Max-Plus Operators Applied to Filter Selection and Model Pruning in Neural Networks

Yunxiang Zhang[1,2,3](✉), Samy Blusseau[3], Santiago Velasco-Forero[3], Isabelle Bloch[2], and Jesús Angulo[3]

[1] Ecole Polytechnique, 91128 Palaiseau, France
yunxiang.zhang@polytechnique.edu
[2] LTCI, Télécom ParisTech, Université Paris-Saclay, Paris, France
[3] Centre for Mathematical Morphology, Mines ParisTech, PSL Research University, Fontainebleau, France
samy.blusseau@mines-paristech.fr

Abstract. Following recent advances in morphological neural networks, we propose to study in more depth how Max-plus operators can be exploited to define morphological units and how they behave when incorporated in layers of conventional neural networks. Besides showing that they can be easily implemented with modern machine learning frameworks, we confirm and extend the observation that a Max-plus layer can be used to select important filters and reduce redundancy in its previous layer, without incurring performance loss. Experimental results demonstrate that the filter selection strategy enabled by a Max-plus layer is highly efficient and robust, through which we successfully performed model pruning on two neural network architectures. We also point out that there is a close connection between Maxout networks and our pruned Max-plus networks by comparing their respective characteristics. The code for reproducing our experiments is available online (for code release, please visit https://github.com/yunxiangzhang.).

Keywords: Mathematical morphology · Morphological neural networks · Max-plus operator · Deep learning · Filter selection · Model pruning

1 Introduction

During the previous era of high interest in artificial neural networks, in the late 1980s, morphological networks [4,27] were introduced merely as an alternative to their linear counterparts. In a nutshell, they consist in changing the elementary neural operation from the usual scalar product to dilations, erosions and more general rank filters. Besides the practical achievements in this research line, which reached state-of-the-art results at its time [19], it questioned the main practices in the field of artificial neural networks on crucial topics such as architectures and optimization methods, improving their understanding.

B. Burgeth et al. (Eds.): ISMM 2019, LNCS 11564, pp. 310–322, 2019.
https://doi.org/10.1007/978-3-030-20867-7_24

After the huge technological leap taken by deep neural networks in the last decade, understanding them remains challenging and insights can still be brought by testing alternatives to the most popular practices. However, since the emergence of highly efficient deep learning frameworks (Caffe, Torch, TensorFlow, *etc.*), morphological neural networks have been very little re-investigated [2,17]. Motivated by the promising results in [2], this paper proposes to extend and deepen the study on morphological neural networks with Max-plus layers. More precisely, we aim to validate and exploit a specific property, namely that Max-plus layer (*i.e.* a dilation layer) following a conventional linear layer tends to select a reduced number of filters from the latter, making the others useless.

Our findings and contributions are three-fold. First, we propose an efficient training framework for morphological neural networks with Max-plus blocks, and theoretically demonstrate that they are universal function approximators under mild conditions. Secondly, we perform extensive experiments and visualization analysis to validate the robustness and effectiveness of the filter selection property provided by Max-plus blocks. Thirdly, we successfully applied the resulting Max-plus blocks to the task of model pruning on different neural network architectures. Related work is briefly summarized in Sect. 2. Then we show how to introduce Max-plus operators in neural networks in Sect. 3. Experimental results are discussed in Sect. 4.

2 Related Work

Morphological neural networks were defined almost simultaneously in two different ways at the end of the 1980s [4,27]. Davidson [4] introduced neural units that can be seen as pure dilations or erosions, whereas Wilson [27] focused on a more general formulation based on rank filters, in which max and min operators are two particular cases. Davidson's definition interprets morphological neurons as bounding boxes in the feature space [20,21,24]. In the latter studies, networks were trained to perform perfectly on training sets after few iterations, but little attention was drawn to generalization. Only recently, a backpropagation-based algorithm was adopted and improved constructive ones [28]. Still, the "bounding-box" approach does not seem to generalize well to test set when faced with high-dimensional problems like image analysis.

Wilson's idea, on the other hand, inspired hybrid linear/rank filter architectures, which were trained by gradient descent [18] and backpropagation [19]. In this case, the geometrical interpretation of decision surfaces becomes much richer, and the resulting framework was successfully applied to an image classification problem. The previously mentioned study [2] is one of the latest in this area, introducing a hybrid architecture that experimentally shows an interesting property on network pruning. In this classification experiment [2], each Max-plus unit shows, after training, one or two large weight values compared to the others. At the inference stage, this non-uniform distribution of weight values induces a selection of important filters in the previous layer, whereas the other filters (the majority) are no longer used in the subsequent classification task.

