Convolutional Rectifier Networks as Generalized Tensor Decompositions

Nadav Cohen COHENNADAV@CS.HUJI.AC.IL
Amnon Shashua SHASHUA@CS.HUJI.AC.IL
The Hebrew University of Jerusalem

Abstract

Convolutional rectifier networks, i.e. convolutional neural networks with rectified linear activation and max or average pooling, are the cornerstone of modern deep learning. However, despite their wide use and success, our theoretical understanding of the expressive properties that drive these networks is partial at best. On the other hand, we have a much firmer grasp of these issues in the world of arithmetic circuits. Specifically, it is known that convolutional arithmetic circuits possess the property of "complete depth efficiency", meaning that besides a negligible set, all functions realizable by a deep network of polynomial size, require exponential size in order to be realized (or approximated) by a shallow network. In this paper we describe a construction based on generalized tensor decompositions, that transforms convolutional arithmetic circuits into convolutional rectifier networks. We then use mathematical tools available from the world of arithmetic circuits to prove new results. First, we show that convolutional rectifier networks are universal with max pooling but not with average pooling. Second, and more importantly, we show that depth efficiency is weaker with convolutional rectifier networks than it is with convolutional arithmetic circuits. This leads us to believe that developing effective methods for training convolutional arithmetic circuits, thereby fulfilling their expressive potential, may give rise to a deep learning architecture that is provably superior to convolutional rectifier networks but has so far been overlooked by practitioners.

1. Introduction

Deep neural networks are repeatedly proving themselves to be extremely effective machine learning models, providing state of the art accuracies on a wide range of tasks. Arguably, the most successful deep learning architecture to date is that of convolutional neural networks (*ConvNets*, (LeCun and Bengio, 1995)), which prevails in the field of computer vision, and is recently being harnessed for many other application domains as well (*e.g.* (Shen et al., 2014; Wallach et al., 2015)). Modern ConvNets are formed by stacking layers one after the other, where each layer consists of a linear convolutional operator followed by Rectified Linear Unit (*ReLU*) activation ($\sigma(z) = \max\{0, z\}$), which in turn is followed by max or average pooling ($P\{c_j\} = \max\{c_j\}$ or $P\{c_j\} = \text{mean}\{c_j\}$ respectively). Such models, which we refer to as *convolutional rectifier networks*, have driven the resurgence of deep learning (Krizhevsky et al., 2012), and represent the cutting edge of the ConvNet architecture.

Despite their empirical success, and the vast attention they are receiving, our theoretical understanding of convolutional rectifier networks is limited. It is believed that they enjoy *depth efficiency*, *i.e.* that when allowed to go deep, such networks can implement with polynomial size computations that would require super-polynomial size if the networks were shallow. However, formal arguments that support this are scarce. It is unclear to what extent convolutional rectifier networks leverage depth efficiency, or more formally, what is the proportion of weight settings that would lead a deep network to implement a computation that cannot be efficiently realized by a shallow network. We refer to the most optimistic situation, where this takes place for all weight settings but a negligible (zero measure) set, as *complete depth efficiency*.

Compared to convolutional rectifier networks, our theoretical understanding of depth efficiency for arithmetic circuits, and in particular for convolutional arithmetic circuits, is much more developed. *Arithmetic circuits* (also known as Sum-Product Networks, (Poon and Domingos, 2011)) are networks with two types of nodes: sum nodes, which compute a weighted sum of their inputs, and product nodes, computing the product of their inputs. The depth efficiency of arithmetic circuits has been studied by the theoretical computer science community for the last five decades, long

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).

before the resurgence of deep learning. Although many problems in the area remain open, significant progress has been made over the years, making use of various mathematical tools. *Convolutional arithmetic circuits* form a specific sub-class of arithmetic circuits. Namely, these are ConvNets with linear activation ($\sigma(z) = z$) and product pooling ($P\{c_j\} = \prod c_j$). Recently, (Cohen et al., 2016b) analyzed convolutional arithmetic circuits through tensor decompositions, essentially proving, for the type of networks considered, that *depth efficiency holds completely*. Although convolutional arithmetic circuits are known to be equivalent to SimNets (Cohen and Shashua, 2014), a new deep learning architecture that has recently demonstrated promising empirical performance (Cohen et al., 2016a), they are fundamentally different from convolutional rectifier networks. Accordingly, the result established in (Cohen et al., 2016b) does not apply to the models most commonly used in practice.

In this paper we present a construction, based on the notion of *generalized tensor decompositions*, that transforms convolutional arithmetic circuits of the type considered in (Cohen et al., 2016b) into convolutional rectifier networks. We then use the available mathematical tools from the world of arithmetic circuits to prove new results concerning the expressive power and depth efficiency of convolutional rectifier networks. Namely, we show that with ReLU activation, average pooling leads to loss of universality, whereas max pooling is universal but enjoys depth efficiency to a lesser extent than product pooling with linear activation (convolutional arithmetic circuits). These results indicate that from the point of view of expressive power and depth efficiency, convolutional arithmetic circuits (SimNets) have an advantage over the prevalent convolutional rectifier networks. This leads us to believe that developing effective methods for training convolutional arithmetic circuits, thereby fulfilling their expressive potential, may give rise to a deep learning architecture that is provably superior to convolutional rectifier networks but has so far been overlooked by practitioners. [1]

