remains. The overall training model is schematized in Algorithm 2.

A significant issue of the forward propagation is that the gradients over the second argument of the *gather* operator G are not defined, since they are indexes. As a consequence, the connection tensor C cannot be trained using backpropagation. For this reason, to ensure topological variability, the neuron entropy is used to apply a topological mutation. Specifically, the entropy of the neuron η is defined as follows:

$$\boldsymbol{h}_\eta = -\log(\boldsymbol{p}_\eta)\boldsymbol{p}_\eta - \log(1-\boldsymbol{p}_\eta)(1-\boldsymbol{p}_\eta) \tag{14}$$

where $\boldsymbol{p}_\eta$ is the probability of η to fire. The definition of neuron firing depends on the neuron activation function. For most, the activation function is $y > \tau$ for some threshold τ (for example, $\tau = 0$ for ReLU or sigmoid). In the proposed approach, $\boldsymbol{p}_\eta$ is approximated for all the neurons using the Exponential Moving Average (EMA). The activation neuron entropy is an effective way to measure each neuron's information contribution to the network. The rationale behind the approach is that low entropy neurons carry little to no information in the network; as a such, the network can easily adapt to the topological mutations affecting low entropy neurons because little to no information is removed from the network. The topological mutation happens only when the activation entropy of a neuron falls below a certain threshold, called target entropy T_h. The target entropy is a hyperparameter that is tuned in the hyperparameter optimization. Then, if a neuron entropy $\boldsymbol{h}_\eta$ is lower than the target entropy T_h, its incoming hidden neuron with the lowest entropy is selected and replaced with the *zero-bias* neuron. Similarly, if the neuron has an incoming *zero-bias* neuron, it is replaced with the hidden neuron with the highest entropy. This mutation operation is carried out every training step, after an initial warm-up period.

Algorithm 2: Pseudo-code of the CJUNN training function

```
Function TopologicalMut(C, p) is
    h ← −log(p)p − log(1 − p)(1 − p)
    for η in {1, 2, ··· , h + o} do
        if h_η < T_h then
            C ← RemoveLowEnt(C, h)
            C ← ReplaceHighEnt(C, h)
        end
    end
end

Function RmNetwork(C, α) is
    α_min ← arg min(α)
    C ← PruneNetwork(C, α_min)
    α ← RemoveElement(α, α_min)
end

Function Train() is
    W ← RandomWeights()
    C ← RandomConnections()
    α ← RandomVector()
    p ← 0
    for epoch in {1, 2, ··· , epochs} do
        for (fx, fy) in trainingSet do
            if epoch > warmup then
                C ← TopologicalMut(C, p)
            end
            if EarlyStop(validationSet) then
                if NetNumber(C) == 1 then
                    return W, C
                else
                    C ← RmNetwork(C, α)
                end
            end
            y, H ← FwProp(fx, W, C, σ(α))
            p ← EMA(Act(H))
            ∇ ← ∇_{W,α} E(y, fy)
            W, α ← SGD(∇, W, α)
        end
    end
end
```

4 Experimental studies

The proposed architectural model has been implemented, tested, and publicly released on the GitHub platform [16], to foster its application in various research environments. For simplicity and readability, the pseudo-code shown in the previous section omits the batch dimension. In the implementation and for all experiments a *batch size* of 16 has been set.

The proposed approach is hybrid in the sense that, from one side the CJUNNs are natively sparse networks, and from the other side, the training model prunes entire CJUNNs instead of single connections. However, as discussed in the introductory section, the approach is methodologically very different from natively sparse approaches existing in the literature. Given the generality, maturity, and availability of dense NN pruning approaches, they can be efficiently and fairly compared with the proposed CJUNN-based model. Given the above reasons, in this experimental section, the CJUNN model is experimented with and evaluated with respect to MLPs as a reference dense architecture, by adopting different pruning levels and methods, as well as different numbers of hidden layers. In all cases, the pruning is done at the end of the training. Specifically, the considered approaches are summarized in the following Table 1.