Therefore, after removal of the redundant filters, the network behaves exactly as it did before pruning. In this paper, we focus on the exploration of this property and show that it becomes more stable and effective provided that a proper regularization is applied.

Model pruning stands for a family of algorithms that explore the redundancy in model parameters and remove unimportant ones while not compromising the performance. Rapidly increased computational and storage requirements for deep neural networks in recent years have largely motivated research interests in this domain. Early works on model pruning dates back to [9,13], which prune model parameters based on the Hessian of the loss function. More recently, the model pruning problem was addressed by first dropping the neuronal connections with insignificant weight value and then fine-tuning the resulting sparse network [8]. This technique was later integrated into the Deep Compression framework [7] to achieve even higher compression rate. The HashNets model [3] randomly groups parameters into hash buckets using a low-cost hash function and performs model compression *via* parameter sharing. While previous methods achieved impressive performance in terms of compression rate, one notable disadvantage of these non-structured pruning algorithms lies in the sparsity of the resulting weight matrices, which cannot lead to speedup without dedicated hardwares or libraries.

Various structured pruning methods were proposed to overcome this subtlety in practical applications by pruning at the level of channels or even layers. Filters with smaller L_1 norm were pruned based on a predefined pruning ratio for each layer in [14]. Model pruning was transformed into an optimization problem in [16] and the channels to remove were determined by minimizing next layer reconstruction error. A branch of algorithms in this category employs L_p regularization to induce shrinkage and sparsity in model parameters. Sparsity constraints were imposed in [15] on channel-wise scaling factors and pruning was based on their magnitude, while in [26] group-sparsity was leveraged to learn compact CNNs via a combination of L_1 and L_2 regularization. One minor drawback of these regularization-based methods is that the training phase generally requires more iterations to converge. Our approach also falls into the structured pruning category and thus no dedicated hardware or libraries are required to achieve speedup, yet no regularization is imposed during model training. Moreover, in contrast to most existing pruning algorithms, our method does not need fine-tuning to regain performance.

3 Max-Plus Operator as a Morphological Unit

3.1 Morphological Perceptron

In traditional literature on machine learning and neural networks, a perceptron [22] is defined as a linear computational unit, possibly followed by a non-linear activation function. Among all popular choices of activation functions, such as logistic function, hyperbolic tangent function and rectified linear unit

(ReLU) function, ReLU [5] generally achieves better performance due to its simple formulation and non-saturating property. Instead of multiplication and addition, the morphological perceptron employs addition and maximum, which results in a non-linear computational unit. A simplified version [2] of the initial formulation [4,20] is defined as follows.

Definition 1 (Morphological Perceptron). *Given an input vector* $\mathbf{x} \in \mathbb{R}^n_{max}$ *(with* $\mathbb{R}_{max} = \mathbb{R} \cup \{-\infty\}$*), a weight vector* $\mathbf{w} \in \mathbb{R}^n_{max}$*, and a bias* $\mathbf{b} \in \mathbb{R}_{max}$*, the morphological perceptron computes its activation as:*

$$a(\mathbf{x}) = \max\left\{\mathbf{b}, \max_{i\in\{1,\ldots,n\}} \{\mathbf{x}_i + \mathbf{w}_i\}\right\} \tag{1}$$

where $\mathbf{x}_i$ *(resp.* $\mathbf{w}_i$*) denotes the i-th component of* $\mathbf{x}$ *(resp.* $\mathbf{w}$*).*

This model may also be referred to as $(\max, +)$ perceptron since it relies on the $(\max, +)$ semi-ring with underlying set $\mathbb{R}_{max}$. It is a dilation on the complete lattice $\left((\mathbb{R} \cup \{\pm\infty\})^n, \leq_n\right)$ with $\leq_n$ the Pareto ordering.