2. Related Work

The literature on the computational complexity of arithmetic circuits is far too wide to cover here, dating back over five decades. Although many of the fundamental questions in the field remain open, significant progress has been made over the years, developing and employing a vast share of mathematical tools from branches of geometry, algebra, analysis, combinatorics, and more. We refer the interested reader to (Shpilka and Yehudayoff, 2010) for a survey written in 2010, and mention here the more recent works (Delalleau and Bengio, 2011) and (Martens and Medabalimi, 2014) studying depth efficiency of arithmetic circuits in the context of deep learning (Sum-Product Networks). Compared to arithmetic circuits, the literature on depth efficiency of neural networks with ReLU activation is far less developed, primarily since these models were only introduced several years ago (Nair and Hinton, 2010). There have been some notable works on this line, but these employ dedicated mathematical machinery, not making use of the plurality of tools available in the world of arithmetic circuits. (Pascanu et al., 2013) and (Montufar et al., 2014) use combinatorial arguments to characterize the maximal number of linear regions in functions generated by ReLU networks, thereby establishing existence of depth efficiency. (Telgarsky, 2016) uses semi-algebraic geometry to analyze the number of oscillations in functions realized by neural networks with semi-algebraic activations, ReLU in particular. The fundamental result proven in this work is the existence, for every $k \in \mathbb{N}$, of functions realizable by networks with $\Theta(k^3)$ layers and $\Theta(1)$ nodes per layer, which cannot be approximated by networks with $\mathcal{O}(k)$ layers unless these are exponentially large (have $\Omega(2^k)$ nodes). The work of (Eldan and Shamir, 2015) makes use of Fourier analysis to show existence of functions that are efficiently computable by depth-3 networks, yet require exponential size in order to be approximated by depth-2 networks. The result applies to various activations, including ReLU. (Poggio et al., 2015) also compares the computational abilities of deep *vs.* shallow networks under different activations that include ReLU. However, the complexity measure considered in (Poggio et al., 2015) is the VC dimension, whereas our interest lies in network size.

None of the analyses above account for convolutional networks [2], thus they do not apply to the deep learning architecture most commonly used in practice. Recently, (Cohen et al., 2016b) introduced convolutional arithmetic circuits, which may be viewed as ConvNets with linear activation and product pooling. These networks were shown to correspond to hierarchical tensor decompositions (see (Hackbusch, 2012)). Tools from linear algebra, functional analysis and measure theory were then employed to prove that the networks are universal, and exhibit *complete depth efficiency*. Although similar in structure, convolutional arithmetic circuits are inherently different from convolutional rectifier networks (ConvNets with ReLU activation and max or average pooling). Accordingly, the analysis carried out in (Cohen et al., 2016b) does not apply to the networks at the forefront of deep learning.

Closing the gap between the networks analyzed in (Cohen

[1] Due to lack of space, a significant portion of the paper is deferred to the appendices. We refer the reader to (Cohen and Shashua, 2016) for a self-contained version of this manuscript.

[2] By this we mean that in all analyses, the deep networks shown to benefit from depth (*i.e.* to realize functions that require super-polynomial size from shallow networks) are not ConvNets.

et al., 2016b) and convolutional rectifier networks is the topic of this paper. We achieve this by generalizing tensor decompositions, thereby opening the door to mathematical machinery as used in (Cohen et al., 2016b), harnessing it to analyze, for the first time, the depth efficiency of convolutional rectifier networks.

3. Generalized Tensor Decompositions

We begin by establishing basic tensor-related terminology and notations [3]. For our purposes, a *tensor* is simply a multi-dimensional array. The *order* of a tensor is defined to be the number of indexing entries in the array, which are referred to as *modes*. The *dimension* of a tensor in a particular mode is defined as the number of values that may be taken by the index in that mode. For example, a 4-by-3 matrix is a tensor of order 2, *i.e.* it has two modes, with dimension 4 in mode 1 and dimension 3 in mode 2. If $\mathcal{A}$ is a tensor of order N and dimension M_i in each mode $i \in [N] := \{1, \ldots, N\}$, the space of all configurations it can take is denoted, quite naturally, by $\mathbb{R}^{M_1 \times \cdots \times M_N}$.

The fundamental operator in tensor analysis is the *tensor product*, denoted by $\otimes$. It is an operator that intakes two tensors $\mathcal{A} \in \mathbb{R}^{M_1 \times \cdots \times M_P}$ and $\mathcal{B} \in \mathbb{R}^{M_{P+1} \times \cdots \times M_{P+Q}}$ (orders P and Q respectively), and returns a tensor $\mathcal{A} \otimes \mathcal{B} \in \mathbb{R}^{M_1 \times \cdots \times M_{P+Q}}$ (order $P+Q$) defined by:

$$(\mathcal{A} \otimes \mathcal{B})_{d_1,\ldots,d_{P+Q}} = \mathcal{A}_{d_1,\ldots,d_P} \cdot \mathcal{B}_{d_{P+1},\ldots,d_{P+Q}} \tag{1}$$

Tensor decompositions (see (Kolda and Bader, 2009) for a survey) may be viewed as schemes for expressing tensors using tensor products and weighted sums. For example, suppose we have a tensor $\mathcal{A} \in \mathbb{R}^{M_1 \times \cdots \times M_N}$ given by:

$$\mathcal{A} = \sum\nolimits_{j_1 \ldots j_N = 1}^{J} c_{j_1 \ldots j_N} \cdot \mathbf{a}^{j_1,1} \otimes \cdots \otimes \mathbf{a}^{j_N,N}$$

This expression is known as a Tucker decomposition, parameterized by the coefficients $\{c_{j_1 \ldots j_N} \in \mathbb{R}\}_{j_1 \ldots j_N \in [J]}$ and vectors $\{\mathbf{a}^{j,i} \in \mathbb{R}^{M_i}\}_{i \in [N], j \in [J]}$. It is different from the *CP* and *Hierarchical Tucker* decompositions our analysis will rely upon (see sec. 4), yet all decompositions are closely related, specifically in the fact that they are based on iterating between tensor products and weighted sums.

