Optimizing sparse topologies via Competitive Joint Unstructured Neural Networks

Federico A. Galatolo and Mario G.C.A. Cimino
Department of Information Engineering, University of Pisa, Italy

***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 in removing connections of a dense model, and natively sparse models, based on training sparse models using meta-heuristics to guarantee their topological properties. In this paper, the limits of both approaches are discussed. A novel hybrid training approach is developed and experimented, 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 together with an efficient derivation in tensor algebra, which has been implemented and publicly released. Experimental results show its effectiveness on benchmark datasets and in comparison with structured pruning.**

***Index Terms*—Artificial neural networks, Neural network pruning, Unstructured topology**

I. Introduction and background

In the last years, Artificial Neural Networks (NNs) have been object of great improvements and major milestones. However, the significant rise 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 in pruning parameters of a dense model, which can be done iteratively, i.e., during the training, or at the end of the training. The second approach consists in training sparse models using meta-heuristics to guarantee topological properties [2]. Hybrid categories have been also proposed [3]. In this paper, a novel hybrid approach is developed and experimented, 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 overcome 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 illustrated and discussed in Section 3. Finally, conclusions and future work are drawn in Section 4.

A. 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 weights 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 $\boldsymbol{y} = \varphi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$. 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 subset of data [4]. In supervised training settings, a set of desired input-output pairs $\hat{\mathbb{X}}$, $\hat{\mathbb{Y}}$ and an error function $\epsilon(\boldsymbol{y}, \hat{\boldsymbol{y}})$ are given. By denoting with p the parameters of a NNs, and with $f(x)$ its input-output function, then in SGD the mean gradient of the error function with respect to the NNs parameters is computed as $\Delta_p E_{(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in (\hat{\mathbb{X}}, \hat{\mathbb{Y}})}[\epsilon(f(\hat{\boldsymbol{x}}), \hat{\boldsymbol{y}})]$. In its common form, the parameters are updated to follow the minimum error defined by the gradient. To compute the minimum-error gradient, the *backpropagation* algorithm is used [5].

B. 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 in 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 in using *gate variables* representing the probability of a certain weights to be pruned [8]. The *gate variables* approach consists in 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 as a random sample of the Bernoulli random variable with $p = \boldsymbol{G}_{i,j}$. Finally the masked weight matrix $\boldsymbol{W}_s = \boldsymbol{W} \cdot \boldsymbol{G_S}$ is used instead of the dense weight matrix $\boldsymbol{W}$. In order to promote the sparsity and to avoid the gate values to converge to 0.5 the authors also suggested to use l_1 and l_2 regularization. This approach can reach approximately 10x - 20x compression rate 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 models was proposed by NVIDIA [9]. The idea of their approach is to approximate the importance of each parameter and prune the less important ones. Given a dataset $\mathbb{D}$ and an error function E, the authors define the importance of a parameter w_m as the squared difference of the error of the network with and without that parameter: $I_m = (E(\mathbb{D}, \boldsymbol{W}) - E(\mathbb{D}, \boldsymbol{W} | w_m = 0))^2$. A direct computation of I_m is infeasible, because it requires to evaluate the network a number of times equal to the number of elements of $\boldsymbol{W}$. For this reason, in the paper the authors suggest to use the first order Taylor expansion of I_m: ${I_m}^{(1)} = (\frac{\partial E(\mathbb{D}, \boldsymbol{W})}{\partial w_m} w_m)^2$. This 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.

C. Natively sparse models

Another approach to reduce the number of parameters of NNs is to use natively sparse models. The main issue of using sparse models is how to ensure both high sparsity and connectivity. One of the most recent architecture facing this problem is based on the so-called *X-Nets* [10]. 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 outperforms the classical counterpart by up to 4% in accuracy, while having more than 10x parameters reduction.