To develop the model, the dataset is split into training (60%), validation (20%), and testing (20%), using the holdout method. To fairly compare the models, the hyperparameters optimization for the MLP and CJUNN-based models has been carried out using as the objective function the accuracy of the validation set. To sample the hyperparameters to use for each run, the Tree-structured Parzen Estimator (TPE) has been used [17]. To prune unpromising runs, the Successive Halve Pruning (SHP) method has been used [18]. For each approach, the testing accuracy and the number of parameters have been considered as performance metrics.

Table 1 MLP pruning based approaches

Acronym	Hidden layers	Pruning type
$MLP^{2}_{100\%}$	2	Non-pruned
$MLP^{2}_{pr50\%}$	2	50% Magnitude pruned
$MLP^{2}_{pr90\%}$	2	90% Magnitude pruned
$MLP^{2}_{lt50\%}$	2	50% Lottery ticket pruned
$MLP^{2}_{lt90\%}$	2	90% Lottery ticket pruned
$MLP^{3}_{100\%}$	3	Non-pruned
$MLP^{3}_{pr50\%}$	3	50% Magnitude pruned
$MLP^{3}_{pr90\%}$	3	90% Magnitude pruned
$MLP^{3}_{lt50\%}$	3	50% Lottery ticket pruned
$MLP^{3}_{lt90\%}$	3	90% Lottery ticket pruned
$MLP^{4}_{100\%}$	4	Non-pruned
$MLP^{4}_{pr50\%}$	4	50% Magnitude pruned
$MLP^{4}_{pr90\%}$	4	90% Magnitude pruned
$MLP^{4}_{lt50\%}$	4	50% Lottery ticket pruned
$MLP^{4}_{lt90\%}$	4	90% Lottery ticket pruned

Table 2 Models hyperparameters

Hyperparameter	CJUNNs	$(MLP^{*}_{**\%})$
l (neurons in hid. layers)		✓
ρ (learning rate)	✓	✓
h (hidden nodes)	✓	
m (max inc. conn.)	✓	
T_h (target entropy)	✓	

The learning rate ρ is a common hyperparameter for all the considered models. The MLP architecture has one additional hyperparameter: the number of neurons of the hidden layers l. In contrast, CJUNNs have three additional hyperparameters: (i) the maximum number of incoming connections m, (ii) the mutation entropy threshold T_h, and (iii) the number of hidden nodes h. Table 2 shows the hyperparameters for each model.

To assess the performance on a wide variety of tasks, the models have experimented with 8 different classification datasets with very different scopes, number of features, classes, and instances:

- The *Iris Dataset* [19], which consists of 150 instances of sepal, petal length, and width, of three iris plants classes (Setosa, Versicolour, and Virginica);
- the *Seeds Dataset* [20], which is composed of 210 instances of 3 different varieties of wheat (Kama, Rosa, and Canadian);
- the *Transfusion Dataset* [21], which is a binary classification problem made by 748 instances of information regarding previous blood donations, and two classes representing if the donor donated again or not;
- the *QSAR Dataset* [22], which contains values for 41 molecular descriptors used to classify 1055 chemicals into 2 classes (ready and not ready biodegradable);
- the *Plates Dataset* [23], consisting of steel plates faults, classified into 7 different classes with 27 numerical features and 7 binary;
- the *KR-KP Dataset* [24] composed by 3196 chess white King+Rook versus black King+Pawn on a7 with white to play endings described by 36 attributes representing the board and divided into two classes (white can win or not);
- the *Robot Dataset* [25] composed by 5456 readings of 24 ultrasound sensors arranged circularly around a robot waist to be classified in 4 actions (Move-Forward, Slight-Right-Turn, Sharp-Right-Turn, Slight-Left-Turn);

Table 3 Dataset dimensions

Dataset	Features	Classes	Instances
Iris	5	3	150
Seeds	8	3	210
Transfusion	5	2	748
QSAR	42	2	1055
Plates	34	2	1941
KR-KP	37	2	3196
Robot	24	4	5456
Nursery	9	5	12,960

- the *Nursery Dataset* [26] derived from a hierarchical decision model originally developed to rank applications for nursery schools, and composed of 12960 instances of 9 features describing the family state and divided into 5 classes (not recommend admission, mild recommend admission, recommend admission, recommend admission with priority and recommend admission with special priority).