Our construction and analysis are facilitated by generalizing the tensor product, which in turn generalizes tensor decompositions. For an associative and commutative binary operator g, *i.e.* a function $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ such that $\forall a, b, c \in \mathbb{R} : g(g(a,b),c) = g(a,g(b,c))$ and $\forall a, b \in \mathbb{R} : g(a,b) = g(b,a)$, the *generalized tensor product* $\otimes_g$, an operator intaking tensors $\mathcal{A} \in \mathbb{R}^{M_1 \times \cdots \times M_P}$, $\mathcal{B} \in \mathbb{R}^{M_{P+1} \times \cdots \times M_{P+Q}}$ and returning tensor $\mathcal{A} \otimes_g \mathcal{B} \in \mathbb{R}^{M_1 \times \cdots \times M_{P+Q}}$, is defined as follows:

$$(\mathcal{A} \otimes_g \mathcal{B})_{d_1,\ldots,d_{P+Q}} = g(\mathcal{A}_{d_1,\ldots,d_P}, \mathcal{B}_{d_{P+1},\ldots,d_{P+Q}}) \tag{2}$$

Generalized tensor decompositions are simply obtained by plugging in the generalized tensor product $\otimes_g$ in place of the standard tensor product $\otimes$.

4. From Networks to Tensors

The ConvNet architecture analyzed in this paper is presented in fig. 1. The input to a network, denoted X, is composed of N *patches* $\mathbf{x}_1 \ldots \mathbf{x}_N \in \mathbb{R}^s$. For example, X could represent a 32-by-32 RGB image through 5×5 regions crossing the three color bands, in which case, assuming a patch is taken for every pixel (boundaries padded), we have $N = 1024$ and $s = 75$. The first layer in a network is referred to as *representation*, and may be thought of as a generalized convolution. Namely, it consists of applying M *representation functions* $f_{\theta_1} \ldots f_{\theta_M} : \mathbb{R}^s \to \mathbb{R}$ to all patches of the input, thereby creating M feature maps. In the case where representation functions are standard neurons, *i.e.* $f_{\theta_d}(\mathbf{x}) = \sigma(\mathbf{w}_d^\top \mathbf{x} + b_d)$ for parameters $\theta_d = (\mathbf{w}_d, b_d) \in \mathbb{R}^s \times \mathbb{R}$ and some chosen activation $\sigma(\cdot)$, we obtain a conventional convolutional layer.

Following the representation, a network includes L hidden layers indexed by $l = 0 \ldots L-1$. Each hidden layer l begins with a 1×1 *conv* operator, which is simply a 3D convolution with r_l channels and receptive field 1×1 followed by point-wise activation $\sigma(\cdot)$. We allow the convolution to operate without weight sharing, in which case the filters that generate feature maps by sliding across the previous layer may have different coefficients at different spatial locations. This is often referred to in the deep learning community as a locally-connected layer (see (Taigman et al., 2014)). We refer to it as the *unshared* case, in contrast to the *shared* case that gives rise to a standard 1×1 convolution. The second (last) operator in a hidden layer is spatial pooling. Feature maps generated by 1×1 conv are decimated, by applying the pooling operator $P(\cdot)$ (*e.g.* max or average) to non-overlapping 2D windows that cover the spatial extent. The last of the L hidden layers ($l = L-1$) reduces feature maps to singletons (global pooling), creating a vector of dimension r_{L-1}. This vector is mapped into Y network outputs through a final dense linear layer.

Altogether, the architectural parameters of a ConvNet are the type of representation functions (f_{θ_d}), the pooling window sizes (which in turn determine the number of hidden layers L), the setting of conv weights as shared or unshared, the number of channels in each layer (M for representation, $r_0 \ldots r_{L-1}$ for hidden layers, Y for output), and the choice of activation and pooling operators ($\sigma(\cdot)$ and $P(\cdot)$ respec-

[3] The definitions we give are actually concrete special cases of more abstract algebraic definitions as given in (Hackbusch, 2012). We limit the discussion to these special cases since they suffice for our needs and are easier to grasp.

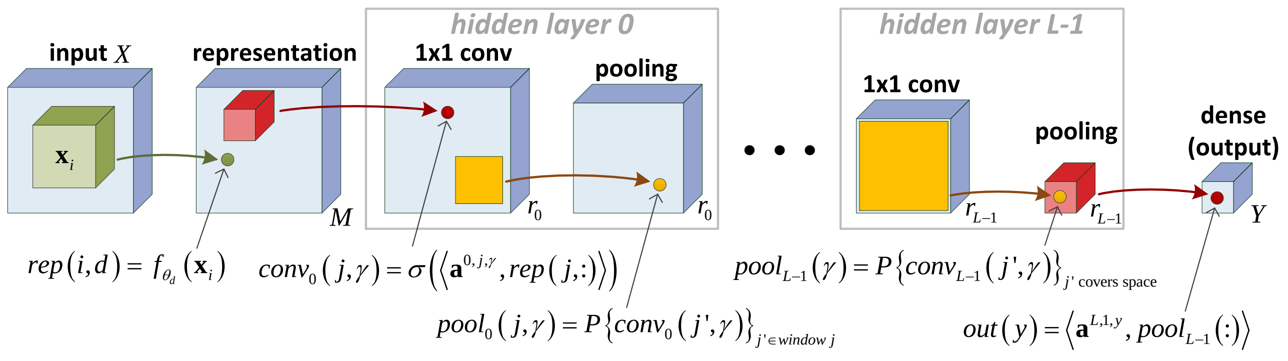
hidden layer 0
hidden layer L-1
input X
representation
1x1 conv
pooling
$\mathbf{x}_i$
M
r_0
r_0
1x1 conv
pooling
dense (output)
r_{L-1}
r_{L-1}
Y
$rep(i,d) = f_{\theta_d}(\mathbf{x}_i)$
$conv_0(j,\gamma) = \sigma(\langle \mathbf{a}^{0,j,\gamma}, rep(j,:)\rangle)$
$pool_0(j,\gamma) = P\{conv_0(j',\gamma)\}_{j' \in window\ j}$
$pool_{L-1}(\gamma) = P\{conv_{L-1}(j',\gamma)\}_{j' \text{ covers space}}$
$out(y) = \langle \mathbf{a}^{L,1,y}, pool_{L-1}(:)\rangle$

Figure 1. ConvNet architecture analyzed in this paper (see description in sec. 4). Best viewed in color.

tively). Given these architectural parameters, the learnable parameters of a network are the representation weights (θ_d), the conv weights ($\mathbf{a}^{l,j,\gamma}$ for hidden layer l, location j and channel γ in unshared case; $\mathbf{a}^{l,\gamma}$ for hidden layer l and channel γ in shared case), and the output weights ($\mathbf{a}^{L,1,y}$).