3.2 Max-Plus Block

Based on the formulation of the morphological perceptron, we define the Max-plus block as a standalone module that combines a fully-connected layer (or convolutional layer) with a Max-plus layer [2]. Let us denote the input vector of the fully-connected layer[1], the input and output vectors of the Max-plus layer respectively by $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$, whose components are indexed by $i \in \{1, \ldots, I\}$, $j \in \{1, \ldots, J\}$ and $k \in \{1, \ldots, K\}$, respectively. The corresponding weight matrices are denoted by $\mathbf{w}^f \in \mathbb{R}^{I\times J}_{max}$ and $\mathbf{w}^m \in \mathbb{R}^{J\times K}_{max}$. Then the operation performed in this Max-plus block is (see Fig. 1):

$$\begin{aligned} \mathbf{y}_j &= \textstyle\sum_{i\in\{1,\ldots,I\}} \mathbf{x}_i \cdot \mathbf{w}^f_{ij} \\ \mathbf{z}_k &= \max_{j\in\{1,\ldots,J\}} \left\{\mathbf{y}_j + \mathbf{w}^m_{jk}\right\} \end{aligned} \tag{2}$$

Note that the bias vector of the fully-connected layer (convolutional layer) is removed in our formulation, since its effect overlaps with that of the weight matrix $\mathbf{w}^m$. In addition, the bias vector of the Max-plus layer is shown to be ineffective in practice and is therefore not used here.

3.3 Universal Function Approximator Property

The result presented here is very similar to the approximation theorem on Maxout networks[2] [6], based on Wang's work [25]. As shown in [6], Maxout networks with enough affine components in each Maxout unit are universal function approximators. Recall that a model is called a universal function approximator if it can approximate arbitrarily well any continuous function provided

[1] This formulation can be easily generalized to the case of convolutional layers.

[2] Note that the classical universal approximation theorems for neural networks (see for example [10]) do not hold for networks containing max-plus units.

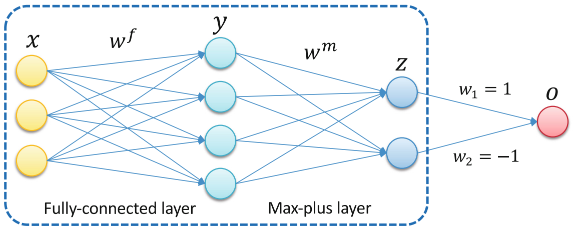

Fig. 1. A Max-plus block with two output units, applied to an input vector x.

enough capacity. Similarly, provided that the input vector (or input feature maps) $\mathbf{y} \in \mathbb{R}^J_{max}$ of the Max-plus layer may have arbitrarily many affine components (or affine feature maps), we show that a Max-plus model with just two output units in its Max-plus block can approximate arbitrarily well any continuous function of the input vector (or input feature maps) $\mathbf{x} \in \mathbb{R}^I$ of the block on a compact domain. A diagram illustrating the basic idea of the proof is shown in Fig. 1.

Theorem 1 (Universal function approximator). *A Max-plus model with two output units in its Max-plus block can approximate arbitrarily well any continuous function of the input of the block on a compact domain.*

Proof. We provide here a sketch of the proof, which follows very closely the one of Theorem 4.3 in [6]. By Proposition 4.2 in [6], any continuous function defined on a compact domain $C \subset \mathbb{R}^I$ can be approximated arbitrarily well by a piecewise linear (PWL) continuous function g, composed of k affine regions. By Proposition 4.1 in [6], there exist two matrices $W_1, W_2 \in \mathbb{R}^{k \times I}$ and two vectors $b_1, b_2 \in \mathbb{R}^k$ such that

$$\forall x \in C,\ g(x) = \max_{1 \leq j \leq k} \{W_{1j}x + b_{1j}\} - \max_{1 \leq j \leq k} \{W_{2j}x + b_{2j}\}, \tag{3}$$

where W_{ij} is the j-th row of matrix W_i and b_{ij} the j-th coefficient of b_i, $i = 1, 2$. Now, Eq. 3 is the output of the Max-plus block of Fig. 1, provided $\mathbf{w}^f = [W_1; W_2] \in \mathbb{R}^{2k \times I}$ is the matrix of the fully connected layer, $\mathbf{w}^m_1 = [b_1^T, -\infty, \ldots, -\infty] \in \mathbb{R}^{2k}_{max}$ and $\mathbf{w}^m_2 = [-\infty, \ldots, -\infty, b_2^T] \in \mathbb{R}^{2k}_{max}$ the two rows of $\mathbf{w}^m$. This concludes the proof.

4 Experiments

In this section, we present experimental results for our Max-plus blocks by integrating them in different types of neural networks. All neural network models shown in this section are implemented with the open-source machine learning library TensorFlow [1] and trained on benchmark datasets MNIST [12] or CIFAR-10 [11] depending on the model complexity and capability.

4.1 Filter Selection Property

In an attempt to reproduce and confirm the experimental results reported in [2], we first implemented a simple Max-plus model composed of a fully-connected layer with J units followed by a Max-plus layer with ten units, namely a Max-plus block in our terminology, to perform image classification on MNIST dataset. In contrast to the original formulation in [2], the two bias vectors are removed for practical concerns explained in Sect. 3.2.