Table 3 shows the dimensions of each dataset.

To evaluate the effectiveness of the proposed approach in terms of performance metrics, Tables 4 and 5 show the accuracy and the parametric complexity of CJUNNs with respect to two hidden-layers MLPs; Tables 6 and 7 show the accuracy and parametric complexity of CJUNNs with respect to three hidden-layers MLPs; Tables 8 and 9 show the accuracy and parametric complexity of CJUNNs with respect to four hidden-layers MLPs. All performance metrics are calculated as the 95% confidence interval over 10 runs. Here, the accuracy of the proposed CJUNNs is highlighted in boldface style.

It is clear from the tables that, under the same parametric complexity, the MLP pruned via the lottery ticket sensibly overcomes the MLP pruned via the weight magnitude. In particular, the former with 50% pruning is equivalent to the dense MLP, whereas the latter sensibly loses accuracy.

In Table 4 it can be noticed that, in all datasets, the accuracy of the dense MLP and of the MLP with 50% pruning via lottery ticket are equivalent. Such accuracy is also equivalent to the accuracy achieved by CJUNNs on the first 5 datasets. The CJUNN-based approach loses about two percentage points on the last 3 datasets. However, in Table 5 it is apparent that the parametric complexity of the CJUNNs is dramatically lower, thus confirming the effectiveness of the proposed approach. It can also be noticed in Tables 8 and 9 that four hidden-layer MLPs drastically outperform two hidden-layer ones. Most notably $MLP^4_{lr90\%}$ achieves similar performance with respect to its unpruned counterpart $MLP^4_{100\%}$. CJUNN models also achieve similar performance to $MLP^4_{100\%}$ and $MLP^4_{lr90\%}$ while having almost an order of magnitude fewer parameters. Similar behavior can be observed in Tables 6 and 7 where three hidden-layer MLPs have similar performances with respect to their four hidden-layer counterparts with slightly less parametric complexity. The CJUNN models are also competitive with respect to the three hidden-layer MLPs.

Overall, the experimental results, as shown in Tables 4, 5, 6, 7, 8 and 9, demonstrate the effectiveness of CJUNNs in terms of both accuracy and parametric simplicity compared to other models. CJUNNs consistently demonstrate robust performance in terms of accuracy, often outperforming other models. Moreover, CJUNNs exhibit significantly lower parametric complexity compared to the other models. These results support the claim that CJUNNs provide an efficient and effective solution to the challenge of reducing the number of parameters in NN models. The competitive and joint nature of CJUNNs, where multiple networks compete and share connections, allows for iterative pruning of less important networks while preserving the most important network. This pruning process results in a sparse UNN model with significantly fewer parameters.

To investigate the resulting topologies, Table 10 shows, for each dataset, average and maximum path lengths of pruned (PL_*) and dense ($\widehat{PL}_*$) CJUNNs. The parameters of the dense CJUNNs are computed according to Formulas 10 and 11. Some important considerations can be drawn. For small datasets (i.e. Iris, Seeds, and Transf.), the pruned CJUNN is characterized by a larger average path and by a similar maximum path, with respect to the dense CJUNN. On the other side, for large datasets, the pruned CJUNN is characterized by a smaller average path and smaller maximum path, with respect to the dense CJUNN. This shows that for increasing complexity, the generated network is able to achieve a better exploitation of the hidden nodes, resulting in more internal hops.

5 Conclusions

This paper aims to formally present a novel perspective on optimizing sparse neural network topologies, referred to as Competitive Joint Unstructured Neural Networks (CJUNNs). In contrast to dense NN pruning methods, the proposed approach combines multiple natively sparse NNs that are joint because they share connection weights. In contrast to native sparse methods, the proposed technique does not necessitate specific assumptions and constraints. It is a hybrid method because the pruning is performed at the level of entire sparse networks, and it is guided by an importance metric that actively participates in the gradient optimization, leading to competition between NNs. A mutation mechanism at the connection level is also included to preserve space exploration.