The choice of activation and pooling operators determines the type of network we arrive at. For linear activation ($\sigma(z) = z$) and product pooling ($P\{c_j\} = \prod c_j$) we get a convolutional arithmetic circuit as analyzed in (Cohen et al., 2016b). For ReLU activation ($\sigma(z) = \max\{0, z\}$) and max or average pooling ($P\{c_j\} = \max\{c_j\}$ or $P\{c_j\} = \text{mean}\{c_j\}$) we get the commonly used convolutional rectifier networks, on which we focus in this paper.

In terms of pooling window sizes and network depth, we direct our attention to two special cases representing the extremes. The first is a shallow network that includes global pooling in its single hidden layer – see illustration in fig. 2. The second is the deepest possible network, in which all pooling windows cover only two entries, resulting in $L = \log_2 N$ hidden layers. These ConvNets, which we refer to as *shallow* and *deep* respectively, will be shown to correspond to canonical tensor decompositions. It is for this reason, and for simplicity of presentation, that we focus on these special cases. One may just as well consider networks of intermediate depths with different pooling window sizes, and that would correspond to other, non-standard, tensor decompositions. The analysis carried out in sec. 5 can easily be adapted to such networks.

In a classification setting, the Y outputs of a network correspond to different categories, and prediction follows the output with highest activation. Specifically, if we denote by $h_y(\cdot)$ the mapping from network input to output y, the predicted label for the instance $X = (\mathbf{x}_1, \ldots, \mathbf{x}_N) \in (\mathbb{R}^s)^N$ is determined by the following classification rule:

$$\hat{y} = \text{argmax}_{y \in [Y]}\, h_y(X)$$

We refer to h_y as the *score function* of category y. Score functions are studied in this paper through the notion of *grid tensors*. Given fixed vectors $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(M)} \in \mathbb{R}^s$, referred to as *templates*, the grid tensor of h_y, denoted $\mathcal{A}(h_y)$, is defined to be the tensor of order N and dimension M in each mode given by:

$$\mathcal{A}(h_y)_{d_1 \ldots d_N} = h_y(\mathbf{x}^{(d_1)}, \ldots, \mathbf{x}^{(d_N)}) \tag{3}$$

That is to say, the grid tensor of a score function under M templates $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(M)}$, is a tensor of order N and dimension M in each mode, holding score values on the exponentially large grid of instances $\{X_{d_1 \ldots d_N} := (\mathbf{x}^{(d_1)}, \ldots, \mathbf{x}^{(d_N)}) : d_1 \ldots d_N \in [M]\}$. Before heading on to our analysis of grid tensors generated by ConvNets, to simplify notation, we define $F \in \mathbb{R}^{M \times M}$ to be the matrix holding the values taken by the representation functions $f_{\theta_1} \ldots f_{\theta_M} : \mathbb{R}^s \to \mathbb{R}$ on the selected templates $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(M)} \in \mathbb{R}^s$:

$$F := \begin{bmatrix} f_{\theta_1}(\mathbf{x}^{(1)}) & \cdots & f_{\theta_M}(\mathbf{x}^{(1)}) \\ \vdots & \ddots & \vdots \\ f_{\theta_1}(\mathbf{x}^{(M)}) & \cdots & f_{\theta_M}(\mathbf{x}^{(M)}) \end{bmatrix} \tag{4}$$

To express the grid tensor of a ConvNet's score function using generalized tensor decompositions (see sec. 3), we set the underlying function $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ to be the *activation-pooling* operator defined by:

$$g(a, b) = P(\sigma(a), \sigma(b)) \tag{5}$$

where $\sigma(\cdot)$ and $P(\cdot)$ are the network's activation and pooling functions, respectively. Notice that the activation-pooling operator meets the associativity and commutativity requirements under product pooling with linear activation ($g(a, b) = a \cdot b$), and under max pooling with ReLU activation ($g(a, b) = \max\{a, b, 0\}$). To account for the case of average pooling with ReLU activation, which a-priori leads to a non-associative activation-pooling operator, we simply replace average by sum, *i.e.* we analyze sum pooling with ReLU activation ($g(a, b) = \max\{a, 0\} + \max\{b, 0\}$), which from the point of view of expressiveness is completely equivalent to average pooling with ReLU activation (linear weights that follow pooling absorb scaling factors).

With the activation-pooling operator g in place, it is straightforward to see that the grid tensor of h_y^S – a score function generated by the shallow ConvNet (fig. 2), is given by the following generalized tensor decomposition:

$$\mathcal{A}\left(h_y^S\right) = \sum_{z=1}^{Z} a_z^y \cdot \left(F\mathbf{a}^{z,1}\right) \otimes_g \cdots \otimes_g \left(F\mathbf{a}^{z,N}\right) \qquad (6)$$

Z here is the number of channels in the network's hidden layer, $\{\mathbf{a}^{z,i} \in \mathbb{R}^M\}_{z\in[Z],i\in[N]}$ are the weights in the hidden conv, and $\mathbf{a}^y \in \mathbb{R}^Z$ are the weights of output y. The factorization in eq. 6 generalizes the classic CP decomposition (see (Kolda and Bader, 2009)), and we accordingly refer to it as the *generalized CP decomposition*.