Table 1 summarizes the classification accuracy achieved by this simple model on the validation set of MNIST dataset for different values of J in columns 3 to 5. Note that all the experiments contained in these 3 columns are conducted under the same training setting (initial learning rate, learning rate decay steps, batch size, optimizer, *etc.*) except for parameter initialization (each column corresponds to a different random seed). The performance of the original model in [2] is included in column 2 for comparison. As shown in Table 1, provided a proper initialization, our simple Max-plus model generally achieves an improved performance compared to the original model. More interestingly, through horizontal comparison across different runs, we find that the performance of this naive Max-plus model is highly sensitive to parameter initialization.

Table 1. Classification accuracy on the validation set of MNIST dataset with three different random seeds.

J units	Model [2]	Max-plus			Max-plus + dropout		
24	84.3%	76.7%	**84.4%**	76.0%	93.7%	93.7%	**94.2%**
32	84.8%	84.5%	76.5%	**94.0%**	**94.6%**	94.2%	93.7%
48	84.6%	**94.0%**	93.8%	75.8%	**94.6%**	94.5%	94.5%
64	92.1%	85.4%	94.7%	**94.9%**	94.8%	**95.1%**	94.8%
100	–	**94.8%**	85.3%	93.7%	94.8%	**95.5%**	95.2%
144	–	**95.1%**	85.8%	85.4%	95.3%	95.7%	**95.9%**

In order to gain more insight into this instability problem, we follow the approach of [2] and visualize the weight matrix of the Max-plus layer $\mathbf{w}^m \in \mathbb{R}_{max}^{J\times 10}$ by splitting and reshaping it into 10 gray-scale images $\mathbf{w}^m_{\cdot 1} \dots \mathbf{w}^m_{\cdot i} \dots \mathbf{w}^m_{\cdot 10}$, each corresponding to a specific class (Fig. 2, left). In addition we also visualized, as on the right hand side of Fig. 2, the ten linear filters from the fully-connected layer that correspond to the maximum value of each weight vector $\mathbf{w}^m_{\cdot i}$, defined as:

$$\text{Filter}_i = \mathbf{w}^f_{\cdot j^{(i)}_{\max}} \in \mathbb{R}^I_{max} \quad \text{where } j^{(i)}_{\max} = \underset{j\in\{1,\dots,J\}}{\arg\max} \left\{\mathbf{w}^m_{ji}\right\}. \tag{4}$$

Figure 2 shows specifically the weights of the Max-plus model that achieves an accuracy of 85.8% with $J = 144$ units (fifth column and last row in Table 1). We notice that there exists a severe filter-collision problem in the Max-plus block, namely different output units select the same linear filter in the fully-connected layer to compute their outputs. More specifically, class 3 and class 8 share the same *argmax* (highlighted by red circles) in their weight vectors and consequently select the same linear filter (visualized filters for class 3 and class 8 are the same). This collision between output units directly leads to classification confusion (because the Max-plus layer is the last layer in this simple model) since the classes that employ the same linear filter are completely indistinguishable. Furthermore, we only observed this filter-collision problem in the experiments that achieve relatively poor performance compared to the others (same model with different initialization). This observation consistently verifies our hypothesis that filter-collision is at the root of lower performance.

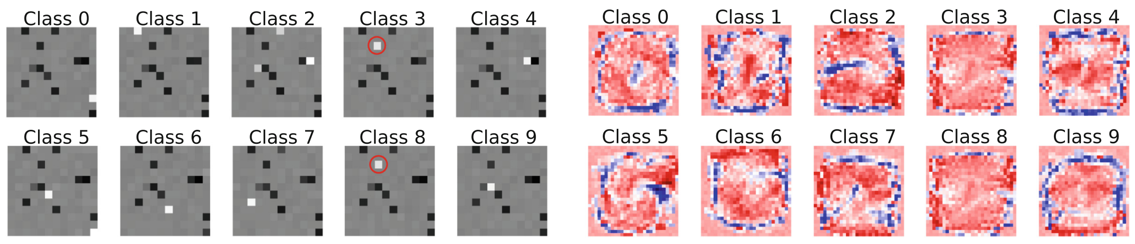

Fig. 2. Visualization of the weight matrix $\mathbf{w}^m$ (left) and the 10 linear filters that correspond to the maximum value of each weight vector $\mathbf{w}^m_{\cdot i}$ (right). Visualization of the weight matrix $\mathbf{w}^m$ (left) and the 10 linear filters that correspond to the maximum value of each weight vector $\mathbf{w}^m_{\cdot i}$ (right). (Color figure online)