Table 4 Accuracy of two hidden-layers models for each dataset

Dataset	$MLP^2_{100\%}$	$MLP^2_{pr50\%}$	$MLP^2_{pr90\%}$	$MLP^2_{lt50\%}$	$MLP^2_{lt90\%}$	**CJUNNs**
Iris	96.3 ± 0.8	89.0 ± 9.1	38.7 ± 13.8	96.0 ± 1.0	65.7 ± 15.9	$\mathbf{97.0 \pm 0.8}$
Seeds	89.1 ± 1.2	87.9 ± 3.4	33.3 ± 8.1	90.2 ± 1.7	65.5 ± 14.6	$\mathbf{89.9 \pm 2.6}$
Transf	78.1 ± 0.8	76.9 ± 1.0	74.3 ± 2.1	77.7 ± 1.0	75.5 ± 0.3	$\mathbf{77.9 \pm 0.8}$
QSAR	87.3 ± 0.9	86.8 ± 0.9	75.2 ± 3.2	88.0 ± 0.8	85.3 ± 1.4	$\mathbf{86.3 \pm 1.6}$
Plates	100 ± 0.0	100 ± 0.0	90.4 ± 6.6	100 ± 0.0	99.7 ± 0.8	$\mathbf{100 \pm 0.0}$
KR-KP	98.6 ± 0.2	98.2 ± 0.4	78.5 ± 12.2	98.6 ± 0.3	95.7 ± 0.2	$\mathbf{96.3 \pm 0.4}$
Robot	86.9 ± 0.6	71.7 ± 4.2	46.7 ± 7.0	87.4 ± 0.7	72.9 ± 7.3	$\mathbf{85.1 \pm 2.2}$
Nursery	97.5 ± 0.0	89.7 ± 3.1	32.5 ± 0.0	97.4 ± 0.0	33.1 ± 0.7	$\mathbf{95.5 \pm 0.3}$

Table 5 Parametric complexity of two hidden-layers models for each dataset

Dataset	$MLP^2_{100\%}$	$MLP^2_{pr50\%}$	$MLP^2_{pr90\%}$	$MLP^2_{lt50\%}$	$MLP^2_{lt90\%}$	**CJUNNs**
Iris	535	270	56	270	56	$\mathbf{20 \pm 0}$
Seeds	928	467	96	467	96	$\mathbf{25 \pm 1}$
Transf	347	175	37	175	37	$\mathbf{38 \pm 2}$
QSAR	1946	975	197	975	197	$\mathbf{148 \pm 2}$
Plates	2416	1208	245	1208	245	$\mathbf{105 \pm 2}$
KR-KP	1538	769	157	769	157	$\mathbf{180 \pm 3}$
Robot	1460	730	149	730	149	$\mathbf{185 \pm 2}$
Nursery	3204	1602	321	1602	321	$\mathbf{128 \pm 2}$

Table 6 Accuracy of three hidden-layers models for each dataset

Dataset	$MLP^3_{100\%}$	$MLP^3_{pr50\%}$	$MLP^3_{pr90\%}$	$MLP^3_{lt50\%}$	$MLP^3_{lt90\%}$	**CJUNNs**
Iris	96.7 ± 0.0	98.0 ± 1.9	61.3 ± 9.2	98.0 ± 1.0	98.0 ± 1.0	$\mathbf{97.0 \pm 0.8}$
Seeds	92.9 ± 1.6	92.6 ± 1.0	62.6 ± 6.1	94.0 ± 1.0	95.2 ± 1.1	$\mathbf{89.9 \pm 2.6}$
Transf	78.7 ± 0.8	78.1 ± 0.6	75.7 ± 0.7	78.3 ± 0.5	78.7 ± 0.8	$\mathbf{77.9 \pm 0.8}$
QSAR	88.4 ± 0.6	87.9 ± 0.9	80.2 ± 2.7	88.6 ± 0.6	88.2 ± 0.6	$\mathbf{86.3 \pm 1.6}$
Plates	100.0 ± 0.0	100.0 ± 0.0	90.3 ± 11.4	100.0 ± 0.0	100.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$
KR-KP	98.0 ± 0.3	98.2 ± 0.4	92.6 ± 1.4	98.7 ± 0.1	96.3 ± 0.3	$\mathbf{96.3 \pm 0.4}$
Robot	87.5 ± 0.4	82.8 ± 1.3	58.6 ± 3.9	87.9 ± 0.8	88.3 ± 1.0	$\mathbf{85.1 \pm 2.2}$
Nursery	97.4 ± 0.1	96.7 ± 0.7	66.3 ± 13.2	97.5 ± 0.0	87.3 ± 2.7	$\mathbf{95.5 \pm 0.3}$