Turning to the deep ConvNet (fig. 1 with size-2 pooling windows and $L = \log_2 N$ hidden layers), the grid tensor of its score function h_y^D is given by the hierarchical generalized tensor decomposition below:

$$\begin{aligned}
\phi^{1,j,\gamma} &= \sum_{\alpha=1}^{r_0} a_\alpha^{1,j,\gamma} (F\mathbf{a}^{0,2j-1,\alpha}) \otimes_g (F\mathbf{a}^{0,2j,\alpha}) \\
&\cdots \\
\phi^{l,j,\gamma} &= \sum_{\alpha=1}^{r_{l-1}} a_\alpha^{l,j,\gamma} \underbrace{\phi^{l-1,2j-1,\alpha}}_{\text{order } 2^{l-1}} \otimes_g \underbrace{\phi^{l-1,2j,\alpha}}_{\text{order } 2^{l-1}} \\
&\cdots \\
\phi^{L-1,j,\gamma} &= \sum_{\alpha=1}^{r_{L-2}} a_\alpha^{L-1,j,\gamma} \underbrace{\phi^{L-2,2j-1,\alpha}}_{\text{order } \frac{N}{4}} \otimes_g \underbrace{\phi^{L-2,2j,\alpha}}_{\text{order } \frac{N}{4}} \\
\mathcal{A}\left(h_y^D\right) &= \sum_{\alpha=1}^{r_{L-1}} a_\alpha^{L,1,y} \underbrace{\phi^{L-1,1,\alpha}}_{\text{order } \frac{N}{2}} \otimes_g \underbrace{\phi^{L-1,2,\alpha}}_{\text{order } \frac{N}{2}} \qquad (7)
\end{aligned}$$

$r_0 \ldots r_{L-1} \in \mathbb{N}$ here are the number of channels in the network's hidden layers, $\{\mathbf{a}^{0,j,\gamma} \in \mathbb{R}^M\}_{j\in[N],\gamma\in[r_0]}$ are the weights in the first hidden conv, $\{\mathbf{a}^{l,j,\gamma} \in \mathbb{R}^{r_{l-1}}\}_{l\in[L-1],j\in[N/2^l],\gamma\in[r_l]}$ are the weights in the following hidden convs, and $\mathbf{a}^{L,1,y} \in \mathbb{R}^{r_{L-1}}$ are the weights of output y. The factorization in eq. 7 generalizes the Hierarchical Tucker decomposition introduced in (Hackbusch and Kühn, 2009), and is accordingly referred to as the *generalized HT decomposition*.

To conclude this section, we presented a ConvNet architecture (fig. 1) whose activation and pooling operators may be chosen to realize convolutional arithmetic circuits (linear activation, product pooling) or convolutional rectifier networks (ReLU activation, max/average pooling). We then defined the grid tensor of a network's score function as a tensor holding function values on an exponentially large

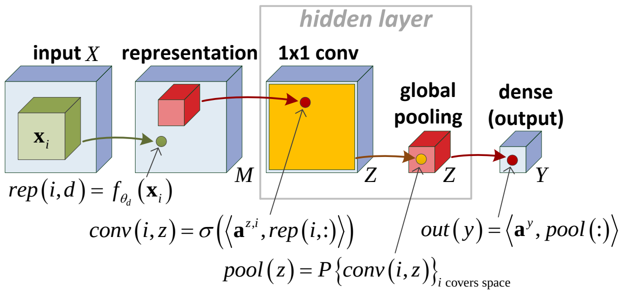
hidden layer
input X
representation
1x1 conv
global pooling
dense (output)
$\mathbf{x}_i$
M
Z
Z
Y
$rep(i,d) = f_{\theta_d}(\mathbf{x}_i)$
$conv(i,z) = \sigma\left(\left\langle \mathbf{a}^{z,i}, rep(i,:)\right\rangle\right)$
$out(y) = \left\langle \mathbf{a}^y, pool(:)\right\rangle$
$pool(z) = P\{conv(i,z)\}_{i \text{ covers space}}$

Figure 2. Shallow ConvNet with global pooling in its single hidden layer. Best viewed in color.

grid whose points are sequences with elements chosen from a finite set of templates. Then, we saw that the grid tensor of a shallow ConvNet (fig. 2) is given by the generalized CP decomposition (eq. 6), and a grid tensor of a deep ConvNet (fig. 1 with $L = \log_2 N$) is given by the generalized HT decomposition (eq. 7). In the next section we utilize the connection between ConvNets and generalized tensor decompositions for an analysis of the expressive power and depth efficiency of convolutional rectifier networks.

5. Capacity Analysis

In this section we study score functions expressible by the shallow and deep ConvNets (fig. 2, and fig. 1 with $L = \log_2 N$, respectively) under ReLU activation with max or average pooling (convolutional rectifier networks), comparing these settings against linear activation with product pooling (convolutional arithmetic circuits). Due to lack of space, ***the proofs of our claims are all given in app. E***. In these proofs, score functions are analyzed through grid tensors (eq. 3), represented by the generalized tensor decompositions established in the previous section: the generalized CP decomposition (eq. 6) corresponding to the shallow network, and the generalized HT decomposition (eq. 7) corresponding to the deep network.

5.1. Templates and Representation Functions

The expressiveness of our ConvNets obviously depends on the possible forms that may be taken by the representation functions $f_{\theta_1} \ldots f_{\theta_M} : \mathbb{R}^s \to \mathbb{R}$. For example, if representation functions are limited to be constant, the ConvNets can only realize constant score functions. We denote by $\mathcal{F} := \{f_\theta : \mathbb{R}^s \to \mathbb{R} : \theta \in \Theta\}$ the parametric family from which representation functions are chosen, and make two mild assumptions on this family:

- **Continuity**: $f_\theta(\mathbf{x})$ is continuous w.r.t. both θ and $\mathbf{x}$.
- **Non-degeneracy**: For any $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(M)} \in \mathbb{R}^s$ such that $\mathbf{x}_i \neq \mathbf{x}_j\ \forall i{\neq}j$, there exist $f_{\theta_1} \ldots f_{\theta_M} \in \mathcal{F}$ for which the matrix F defined in eq. 4 is non-singular.