In order to separate the output units that got stuck with the same linear filter, we applied a dropout regularization [23] to randomly switch off the neuronal connections between the fully-connected layer and the Max-plus layer during training. Empirically, we found this approach highly effective, as the performance of the dropout-regularized Max-plus model becomes much less sensitive to parameter initialization. Table 1 shows that the Max-plus model with dropout regularization achieves both a better performance and more stable results. We further validate the effect provided by dropout regularization through an additional experiment. With $J = 144$, we carried out 25 runs with different initializations for different dropout ratios. As shown in Fig. 3, dropout reduces dramatically the variability across experiments. The interpretation of this improvement is that dropout forces each class to use more than one linear filter to represent its corresponding digit. This allows for a more general representation and limits the risk of collisions between classes. Interestingly, we observe a slight accuracy drop when dropout ratio surpasses a certain level. This indicates that a trade-off between stability and performance needs to be found, although a large range of dropout values (between 25% and 75%) show robustness to random seed values without penalizing the effectiveness on classification accuracy.

Whereas in general (with or without dropout) linear filters $\mathbf{w}^f_{\cdot j(i)}$ that correspond to large values of $\mathbf{w}^m_{\cdot i}$ are similar to the images in Fig. 2 (right), *i.e.* digit-like shape with high contrast, those corresponding to smaller values in

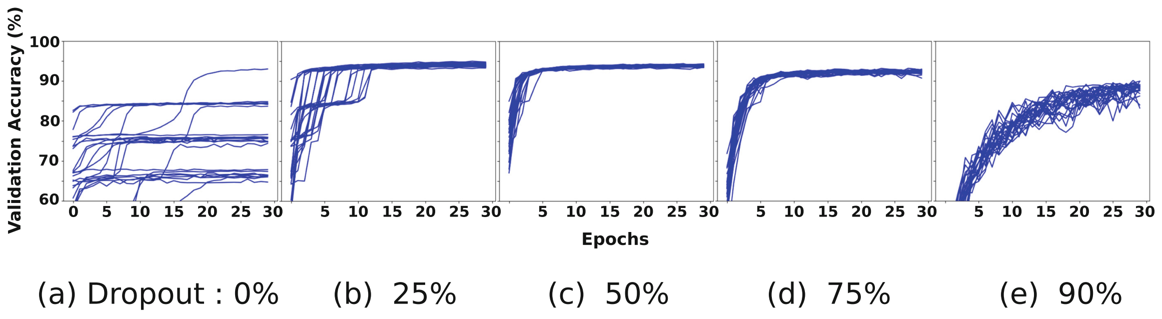

Fig. 3. Classification accuracy per epochs for 25 runs with different random initialization and dropout ratios on MNIST validation set. Dropout values between 25% and 75% in (b–d) show robustness to random seed values without penalizing the effectiveness on classification accuracy.

the weight matrix $\mathbf{w}^m$ are noisy or low-contrast images showing the shape of a specific digit. This means that these filters were activated by few training examples. Figure 4 (left) shows several linear filters of this kind. The visualization above suggests that large values in $\mathbf{w}^m$ correspond to linear filters of the fully-connected layer that contain rich information for the subsequent classification task, and hence that strongly respond to the samples of a specific class. If that holds, then it is likely that only these filters shall achieve the maximum in Eq. (2), and contribute to the classification output. Therefore, we might be able to reduce the complexity of a model while not degrading its performance by exploiting this filter selection property of Max-plus layers.

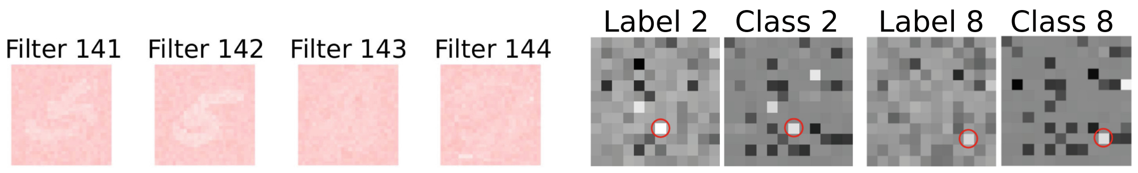

Fig. 4. Visualization of several linear filters corresponding to small values in $\mathbf{w}^m$ (left) and comparison between the activations of the fully-connected layer for two training examples and their corresponding weight vectors in $\mathbf{w}^m$ (right).