Table 7 Parametric complexity of three hidden-layers models for each dataset

Dataset	$MLP^3_{100\%}$	$MLP^3_{pr50\%}$	$MLP^3_{pr90\%}$	$MLP^3_{lt50\%}$	$MLP^3_{lt90\%}$	**CJUNNs**
Iris	3775	2158	533	1405	417	$\mathbf{20 \pm 0}$
Seeds	4260	2224	325	583	449	$\mathbf{25 \pm 1}$
Transf	2921	741	93	64	431	$\mathbf{38 \pm 2}$
QSAR	4062	3651	685	2949	467	$\mathbf{148 \pm 2}$
Plates	4334	463	580	256	400	$\mathbf{105 \pm 2}$
KR-KP	1388	2035	685	3526	531	$\mathbf{180 \pm 3}$
Robot	4255	2222	357	2035	341	$\mathbf{185 \pm 2}$
Nursery	3136	2773	178	2877	272	$\mathbf{128 \pm 2}$

Table 8 Accuracy of four hidden-layers models for each dataset

Dataset	$MLP^4_{100\%}$	$MLP^4_{pr50\%}$	$MLP^4_{pr90\%}$	$MLP^4_{lt50\%}$	$MLP^4_{lt90\%}$	**CJUNNs**
Iris	97.7 ± 0.9	96.0 ± 2.6	60.0 ± 10.2	96.7 ± 0.0	96.7 ± 1.6	$\mathbf{97.0 \pm 0.8}$
Seeds	92.4 ± 1.1	91.2 ± 2.2	56.2 ± 11.7	93.3 ± 1.1	94.8 ± 1.3	$\mathbf{89.9 \pm 2.6}$
Transf	78.3 ± 0.4	77.9 ± 0.6	75.9 ± 0.7	78.3 ± 0.6	78.2 ± 0.5	$\mathbf{77.9 \pm 0.8}$
QSAR	88.1 ± 0.6	87.3 ± 1.1	75.6 ± 5.3	88.5 ± 0.7	88.3 ± 0.7	$\mathbf{86.3 \pm 1.6}$
Plates	100.0 ± 0.0	100.0 ± 0.0	93.1 ± 1.8	100.0 ± 0.0	100.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$
KR-KP	98.0 ± 0.2	98.1 ± 0.3	93.9 ± 0.9	98.4 ± 0.3	96.5 ± 0.3	$\mathbf{96.3 \pm 0.4}$
Robot	87.0 ± 0.6	82.5 ± 1.3	50.2 ± 6.6	87.7 ± 0.7	86.3 ± 1.9	$\mathbf{85.1 \pm 2.2}$
Nursery	97.4 ± 0.1	95.9 ± 0.4	60.9 ± 11.4	97.4 ± 0.1	81.0 ± 4.9	$\mathbf{95.5 \pm 0.3}$

Table 9 Parametric complexity of four hidden-layers models for each dataset

Dataset	$MLP^4_{100\%}$	$MLP^4_{pr50\%}$	$MLP^4_{pr90\%}$	$MLP^4_{lt50\%}$	$MLP^4_{lt90\%}$	**CJUNNs**
Iris	3845	2879	582	2377	331	$\mathbf{20 \pm 0}$
Seeds	888	3359	537	1289	732	$\mathbf{25 \pm 1}$
Transf	639	1501	521	201	603	$\mathbf{38 \pm 2}$
QSAR	5582	2791	563	3939	986	$\mathbf{148 \pm 2}$
Plates	8144	1625	131	1625	187	$\mathbf{105 \pm 2}$
KR-KP	5930	3681	961	2701	740	$\mathbf{180 \pm 3}$
Robot	9104	2911	255	4552	787	$\mathbf{185 \pm 2}$
Nursery	3799	3999	499	2984	603	$\mathbf{128 \pm 2}$