Both of the assumptions above are met for most reasonable choices of $\mathcal{F}$. In particular, non-degeneracy holds when representation functions are standard neurons:

Claim 1. *The parametric family:*

$$\mathcal{F} = \left\{ f_\theta(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b) : \theta = (\mathbf{w}, b) \in \mathbb{R}^s \times \mathbb{R} \right\}$$

where $\sigma(\cdot)$ is any sigmoidal activation [4] *or the ReLU activation, meets the non-degeneracy condition (*i.e. *for any distinct $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(M)} \in \mathbb{R}^s$ there exist $f_{\theta_1} \ldots f_{\theta_M} \in \mathcal{F}$ such that the matrix F defined in eq. 4 is non-singular).*

Non-degeneracy means that given distinct templates, one may choose representation functions for which F is non-singular. We may as well consider the opposite situation, where we are given representation functions, and would like to choose templates leading to a non-singular F. Apparently, so long as the representation functions are linearly independent, this is always possible:

Claim 2. *Let $f_{\theta_1} \ldots f_{\theta_M} : \mathbb{R}^s \to \mathbb{R}$ be any linearly independent continuous functions. Then, there exist $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(M)} \in \mathbb{R}^s$ such that F (eq. 4) is non-singular.*

As stated previously, in this paper we study score functions expressible by ConvNets through the notion of grid tensors. The translation of score functions into grid tensors is facilitated by the choice of templates $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(M)} \in \mathbb{R}^s$ (eq. 3). For general templates, the correspondence between score functions and grid tensors is not injective – a score function corresponds to a single grid tensor, but a grid tensor may correspond to more than one score function. We use the term *covering* to refer to templates leading to an injective correspondence, *i.e.* to a situation where two score functions associated with the same grid tensor are effectively identical. In other words, the templates $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(M)}$ are covering if the value of score functions outside the exponentially large grid $\left\{ X_{d_1 \ldots d_N} := (\mathbf{x}^{(d_1)}, \ldots, \mathbf{x}^{(d_N)}) : d_1 \ldots d_N \in [M] \right\}$ is irrelevant for classification. Some of the claims in our analysis will assume existence of covering templates (it will be stated explicitly when so). We argue in app. A that for structured compositional data (*e.g.*natural images), $M \in \Omega(100)$ suffices in order for this assumption to hold.

5.2. Universality

Universality refers to the ability of a network to realize (or approximate) any function of choice when its size is unlimited. It is well-known that fully-connected neural networks are universal under all types of non-linear activations typically used in practice, even if the number of hidden layers is restricted to one (Cybenko, 1989; Hornik et al., 1989).

[4] $\sigma(\cdot)$ is sigmoidal if it is monotonic with $\lim_{z \to -\infty} \sigma(z) = c$ and $\lim_{z \to +\infty} \sigma(z) = C$ for some $c \neq C$ in $\mathbb{R}$.

To the best of our knowledge universality has never been studied in the context of convolutional rectifier networks. This is the purpose of the current subsection. Specifically, we analyze the universality of our shallow and deep ConvNets (fig. 2, and fig. 1 with $L = \log_2 N$, respectively) under ReLU activation and max or average pooling.

We begin by stating a result similar to that given in (Cohen et al., 2016b), according to which convolutional arithmetic circuits are universal:

Claim 3. *Assuming covering templates exist, with linear activation and product pooling the shallow ConvNet is universal (hence so is the deep).*

Heading on to convolutional rectifier networks, the following claim tells us that max pooling leads to universality:

Claim 4. *Assuming covering templates exist, with ReLU activation and max pooling the shallow ConvNet is universal (hence so is the deep).*

At this point we encounter the first somewhat surprising result, according to which convolutional rectifier networks are not universal with average pooling:

Claim 5. *With ReLU activation and average pooling, both the shallow and deep ConvNets are not universal.*

One may wonder if perhaps the non-universality of ReLU activation and average pooling is merely an artifact of the conv operator in our networks having 1×1 receptive field. Apparently, as the following claim shows, expanding the receptive field does not remedy the situation, and indeed non-universality is an inherent property of convolutional rectifier networks with average pooling:

Claim 6. *Consider the network obtained by expanding the conv receptive field in the shallow ConvNet from 1×1 to $w \times h$, where $w \cdot h < N/2 + 1 - \log_M N$ (conv windows cover less than half the feature maps that precede them). Such a network, when equipped with ReLU activation and average pooling, is not universal.*

We conclude this subsection by noting that the non-universality result in claim 6 does *not* contradict the known universality of shallow (single hidden layer) fully-connected neural networks – see app. B for a discussion on the matter.

5.3. Depth Efficiency

The driving force behind deep learning is the expressive power that comes with depth. It is generally believed that deep networks with non-linear layers efficiently express functions that cannot be efficiently expressed by shallow networks, *i.e.* that would require the latter to have super-polynomial size. We refer to such scenario as *depth efficiency*. Being concerned with the minimal size required by a shallow network in order to realize (or approximate) a given function, the question of depth efficiency implicitly assumes universality, *i.e.* that there exists some (possibly exponential) size with which the shallow network is capable of expressing the target function [5].