In another research work, it has been showed 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* [11]. *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. As a consequence, it is possible to exploit memory and runtime efficiently in 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 [11]. 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 as to 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 [11]. 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. With respect to this problem, the proposed approach is based on a combination of multiple sparse networks, which are progressively pruned by their contributions on the output. This self-organized competition is at the basis of the proposed method.

D. Hybrid models

In an effort 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 $\boldsymbol{s_t} = \varphi(\boldsymbol{s_{t-1}} \boldsymbol{A})$. 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 efficiently exploiting 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.

II. Development of the proposed model

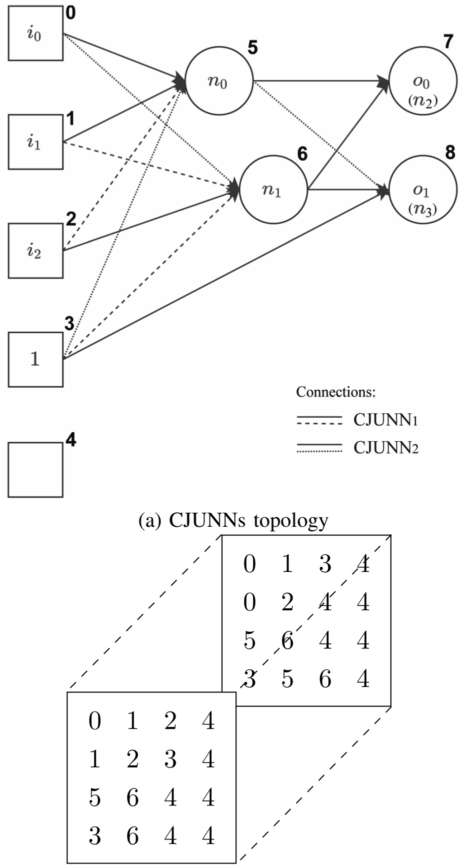
The proposed Competitive Joint Unstructured Neural networks (CJUNNs) is an unstructured model, in which neurons are allowed to be connected in any topology. Differently from MNNs, instead of relying on an implicit topology defined over an Adjacency Matrix and on a state transition function, in CJUNNs the topology is explicitly defined in a connections matrix. Differently from natively sparse models, in which an explicit criterion is established at design time for having 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 guarantee both exploitation of the best networks and exploration of the search space: (i) networks competition: at each early stop, the less important network is pruned; (ii) topological mutation: topological variation of the low-entropy neurons. Differently from [9] it is defined the importance of a whole network 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 [10] and [11] the network topology is not based on an auxiliary graph model, but local topological mutations are stochastically triggered on the low-entropy neurons.

A. Forward propagation model

This section formally defines the forward propagation model of the CJUNNs-based architecture, i.e., the calculation and storage of intermediate variables from inputs to outputs. It generalizes the conventional layer-wise propagation. Let us denote with 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 weights matrix $\boldsymbol{W} \in \mathbb{R}^{(i+h)\times(h+o)}$, a connections 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 weights 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 connections 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 non-connection virtual node, whereas the *one-bias* neuron, when gets multiplied by its corresponding weight, act as a classical linear bias. Neurons output state is updated following the connections propagation order when initialising or modifying 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 [12].

Figure 1 shows a representative example of a CJUNNs-based model with its connections tensor. In particular, two competitive joint unstructured networks are represented in Figure 1a, let us say CJUNN$_1$ and CJUNN$_2$. Shared connections are represented with a solid line, whereas independent connections are represented with dashed and dotted line, 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) CJUNNs topology

(b) Connections tensor C

Fig. 1. Example of a CJUNNs-based model

A bold incremental number is also represented on the top-right of each node: it is an absolute ordering of all nodes serving as 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. According to the order theory, it is a partially ordered set, meaning that not every pair of elements can be ordered. This order relationship is fundamental to update neurons in the correct sequence. This unstructured architecture generalizes, and then can also represent, the conventional multi-layer perceptron.