Table 10 For each dataset, average and maximum path lengths of pruned (PL_*) and dense ($\widehat{PL}_*$) CJUNNs

Dataset	PL_{avg}	$\widehat{PL}_{\text{avg}}$	$PL_{\max}$	$\widehat{PL}_{\max}$
Iris	2.36 ± 0.06	1.67	3.00 ± 0.00	3
Seeds	2.35 ± 0.13	1.67	3.00 ± 0.00	3
Transf	2.97 ± 0.27	2.29	4.00 ± 0.00	4
QSAR	3.57 ± 0.42	4.47	5.00 ± 1.29	8
Plates	3.53 ± 0.73	6.00	5.10 ± 1.43	11
KR-KP	3.83 ± 0.34	4.47	5.40 ± 1.00	8
Robot	3.78 ± 0.10	3.95	5.00 ± 0.00	7
Nursery	4.14 ± 0.38	4.47	6.20 ± 0.82	8

The paper formalizes the forward propagation model and the training model, which have been also implemented on a deep learning framework and publicly released. Experimental results on benchmark data show that the proposed approach can sensibly outperform other dense NN pruning methods in terms of parametric complexity, by keeping almost the same accuracy.

Even if the proposed model has extremely competitive performance, it has some points that need to be addressed. It uses parallel sparse networks with shared weights, which can lead to a large number of connections that need to be stored and computed during the training process. This can limit the model's efficiency and scalability, especially for larger networks. Moreover, the unstructured topology of the CJUNN model makes it more complex to understand and analyze than traditional structured neural networks. This complexity can make it harder to interpret the behavior of the model and diagnose problems. Finally, the CJUNN model has several hyperparameters, and finding the optimal values for these hyperparameters can be challenging and time-consuming, and the performance of the model can be sensitive to their values. Another limitation is related to the training and optimization of the CJUNN model. While natively sparse models based on meta-heuristics can preserve topological properties, they cannot be assured by the proposed model. The CJUNN model attempts to combine the advantages of pruning methods and natively sparse models, but it introduces new complexities. The competitive nature of the CJUNN model, where multiple sparse networks compete and are pruned iteratively, adds a layer of optimization complexity.

This complexity may lead to challenges in finding the optimal balance between network competition and exploration of the search space, potentially hindering the convergence and overall performance of the model.

Future research can aim to incorporate more advanced architectures to enhance the performance of the CJUNN model, which is already quite complex. For instance, attention mechanisms or reinforcement learning algorithms could be implemented to achieve this goal. To assess the model's generalizability, future research could evaluate its performance on more challenging datasets. Although the CJUNN model has performed well on several benchmark datasets, these datasets are not as complex as real-world scenarios. Another limitation of the CJUNN model is its lack of interpretability, as it operates as a black-box model. Future research could explore methods to enhance the model's interpretability, such as by visualizing its internal representations. The scalability of the CJUNN model is another area that future research could focus on. Currently, the model is computationally expensive, making it difficult to scale up to larger datasets and more complex problems. To address this issue, we could investigate more efficient algorithms and techniques to reduce the computational requirements of the CJUNN model. Another important direction for future research is to explore the structural characteristics and properties of the pruned topologies obtained through the CJUNN model. Analyzing the resulting sparse network architectures can help uncover patterns, regularities, and emergent properties that may contribute to our understanding of neural network structure and function. Investigating the sparsity patterns, connectivity distribution, and other topological properties can shed light on the information flow, information processing capabilities, and computational efficiency of the pruned networks. Furthermore, future investigations could explore the transferability of the pruned topologies obtained through the CJUNN model. Evaluating the performance of the pruned networks across various tasks, datasets, and domains can provide insights into their robustness and versatility. This analysis can help assess the practical applicability of the CJUNN model and its potential for deployment in real-world scenarios like fine-tuning pruning topologies on specific tasks.