To the best of our knowledge, at the time of this writing the only work to formally analyze depth efficiency in the context of ConvNets is (Cohen et al., 2016b). This work focused on convolutional arithmetic circuits, showing that with such networks depth efficiency is *complete*, *i.e.* besides a negligible set, all functions realizable by a deep network are depth efficient. Framing this result in our setup:

Claim 7 (adaptation of theorem 1 in (Cohen et al., 2016b))**.** *Let $f_{\theta_1} \ldots f_{\theta_M}$ be any set of linearly independent representation functions for a deep ConvNet (fig. 1 with $L = \log_2 N$) with linear activation and product pooling. Suppose we randomize the linear weights ($\mathbf{a}^{l,j,\gamma}$) of the network by some continuous distribution. Then, with probability 1, we obtain score functions that cannot be realized by a shallow ConvNet (fig. 2) with linear activation and product pooling if the number of hidden channels in the latter (Z) is less than $\min\{r_0, M\}^{N/2}$.*

We now turn to convolutional rectifier networks, for which depth efficiency has yet to be analyzed. In sec. 5.2 we saw that convolutional rectifier networks are universal with max pooling, and non-universal with average pooling. Since depth efficiency is only applicable to universal architectures, we focus on the former setting. The following claim establishes existence of depth efficiency for ConvNets with ReLU activation and max pooling:

Claim 8. *There exist weight settings for a deep ConvNet with ReLU activation and max pooling, giving rise to score functions that cannot be realized by a shallow ConvNet with ReLU activation and max pooling if the number of hidden channels in the latter (Z) is less than $\min\{r_0, M\}^{N/2} \cdot \frac{2}{M \cdot N}$.*

Nearly all results in the literature that relate to depth efficiency merely show its existence, and claim 8 is no different in that respect. From a practical perspective, the implications of such results are slight, as a-priori, it may be that only a small fraction of the functions realizable by a deep network enjoy depth efficiency, and for all the rest shallow networks suffice. In app. F we extend claim 8, arguing that with ReLU activation and max pooling, depth efficiency becomes more and more prevalent as the number of hidden channels in the deep ConvNet grows. However, no matter how large the deep ConvNet is, with ReLU activation and max pooling depth efficiency is *never complete* – there is always positive measure to the set of weight configurations that lead the deep ConvNet to generate score functions efficiently realizable by the shallow ConvNet:

Claim 9. *Suppose we randomize the weights of a deep ConvNet with ReLU activation and max pooling by some continuous distribution with non-vanishing continuous probability density function. Then, assuming covering templates exist, with positive probability, we obtain score functions that can be realized by a shallow ConvNet with ReLU activation and max pooling having only a single hidden channel ($Z = 1$).*

Comparing claims 7 and 9, we see that depth efficiency is complete under linear activation with product pooling, and incomplete under ReLU activation with max pooling. We interpret this as indicating that ***convolutional arithmetic circuits benefit from the expressive power of depth more than convolutional rectifier networks do***. This result is rather surprising, particularly since convolutional rectifier networks are much more commonly used in practice. We attribute the discrepancy primarily to historical reasons, and conjecture that developing effective methods for training convolutional arithmetic circuits, thereby fulfilling their expressive potential, may give rise to a deep learning architecture that is provably superior to convolutional rectifier networks but has so far been overlooked by practitioners.

Loosely speaking, we have shown that the gap in expressive power between the shallow and deep ConvNets is greater with linear activation and product pooling than it is with ReLU activation and max pooling. One may wonder at this point if it is plausible to deduce from this which architectural setting is more expressive, as a-priori, altering the shallow *vs.* deep ConvNet comparisons such that one network has linear activation with product pooling and the other has ReLU activation with max pooling, may change the expressive gaps in favor of the latter. Claims 10 and 11 below show that this is not the case. Specifically, they show that the depth efficiency of the deep ConvNet with linear activation and product pooling remains complete when the shallow ConvNet has ReLU activation and max pooling (claim 10), and on the other hand, the depth efficiency of the deep ConvNet with ReLU activation and max pooling remains incomplete when the shallow ConvNet has linear activation and product pooling (claim 11). This affirms our stand regarding the expressive advantage of convolutional arithmetic circuits over convolutional rectifier networks.

Claim 10. *Let $f_{\theta_1} \ldots f_{\theta_M}$ be any set of linearly independent representation functions for a deep ConvNet with linear activation and product pooling. Suppose we randomize the weights of the network by some continuous distribution. Then, with probability 1, we obtain score functions*

[5] While technically it is possible to consider depth efficiency with a non-universal shallow network, in the majority of the cases, particularly in our framework, the shallow network would simply not be able to express a function generated by a deep network, no matter how large we allow it to be. Arguably, this provides little insight into the complexity of functions brought forth by depth.

that cannot be realized by a shallow ConvNet with ReLU activation and max pooling if the number of hidden channels in the latter (Z) is less than $\min\{r_0, M\}^{N/2} \cdot \frac{2}{M \cdot N}$.

Claim 11. *Suppose we randomize the weights of a deep ConvNet with ReLU activation and max pooling by some continuous distribution with non-vanishing continuous probability density function. Then, assuming covering templates exist, with positive probability, we obtain score functions that can be realized by a shallow ConvNet with linear activation and product pooling having only a single hidden channel ($Z = 1$).*

5.3.1. APPROXIMATION

In their current form, the results in our analysis establishing depth efficiency (claims 7, 8, 10 and the analogous ones in app. D) relate to exact realization. Specifically, they provide a lower bound on the size of a shallow ConvNet required in order for it to *realize exactly* a grid tensor generated by a deep ConvNet. From a practical perspective, a more interesting question would be the size required by a shallow ConvNet in order to *approximate* the computation of a deep ConvNet. A-priori, it may be that although the size required for exact realization is exponential, the one required for approximation is only polynomial. As we show in app. C, this is not the case, and in fact all of the lower bounds we have provided apply not only to exact realization, but also to arbitrarily-well approximation.