In order to further validate the stability of this connection before taking advantage of it, we also visualized the activations of the fully-connected layer as gray-scale images for several training examples and compare[3] them to the visualization of $\mathbf{w}^m$. As shown in Fig. 4 (right), there is a clear correspondence between the maximum activation value of a training example with label i and the largest two or three values of the weight vector $\mathbf{w}^m_{\cdot(i+1)}$, which indicates that the linear filters corresponding to large values in the weight matrix $\mathbf{w}^m$ are effectively used for the subsequent classification task.

[3] For example, if a training example has label i, then we compare its activation vector $\mathbf{y} \in \mathbb{R}^J_{max}$ with the weight vector $\mathbf{w}^m_{\cdot(i+1)} \in \mathbb{R}^J_{max}$.

4.2 Application to Model Pruning

Now that our approach to filter selection via Max-plus layers is proved to be quite effective and stable, we formalize our model pruning strategy as follows: given a fixed threshold $s \in [0, 1]$, for each weight vector $\mathbf{w}^m_{\cdot(i+1)}$, we only keep the values that are larger than $s \times \max_{j\in\{1,\dots,J\}} \left\{\mathbf{w}^m_{j(i+1)}\right\} + (1-s) \times \min_{j\in\{1,\dots,J\}} \left\{\mathbf{w}^m_{j(i+1)}\right\}$ and the linear filters that correspond to these retained values. Therefore, if in total J_r linear filters are kept in the pruned model, then the remaining parameters in the Max-plus layer no longer form a weight matrix but a weight vector of size J_r, where each entry corresponds to a linear filter in the fully-connected layer. The pruned fully-connected layer and the pruned Max-plus layer combined together perform a standard linear transformation followed by a maximum operation over uneven groups. Note that the pruning process is conducted independently for each output unit $\mathbf{z}_k$ of the Max-plus block ($1 \leq k \leq 10$), thus the number of retained linear filters in the pruned model may vary from one class to another. From now on, we shall call these selected linear filters by *active filters* and the others by *non-active filters*. Figure 5 shows a graphical illustration of the comparison between the original model and the pruned model.

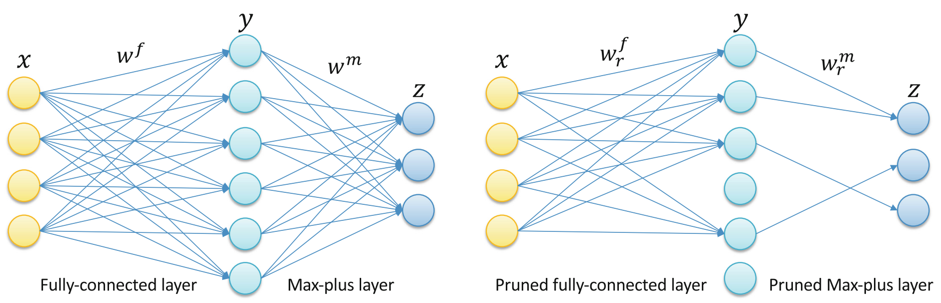

Fig. 5. Illustration of the comparison between the original Max-plus model (left) and the pruned Max-plus model (right). Here we have $\mathbf{w}^f_r \in \mathbb{R}^{I\times J_r}_{max}$ and $\mathbf{w}^m_r \in \mathbb{R}^{J_r}_{max}$.

We tried different pruning levels on this simple Max-plus model by varying the threshold s and tested the pruned models on the validation set and test set of MNIST dataset. We plotted the resulting classification accuracy in function of the number of active filters in Fig. 6. The performance of a single-layer softmax model and a single-layer Maxout model (number of affine components in each Maxout unit is two) is also provided for comparison.