Figure 1b shows the correspondent connections 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 relates to node n_0, which is fed by nodes 0, 1, and 2, following the solid and dashed connections arriving to 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 is related 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 is related 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 is related 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 CJUNNs-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 $E(\boldsymbol{x}, n)$ so that $\boldsymbol{T} = [E(\boldsymbol{x}, n), \boldsymbol{H}]$. 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 $\boldsymbol{IV} = G(\boldsymbol{T}, \boldsymbol{II})$. Similarly, the connections 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. Finally the postsynaptic state of the neuron η is selected for all the networks $\boldsymbol{ov}_i = \varphi(\sum_{j=1}^{m} \boldsymbol{IV}_{i,j}\boldsymbol{IW}_{i,j})$, and the hidden state matrix $\boldsymbol{H}$ is updated with the computed values $\boldsymbol{H}_{:,\eta} = \boldsymbol{ov}$. After repeating this process for all the neurons η, all the networks hidden states $\boldsymbol{o}_i = \sum_{j=1}^{n} \boldsymbol{\alpha}_j \boldsymbol{H}_{j,i}$ are linearly combined, and then the last o neurons are selected from the output state $\boldsymbol{o}$. The overall forward propagation process is schematized in Algorithm 1.

The next section formalizes the training model at the algorithmic level. An important premise is due at this point. The forward model as well as the training model are executed on a deep learning framework, by using an underlying representational model called computational graph. A computational graph is an efficient operational representation of the mathematical expressions. it is generated as a directed graph where the nodes correspond to mathematical operations. In particular, is the training model that is made efficient via the computational graph [13]. Indeed, an essential aspect of training is the numerical evaluation of the gradient during the optimization stage. For this purpose, the backpropagation algorithm allows an efficient gradient calculation. Backpropagation leverages the chain rule of differential calculus, computing the error gradients in terms of summations of local-gradient products over the various paths from a node to output. Using dynamic programming, deep learning frameworks can efficiently compute this summation over all paths.

```
Function $ForwardPropagation(\boldsymbol{x}, \boldsymbol{W}, \mathsf{C}, \boldsymbol{\alpha})$ is
    $\boldsymbol{H} \leftarrow 0$      // $\mathbb{R}^{n\times(h+o)}$
    for $\eta$ 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
```

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

B. Training model

As anticipated in the introductory section, two mechanisms of the training model guarantee the exploitation of the best networks and the exploration of the search space: (i) networks competition: at each early stop, 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 CJUNNs-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 networks competition

mechanism.
Given a input-output pair from the training dataset ($\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$) the backpropagation algorithm is used to compute the gradient of the error function $E(\boldsymbol{y}, \hat{\boldsymbol{y}})$ with respect to the parameters $\nabla_{\boldsymbol{W},\boldsymbol{\alpha}} E(ForwardPropagation(\hat{\boldsymbol{x}}, \boldsymbol{W}, \mathsf{C}, \boldsymbol{\alpha}), \hat{y})$, which are updated according to the Stochastic Gradient Descent (SGD) optimization algorithm.
As previously discussed, the networks competition is based on the network importance metric $\boldsymbol{\alpha}$. In order to guarantee an effective distinction of its elements, the softmax normalization function $\sigma(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$ is applied to the values computed of $\boldsymbol{\alpha}$. This avoids $\boldsymbol{\alpha}$ elements to converge 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 connections tensor C cannot be trained using backpropagation. For this reason, in order to ensure topological variability, the neuron entropy is used to apply a topological mutation. Specifically, the entropy of the neuron η is defined as $\boldsymbol{h}_\eta = -log(\boldsymbol{p}_\eta)\boldsymbol{p}_\eta$, where $\boldsymbol{p}_\eta$ is the probability of η to fire. The definition of neuron firing depends on the neuron activation function. For most activation functions is $y > th$ (for example $th = 0$ for ReLU or sigmoid). In the proposed approach, $\boldsymbol{p}_\eta$ is approximated for all the neurons using the Exponential Moving Average (EMA). If a neuron entropy $\boldsymbol{h}_\eta$ is lower than a target entropy T_h, then its incoming hidden neuron with the lowest entropy is selected and replaced with the *zero-bias* neuron. Similarly, if the neuron has a 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. The target entropy T_h is a hyper-parameter of the CJUNNs-based model.