5.4. Shared Coefficients for Convolution

To this end, our analysis has focused on the unshared setting, where the coefficients of the 1×1 conv filters may vary across spatial locations. In practice, ConvNets typically enforce coefficient sharing, which in our framework implies that in channel γ of hidden layer l, instead of having a coefficient set $\mathbf{a}^{l,j,\gamma}$ for every location j, we have a single set $\mathbf{a}^{l,\gamma}$ for all locations. Due to lack of space, we defer our analysis of the shared setting to app. D, providing here a summary of its results.

In terms of universality, introducing sharing is detrimental. Under all possible choices of activation and pooling, shared coefficients limit the shallow ConvNet to symmetric grid tensors only. The deep ConvNet can go beyond symmetric grid tensors, but is still not universal. Evaluation of depth efficiency requires the shallow network to be universal, and we accordingly compare deep ConvNets with shared coefficients against shallow ConvNets that are not constrained by coefficient sharing. In a nutshell, we find all results established above for the unshared setting still applicable – a deep ConvNet with shared coefficients is completely depth efficient under linear activation with product pooling, and incompletely depth efficient under ReLU activation with max pooling.

6. Discussion

The contribution of this paper is twofold. First, we introduce a construction in the form of *generalized tensor decompositions*, that enables transforming convolutional arithmetic circuits into *convolutional rectifier networks* (ConvNets with ReLU activation and max or average pooling). This opens the door to various mathematical tools from the world of arithmetic circuits, now available for analyzing convolutional rectifier networks. As a second contribution, we make use of such tools to prove new results on the expressive properties that drive this important class of networks.

Our analysis shows that convolutional rectifier networks are universal with max pooling, but not with average pooling. This implies that if non-linearity originates solely from ReLU activation, increasing network size alone is not sufficient for expressing arbitrary functions. More interestingly, we analyze the behavior of convolutional rectifier networks in terms of *depth efficiency*, *i.e.* of cases where a function generated by a deep network of polynomial size requires shallow networks to have super-polynomial size. It is known that convolutional arithmetic circuits exhibit *complete depth efficiency*, meaning that besides a negligible (zero measure) set, all functions generated by deep networks of this type are depth efficient. We show that this is not the case with convolutional rectifier networks, for which depth efficiency exists, but is weaker in the sense that it is not complete (there is positive measure to the set of functions generated by a deep network that may be efficiently realized by shallow networks).

Depth efficiency is believed to be the key factor behind the success of deep learning. Our analysis indicates that from this perspective, the widely used convolutional rectifier networks are inferior to convolutional arithmetic circuits. This leads us to believe that convolutional arithmetic circuits bear the potential to improve the performance of deep learning beyond what is witnessed today. Of course, a practical machine learning model is measured not only by its expressive power, but also by our ability to train it. Over the years, massive amounts of research have been devoted to training convolutional rectifier networks. Convolutional arithmetic circuits on the other hand received far less attention, although they have been successfully trained in recent works on the SimNet architecture (Cohen and Shashua, 2014; Cohen et al., 2016a), demonstrating how the enhanced expressive power can lead to state of the art performance in computationally limited settings.

We believe that developing effective methods for training convolutional arithmetic circuits, thereby fulfilling their expressive potential, may give rise to a deep learning architecture that is provably superior to convolutional rectifier networks but has so far been overlooked by practitioners.

Acknowledgments

This work is partly funded by Intel grant ICRI-CI no. 9-2012-6133 and by ISF Center grant 1790/12. Nadav Cohen is supported by a Google Fellowship in Machine Learning.

References

Richard Caron and Tim Traynor. The zero set of a polynomial. *WSMR Report 05-02*, 2005.

Nadav Cohen and Amnon Shashua. Simnets: A generalization of convolutional networks. *Advances in Neural Information Processing Systems (NIPS), Deep Learning Workshop*, 2014.

Nadav Cohen and Amnon Shashua. Convolutional rectifier networks as generalized tensor decompositions. *arXiv preprint arXiv:1603.00162*, 2016.

Nadav Cohen, Or Sharir, and Amnon Shashua. Deep simnets. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016a.

Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. *Conference On Learning Theory (COLT)*, 2016b.

G Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4): 303–314, 1989.

Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674, 2011.

Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. *arXiv preprint arXiv:1512.03965*, 2015.

W Hackbusch and S Kühn. A New Scheme for the Tensor Representation. *Journal of Fourier Analysis and Applications*, 15 (5):706–722, 2009.

Wolfgang Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*, volume 42 of *Springer Series in Computational Mathematics*. Springer Science & Business Media, Berlin, Heidelberg, February 2012.

Kurt Hornik, Maxwell B Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Frank Jones. *Lebesgue integration on Euclidean space*. Jones & Bartlett Learning, 2001.

Tamara G Kolda and Brett W Bader. Tensor Decompositions and Applications. *SIAM Review ()*, 51(3):455–500, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, pages 1106–1114, 2012.

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10), 1995.

James Martens and Venkatesh Medabalimi. On the expressive efficiency of sum product networks. *arXiv preprint arXiv:1411.7717*, 2014.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2924–2932, 2014.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of inference regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv*, 1312, 2013.

Tomaso Poggio, Fabio Anselmi, and Lorenzo Rosasco. I-theory on depth vs width: hierarchical function composition. Technical report, Center for Brains, Minds and Machines (CBMM), 2015.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 689–690. IEEE, 2011.

Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 373–374. International World Wide Web Conferences Steering Committee, 2014.

Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3–4):207–388, 2010.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *CVPR '14: Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, June 2014.

Matus Telgarsky. Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485*, 2016.

Izhar Wallach, Michael Dzamba, and Abraham Heifets. Atomnet: A deep convolutional neural network for bioactivity prediction in structure-based drug discovery. *arXiv preprint arXiv:1510.02855*, 2015.

Daniel Zoran and Yair Weiss. "Natural Images, Gaussian Mixtures and Dead Leaves". *Advances in Neural Information Processing Systems*, pages 1745–1753, 2012.