As we can see on the diagram, the performance of the pruned Max-plus model is quite inferior to that of a single-layer softmax model when only one active filter is allowed to be selected for each class, *i.e.* the threshold is fixed to 1.0. However, as we relaxed the constraint on the number of total active filters by decreasing the threshold, the accuracy recovers rapidly and approaches that of the unpruned Max-plus model in a monotonic way. With exactly 24 active

J_r	Accuracy	J_r	Accuracy
10	82.7%	18	90.4%
11	86.4%	19	94.6%
12	86.4%	20	94.7%
13	89.9%	21	94.7%
14	90.0%	22	94.8%
15	90.2%	23	94.8%
16	90.2%	24	95.7%
17	90.3%	25	95.7%

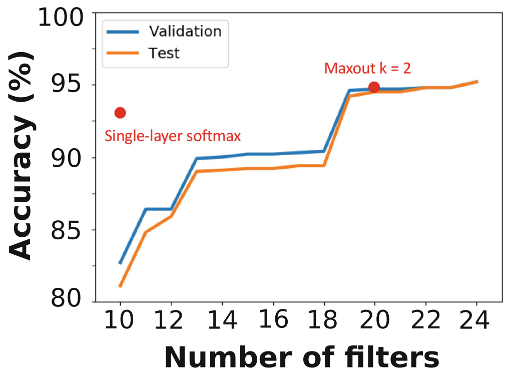

Fig. 6. Classification accuracy of pruned Max-plus model in function of the number of retained active filters.

filters retained in the pruned model, we achieve a full-recovery of the original Max-plus model performance, which means that the other 120 linear filters do not contribute to the classification task. Moreover, we can achieve comparable performance as the 2-degree Maxout model with roughly the same amount of parameters, which again validates the effectiveness of our Max-plus models.

With the same method, we successfully performed model pruning on a much more challenging CNN model by replacing the last fully-connected layer with a Max-plus layer. In order to facilitate the training of deep Max-plus model, we resort to transfer learning by initializing the two convolutional layers with pre-trained weights. The pruned Max-plus model achieves slightly better performance than the CNN model while reducing 94.8% of parameters of the second last fully-connected layer and eliminating the last fully-connected layer compared to the CNN model. Note that we could achieve a full-recovery of the unpruned Max-plus model performance with only ten active filters in this case, namely one linear filter for each output unit. Table 2 summarizes the architectures of the CNN model, the unpruned Max-plus model and the pruned Max-plus model, along with their classification accuracy on the test set of CIFAR-10 dataset.

Table 2. The architectural specifications of the CNN model, unpruned and pruned Max-plus model, along with their performance on the test set of CIFAR-10 dataset.

CNN	Max-plus	Pruned Max-plus
conv(5*5)	conv(5*5)	conv(5*5)
maxpool(2*2)	maxpool(2*2)	maxpool(2*2)
conv(5*5)	conv(5*5)	conv(5*5)
maxpool(2*2)	maxpool(2*2)	maxpool(2*2)
fc(384)	fc(384)	fc(384)
fc(192)	fc(192)	fc(10)
fc(10)	maxplus(10)	maxplus(10)
83.5%	83.9%	83.9%

4.3 Comparison to Maxout Networks

It is noticeable that the pruned Max-plus network differs from Maxout networks only in their grouping strategy for maximum operations. Maxout networks impose this rigid constraint in a way that each Maxout unit has an equal number of affine components while Max-plus networks are more tolerant in this respect. Figure 7 shows a graphical illustration of this comparison.

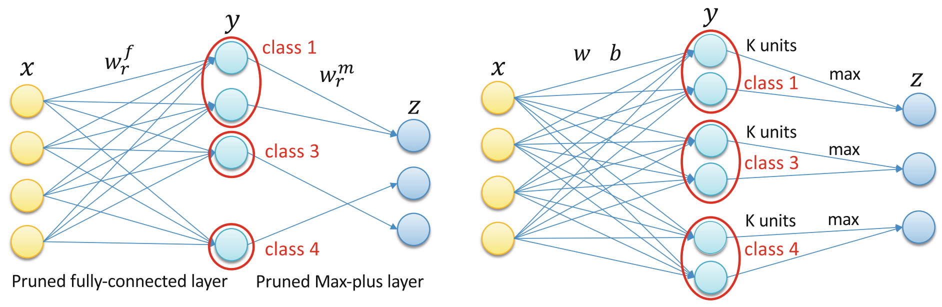

Fig. 7. Illustration of the comparison between the pruned Max-plus model (left) and Maxout model (right).

This higher flexibility hence endows Max-plus blocks with the capability of adapting the number of active filters used for each output unit accordingly. If a latent concept (say digit 1, which can be easily confused with digit 7) is considerably tougher to capture than some other concepts (say digits 3, 4 and 8, each of which has a relatively unique shape among the ten Arabic numbers), then the Max-plus layer will select more linear filters to abstract it than for the others. For example, the partition of the 24 active filters for the ten digit classes in Sect. 4.2 is $[2, 3, 3, 1, 2, 3, 3, 3, 2, 2]$, which is consistent with our point. This adaptive behavior of the filter selection property of Max-plus blocks makes the pruned Max-plus network more computationally efficient (fewer model parameters, smaller run-time memory footprint and faster inference) and is highly desirable in real-life applications.