III. Experimental Studies

The proposed architectural model has been implemented, tested, and publicly released on the Github platform [14], to foster its application on various research environments. For simplicity and readability the pseudo-code shown in the previous section omit the batch dimension. In the implementation and for all experiments a *batch size* of 16 has been set.

```
Function $TopologicalMutation(\mathsf{C}, \boldsymbol{p})$ is
    $\boldsymbol{h} \leftarrow -log(\boldsymbol{p})\boldsymbol{p}$
    for $\eta$ in $\{1, 2, \cdots, h + o\}$ do
        if $\boldsymbol{h}_\eta < T_h$ then
            $\mathsf{C} \leftarrow RemoveLowestIncomingEntropy(\mathsf{C}, \boldsymbol{h})$
            $\mathsf{C} \leftarrow ReplaceZeroWithHighestEntropy(\mathsf{C}, \boldsymbol{h})$
        end
    end
end

Function $NetworkElimination(\mathsf{C}, \boldsymbol{\alpha})$ is
    $\alpha_{min} \leftarrow argmin(\alpha)$
    $\mathsf{C} \leftarrow PruneNetwork(\mathsf{C}, \alpha_{min})$
    $\boldsymbol{\alpha} \leftarrow RemoveElement(\boldsymbol{\alpha}, \alpha_{min})$
end

Function Train() is
    $\boldsymbol{W} \leftarrow RandomWeights()$
    $\mathsf{C} \leftarrow RandomConnections()$
    $\boldsymbol{\alpha} \leftarrow RandomVector()$
    $\boldsymbol{p} \leftarrow 0$
    for epoch in $\{1, 2, \cdots, epochs\}$ do
        for $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ in trainingSet do
            if $epoch > warmup$ then
                $\mathsf{C} \leftarrow TopologicalMutation(\mathsf{C}, \boldsymbol{p})$
            end
            if ShouldEarlyStop(validationSet) then
                if NetworksNumber($\mathsf{C}$) $== 1$ then
                    return $\boldsymbol{W}$, $\mathsf{C}$
                else
                    $\mathsf{C} \leftarrow NetworkElimination(\mathsf{C}, \boldsymbol{\alpha})$
                end
            end
            $\boldsymbol{y}, \boldsymbol{H} \leftarrow ForwardPropagation(\hat{\boldsymbol{x}}, \boldsymbol{W}, \mathsf{C}, \sigma(\boldsymbol{\alpha}))$
            $\boldsymbol{p} \leftarrow EMA(ComputeActivation(\boldsymbol{H}))$
            $\nabla \leftarrow \nabla_{\boldsymbol{W},\boldsymbol{\alpha}} E(\boldsymbol{y}, \hat{\boldsymbol{y}})$
            $\boldsymbol{W}, \boldsymbol{\alpha} \leftarrow SGD(\nabla, \boldsymbol{W}, \boldsymbol{\alpha})$
        end
    end
end
```

Algorithm 2: Pseudo-code of the CJUNN training function

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 also the generality, the maturity, and the availability of dense NN pruning approaches, they can be efficiently and fairly compared with the proposed CJUNNs-based model. Given the above reasons, in this experimental section the CJUNN model is experimented and evaluated with respect to MLPs as a reference dense architecture, by adopting different pruning levels and methods. In all cases the pruning is made at the end of the training. Specifically, assuming 2 hidden layers, the considered approaches are:

- $MLP_{100\%}$: a non-pruned MLP;
- $MLP_{pr50\%}$: a 50% pruned MLP, removing connections with small weights;
- $MLP_{pr90\%}$: a 90% pruned MLP, removing connections

with small weights;

- $MLP_{lt50\%}$: a 50% pruned MLP, via the lottery ticket method;
- $MLP_{lt90\%}$: a 90% pruned MLP, via the lottery ticket method.