A Appendix

A.1 Justification for Eq. 11

Given i input nodes, h hidden nodes, and o output nodes, there are $i \cdot \binom{h}{l-1} \cdot o$ paths from any input node to any output node of length l. The minimum path length is $PL_{\min} = 1$, and the maximum is $PL_{\max} = h+1$. To compute the average path length, let us determine the weighted sum from 1 to $h+1$ of the path lengths, where each path length l is multiplied by the number of paths of that length. Finally, let us divide this sum by the total number of paths:

$$\widehat{PL}_{\text{avg}} = \frac{\sum_{l=1}^{h+1} i\binom{h}{l-1} ol}{\sum_{l=1}^{h+1} i\binom{h}{l-1} o}$$

According to the binomial theorem:

$$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$$

For $x = 1$ and $a = 1$ it follows:

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

Then, the denominator of Eq. 11 can be simplified as follows:

$$\sum_{l=1}^{h+1} i\binom{h}{l-1} o = io \sum_{l=0}^{h} \binom{h}{l} = io2^h$$

As the numerator of Eq. 11, it can be rewritten as follows:

$$\begin{aligned}\sum_{l=1}^{h+1} i\binom{h}{l-1} ol &= io \sum_{l=0}^{h} \binom{h}{l} (l+1) \\ &= io\left(\sum_{l=0}^{h} \binom{h}{l} l + \sum_{l=0}^{h} \binom{h}{l}\right) \\ &= io\left(\sum_{l=0}^{h} \binom{h}{l} l + 2^h\right)\end{aligned}$$

The first addend $\sum_{l=0}^{h} \binom{h}{l} l$ can be simplified by considering the binomial theorem for $a = 1$:

$$(x+1)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

Taking the derivatives of both sides:

$$\frac{\partial}{\partial x}(x+1)^n = \frac{\partial}{\partial x} \sum_{k=0}^{n} \binom{n}{k} x^k$$

$$n(x+1)^{n-1} = \sum_{k=0}^{n} k\binom{n}{k} x^{k-1}$$

For $x = 1$ it follows that:

$$\sum_{k=0}^{n} k\binom{n}{k} = n2^{n-1}$$

i.e.,

$$\sum_{l=0}^{h} \binom{h}{l} l = h2^{h-1}$$

As a consequence, Eq. 11 can be rewritten as follows:

$$\widehat{PL}_{\text{avg}} = \frac{\sum_{l=1}^{h+1} i\binom{h}{l-1} ol}{\sum_{l=1}^{h+1} i\binom{h}{l-1} o} = \frac{io(h2^{h-1} + 2^h)}{io2^h}$$
$$= \frac{2^{h-1}(h+2)}{2^h} = \frac{h+2}{2} = \frac{h}{2} + 1$$

Author Contributions Federico A. Galatolo: public responsibility for the content, concept, design, analysis, writing, and revision of the manuscript. Mario G.C.A. Cimino: public responsibility for the content, concept, design, analysis, writing, and revision of the manuscript.

Funding Work partially supported: (i) by the Italian Ministry of Education and Research (MIUR) in the frameworks of the FoReLab project (Departments of Excellence), of the National Recovery and Resilience Plan (National Center for Sustainable Mobility MOST/Spoke10), and of the "Reasoning" project, PRIN 2020 LS Programme, Project number 2493 04-11-2021; (ii) by the PNRR - M4C2 - Investimento 1.3, Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 1 "Human-centered AI", funded by the European Commission under the NextGeneration EU program; (iii) by the University of Pisa, in the framework of the PRA_2022_101 project "Decision Support Systems for territorial networks for managing ecosystem services".

Availability of data and material The data used in this work is publicly available on: https://github.com/galatolofederico/cjunn.

Code availability The software used in this work is publicly available at: https://github.com/galatolofederico/cjunn.

Declarations

Conflict of interest All the authors declare that they have no known conflict of interest or conflict of interest that could have appeared to influence the work reported in this paper.