5 Conclusions and Future Work

In this work we went a step further on a very new and promising topic, namely the reduction of deep neural networks with Max-plus blocks. Our experiments show strong evidence that model pruning via this method is compatible with high performance when a proper dropout regularization is applied during training. This was tested on data and architectures of variable complexity. Just as interesting as the obtained results are the many questions raised by these new insights. In particular, training these architectures is a challenging task which requires a better understanding on them, both theoretical and practical. We observed that training a deep model containing a Max-plus block is not straightforward, as we needed to resort to transfer learning. New optimization tricks will be needed to train deep architectures with *several* Max-plus blocks. The extension to a convolutional version of Max-plus blocks is also an open question, which we hope can be addressed based on the elements provided by this work.

Acknowledgements. This work was partially funded by a grant from Institut Mines Telecom.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., et al.: TensorFlow: a system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation, vol. 16, pp. 265–283 (2016)
2. Charisopoulos, V., Maragos, P.: Morphological perceptrons: geometry and training algorithms. In: Angulo, J., Velasco-Forero, S., Meyer, F. (eds.) ISMM 2017. LNCS, vol. 10225, pp. 3–15. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57240-6_1
3. Chen, W., Wilson, J., Tyree, S., Weinberger, K., Chen, Y.: Compressing neural networks with the hashing trick. In: International Conference on Machine Learning, pp. 2285–2294 (2015)
4. Davidson, J.L., Ritter, G.X.: Theory of morphological neural networks. In: Digital Optical Computing II, vol. 1215, pp. 378–389 (1990)
5. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: 14th International Conference on Artificial Intelligence and Statistics, pp. 315–323 (2011)
6. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. arXiv preprint arXiv:1302.4389 (2013)
7. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. In: International Conference on Learning Representations (2015)
8. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, pp. 1135–1143 (2015)
9. Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: optimal brain surgeon. In: Advances in Neural Information Processing Systems, pp. 164–171 (1993)
10. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989)
11. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. University of Toronto, Technical report (2009)
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
13. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in Neural Information Processing Systems, pp. 598–605 (1990)
14. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient ConvNets. In: International Conference on Learning Representations (2016)
15. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: ICCV, pp. 2755–2763 (2017)

16. Luo, J., Wu, J., Lin, W.: ThiNet: a filter level pruning method for deep neural network compression. In: ICCV, pp. 5068–5076 (2017)
17. Mondal, R., Santra, S., Chanda, B.: Dense morphological network: an universal function approximator. arXiv preprint arXiv:1901.00109 (2019)
18. Pessoa, L.F., Maragos, P.: MRL-filters: a general class of nonlinear systems and their optimal design for image processing. IEEE Trans. Image Process. **7**(7), 966–978 (1998)
19. Pessoa, L.F., Maragos, P.: Neural networks with hybrid morphological/rank/linear nodes: a unifying framework with applications to handwritten character recognition. Pattern Recogn. **33**(6), 945–960 (2000)
20. Ritter, G.X., Sussner, P.: An introduction to morphological neural networks. In: 13th International Conference on Pattern Recognition, vol. 4, pp. 709–717 (1996)
21. Ritter, G., Urcid, G.: Lattice algebra approach to single-neuron computation. IEEE Trans. Neural Netw. **14**(2), 282–295 (2003)
22. Rosenblatt, F.: Principles of neurodynamics: perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc., Buffalo, NY (1961)
23. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
24. Sussner, P., Esmi, E.L.: Morphological perceptrons with competitive learning: lattice-theoretical framework and constructive learning algorithm. Inf. Sci. **181**(10), 1929–1950 (2011)
25. Wang, S.: General constructive representations for continuous piecewise-linear functions. IEEE Trans. Circ. Syst. I Regul. Pap. **51**(9), 1889–1896 (2004)
26. Wen, W., Wu, C., et al.: Learning structured sparsity in deep neural networks. In: Advances in Neural Information Processing Systems, pp. 2074–2082 (2016)
27. Wilson, S.S.: Morphological networks. In: Visual Communications and Image Processing IV, vol. 1199, pp. 483–496 (1989)
28. Zamora, E., Sossa, H.: Dendrite morphological neurons trained by stochastic gradient descent. Neurocomputing **260**, 420–431 (2017)