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 CJUNNs based models has been carried out using as objective function the accuracy on the validation set. To sample the hyperparameters to use for each run, the Tree-structured Parzen Estimator (TPE) has been used [15]. To prune unpromising runs, the Successive Halve Pruning (SHP) method has been used [16]. For each approach, the testing accuracy and the number of parameters have been considered as performance metrics.

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 I shows the hyperparameters for each model.

TABLE I
MODELS HYPERPARAMETERS

Hyperparameter	CJUNNs	$(MLP_{**\%})$
l (neurons in hidden layers)		✓
ρ (learning rate)	✓	✓
h (hidden nodes)	✓	
m (max incoming connections)	✓	
T_h (mutation entropy threshold)	✓	

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

- The *Iris Dataset* [17], which consists of 150 instances of sepal, petal length and width, of three iris plants classes (Setosa, Versicolour and Virginica);
- the *Seeds Dataset* [18], which is composed by 210 instances of 3 different varieties of wheat (Kama, Rosa and Canadian);
- the *Transfusion Dataset* [19], 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* [20], which contains values for 41 molecular descriptors used to classify 1055 chemicals into 2 classes (ready and not ready biodegradable);
- the *Plates Dataset* [21], consisting in steel plates faults, classified into 7 different class with 27 numerical features and 7 binary;
- the *KR-KP Dataset* [22] 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 in two classes (white can win or not);
- the *Robot Dataset* [23] 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);
- the *Nursery Dataset* [24] derived from a hierarchical decision model originally developed to rank applications for nursery schools, and composed by 12960 instances of 9 features describing the family state and divided in 5 classes (not recommend admission, mild recommend admission, recommend admission, recommend admission with priority and recommend admission with special priority).

Table II shows the dimensions of each dataset.

TABLE II
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	12960

To evaluate the effectiveness of the proposed approach in terms of performance metrics, Table III and Table IV show the accuracy and the parametric complexity for each model, respectively. 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 overcome the MLP pruned via the weight magnitude. In particular, the former with 50% pruning is equivalent to the dense MLP, whereas the latter sensibly looses accuracy.

In Table III 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 CJUNNs-based approach loses about two percentage points on the last 3 datasets.

However, in Table IV is apparent that the parametric complexity of the CJUNNs is dramatically lower, thus confirming the effectiveness of the proposed approach.

IV. CONCLUSIONS

The purpose of this paper is to formally introduce a novel perspective of optimization of sparse NN topologies, called Competitive Joint Unstructured NNs. In contrast to dense NN pruning methods, the proposed approach combines multiple natively sparse NNs that are joint because they share an amount of connections. Differently from natively sparse methods, the proposed technique does not require specific assertions and limitations. It is a hybrid method because the

TABLE III
ACCURACY OF EACH MODEL FOR EACH DATASET

Dataset	$MLP_{100\%}$	$MLP_{pr50\%}$	$MLP_{pr90\%}$	$MLP_{lt50\%}$	$MLP_{lt90\%}$	**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 IV
PARAMETRIC COMPLEXITY OF EACH MODEL FOR EACH DATASET

Dataset	$MLP_{100\%}$	$MLP_{pr50\%}$	$MLP_{pr90\%}$	$MLP_{lt50\%}$	$MLP_{lt90\%}$	**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	1.946	975	197	975	197	$\mathbf{148 \pm 2}$
Plates	2.416	1.208	245	1.208	245	$\mathbf{105 \pm 2}$
KR-KP	1.538	769	157	769	157	$\mathbf{180 \pm 3}$
Robot	1.460	730	149	730	149	$\mathbf{185 \pm 2}$
Nursery	3.204	1.602	321	1.602	321	$\mathbf{128 \pm 2}$

pruning is performed at the level of entire sparse network, on the basis of an NN importance metric that takes part in the gradient optimization, leading to a competition between NNs. A mutation mechanism at the connection level is also included to guarantee space exploration.