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

References

1. Blalock, D., Gonzalez Ortiz, J.J., Frankle, J., Guttag, J.: What is the state of neural network pruning? arXiv preprint arXiv:2003.03033, (2020)
2. Alford, S., Robinett, R., Milechin, L., Kepner, J.: Pruned and structurally sparse neural networks. In: 2018 IEEE MIT Undergraduate Research Technology Conference (URTC), pp. 1–4. IEEE (2018)
3. Galatolo, F.A., Cimino, M.G., Vaglini, G.: Formal derivation of mesh neural networks with their forward-only gradient propagation. Neural Process. Lett. **53**, 1963–1978 (2021)
4. Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 (2016)
5. Munro, P.: Backpropagation, pp. 73–73. Springer US, Boston, MA (2010)
6. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. arXiv preprint arXiv:1810.05270 (2018)
7. Frankle, J., Carbin, M.: The lottery ticket hypothesis: finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635 (2018)
8. Srinivas, S., Subramanya, A., Venkatesh Babu, R.: Training sparse neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 138–145 (2017)
9. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11264–11272 (2019)
10. Tanaka, H., Kunin, D., Yamins, D.L., Ganguli, S.: Pruning neural networks without any data by iteratively conserving synaptic flow. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems, vol. 33, pp. 6377–6389. Curran Associates Inc., New York (2020)
11. Prabhu, A., Varma, G., Namboodiri, A.: Deep expander networks: efficient deep networks from graph theory. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 20–35 (2018)
12. Kepner, J., Robinett, R.: Radix-net: structured sparse matrices for deep neural networks. In: 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 268–274. IEEE (2019)
13. Liu, S., Tian, Y., Chen, T., Shen L.: Don't be so dense: sparse-to-sparse GAN training without sacrificing performance. arXiv preprint arXiv:2203.02770 (2022)
14. Galatolo, F., Cimino, M.G.C.A., Vaglini, G.: Using stigmergy as a computational memory in the design of recurrent neural networks. In: Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods - ICPRAM, pp. 830–836. INSTICC, SciTePress (2019)
15. Galatolo, F.A., Mario, G., Cimino, C.A., Vaglini, G.: Using stigmergy to incorporate the time into artificial neural networks. In: Groza, A., Prasath, R. (eds.) Mining Intelligence and Knowledge Exploration, pp. 248–258. Springer, Cham (2018)
16. Galatolo, F.A.: cjunn, https://github.com/galatolofederico/cjunn (2021)
17. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. Adv. Neural Inf. Process. Syst. **24** (2011)
18. Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., Talwalkar, A.: A system for massively parallel hyperparameter tuning. arXiv preprint arXiv:1810.05934 (2018)
19. Fisher, Ronald A.: The use of multiple measurements in taxonomic problems. Ann. Eugen. **7**(2), 179–188 (1936)
20. Charytanowicz, M., Niewczas, J., Kulczycki, P., Kowalski, P.A., Łukasik, S., Żak S.: complete gradient clustering algorithm for features analysis of x-ray images. In: Information Technologies in Biomedicine, pp. 15–24. Springer (2010)
21. Yeh, I.-C., Yang, K.-J., Ting, T.-M.: Knowledge discovery on RFM model using Bernoulli sequence. Expert Syst. Appl. **36**(3), 5866–5871 (2009)

22. Mansouri, Kamel, Ringsted, Tine, Ballabio, Davide, Todeschini, Roberto, Consonni, Viviana: Quantitative structure-activity relationship models for ready biodegradability of chemicals. J. Chem. Inf. Model. **53**(4), 867–878 (2013)
23. Buscema, M., Terzi, S., Tastle, W.J.: A new meta-classifier. In: Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS), Toronto, ON, Canada, pp. (1–7) (2010)
24. Shapiro, A.D.: The role of structured induction in expert systems, unpublished Ph.D thesis, University of Edinburgh (1983)
25. Freire, A.L., Barreto, G.A., Veloso, M., Varela, A.T.: Short-term memory mechanisms in neural network learning of robot navigation tasks: a case study. In: 2009 6th Latin American Robotics Symposium (LARS 2009), pp. 1–6. IEEE (2009)
26. Zupan, B., Bohanec, M., Bratko, I., Demsar, J.: Machine learning by function decomposition. In: ICML, pp. 421–429 (1997)