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 is able to sensibly outperform other dense NN pruning methods in terms of parametric complexity, by keeping almost the same accuracy.

FUNDING

Work partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence). This research has received partial funding from the European Commission H2020 Framework Programme under Grant No. 101017727 of the project “EXPERIENCE”. Work partially supported by the Italian Ministry of University and Research (MUR), in the framework of the FISR 2019 Programme, under Grant No. 03602 of the project “SERICA”.

REFERENCES

[1] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

[2] Simon Alford, Ryan Robinett, Lauren Milechin, and Jeremy Kepner. Pruned and structurally sparse neural networks. In *2018 IEEE MIT Undergraduate Research Technology Conference (URTC)*, pages 1–4. IEEE, 2018.

[3] Federico A Galatolo, Mario GCA Cimino, and Gigliola Vaglini. Formal derivation of mesh neural networks with their forward-only gradient propagation. *Neural Processing Letters*, pages 1–16, 2021.

[4] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[5] Paul Munro. *Backpropagation*, pages 73–73. Springer US, Boston, MA, 2010.

[6] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[7] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[8] Suraj Srinivas, Akshayvarun Subramanya, and R Venkatesh Babu. Training sparse neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 138–145, 2017.

[9] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[10] Ameya Prabhu, Girish Varma, and Anoop Namboodiri. Deep expander networks: Efficient deep networks from graph theory. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 20–35, 2018.

[11] Jeremy Kepner and Ryan Robinett. Radix-net: Structured sparse matrices for deep neural networks. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 268–274. IEEE,

2019.

[12] Federico Galatolo., Mario Cimino., and Gigliola Vaglini. 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,*, pages 830–836. INSTICC, SciTePress, 2019.

[13] Federico A. Galatolo, Mario Giovanni C. A. Cimino, and Gigliola Vaglini. Using stigmergy to incorporate the time into artificial neural networks. In Adrian Groza and Rajendra Prasath, editors, *Mining Intelligence and Knowledge Exploration*, pages 248–258, Cham, 2018. Springer International Publishing.

[14] Federico A. Galatolo. cjunn, https://github.com/galatolofederico/cjunn, 2021.

[15] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[16] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. *arXiv preprint arXiv:1810.05934*, 2018.

[17] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[18] Małgorzata Charytanowicz, Jerzy Niewczas, Piotr Kulczycki, Piotr A Kowalski, Szymon Łukasik, and Sławomir Żak. Complete gradient clustering algorithm for features analysis of x-ray images. In *Information technologies in biomedicine*, pages 15–24. Springer, 2010.

[19] I-Cheng Yeh, King-Jang Yang, and Tao-Ming Ting. Knowledge discovery on rfm model using bernoulli sequence. *Expert Systems with Applications*, 36(3):5866–5871, 2009.

[20] Kamel Mansouri, Tine Ringsted, Davide Ballabio, Roberto Todeschini, and Viviana Consonni. Quantitative structure–activity relationship models for ready biodegradability of chemicals. *Journal of chemical information and modeling*, 53(4):867–878, 2013.

[21] Massimo Buscema, Stefano Terzi, and WJ Tastle. A new meta-classifier. In *2010 Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS), Toronto, ON, Canada*, pages 1–7, 2010.

[22] AD Shapiro. The role of structured induction in expert systems, unpublished ph. *D thesis, University of Edinburgh*, 1983.

[23] Ananda L Freire, Guilherme A Barreto, Marcus Veloso, and Antonio T Varela. Short-term memory mechanisms in neural network learning of robot navigation tasks: A case study. In *2009 6th Latin American Robotics Symposium (LARS 2009)*, pages 1–6. IEEE, 2009.

[24] Blaz Zupan, Marko Bohanec, Ivan Bratko, and Janez Demsar. Machine learning by function decomposition. In *ICML*, pages 421–429, 1997.