On the Number of Linear Regions of Deep Neural Networks

by

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio

Preprint no.: 73 2014

On the Number of Linear Regions of Deep Neural Networks

Guido Montúfar
Max Planck Institute for Mathematics in the Sciences
montufar@mis.mpg.de

Razvan Pascanu
Université de Montréal
pascanur@iro.umontreal.ca

Kyunghyun Cho
Université de Montréal
kyunghyun.cho@umontreal.ca

Yoshua Bengio
Université de Montréal, CIFAR Fellow
yoshua.bengio@umontreal.ca

Abstract

We study the complexity of functions computable by deep feedforward neural networks with piecewise linear activations in terms of the symmetries and the number of linear regions that they have. Deep networks are able to sequentially map portions of each layer's input-space to the same output. In this way, deep models compute functions that react equally to complicated patterns of different inputs. The compositional structure of these functions enables them to re-use pieces of computation exponentially often in terms of the network's depth. This paper investigates the complexity of such compositional maps and contributes new theoretical results regarding the advantage of depth for neural networks with piecewise linear activation functions. In particular, our analysis is not specific to a single family of models, and as an example, we employ it for rectifier and maxout networks. We improve complexity bounds from pre-existing work and investigate the behavior of units in higher layers.

Keywords: Deep learning, neural network, input space partition, rectifier, maxout

1 Introduction

Artificial neural networks with several hidden layers, called *deep* neural networks, have become popular due to their unprecedented success in a variety of machine learning tasks (see, e.g., Krizhevsky et al. 2012, Ciresan et al. 2012, Goodfellow et al. 2013, Hinton et al. 2012). In view of this empirical evidence, deep neural networks are becoming increasingly favoured over *shallow* networks (i.e., with a single layer of hidden units), and are often implemented with more than five layers. At the time being, however, only a limited amount of publications have investigated deep networks from a theoretical perspective. Recently, Delalleau and Bengio (2011) showed that a shallow network requires exponentially many more sum-product hidden units[1] than a deep sum-product network in order to compute certain families of polynomials. We are interested in extending this kind of analysis to more popular neural networks.

There is a wealth of literature discussing approximation, estimation, and complexity of artificial neural networks (see, e.g., Anthony and Bartlett 1999). A well-known result states that a feedforward neural network with a single, huge, hidden layer is a universal approximator of Borel measurable functions (see Hornik et al. 1989, Cybenko 1989). Other works have investigated universal approximation of probability distributions by deep belief networks (Le Roux and Bengio 2010, Montúfar and Ay 2011), as well as their approximation properties (Montúfar 2014, Krause et al. 2013).

[1] A single sum-product hidden layer summarizes a layer of product units followed by a layer of sum units.

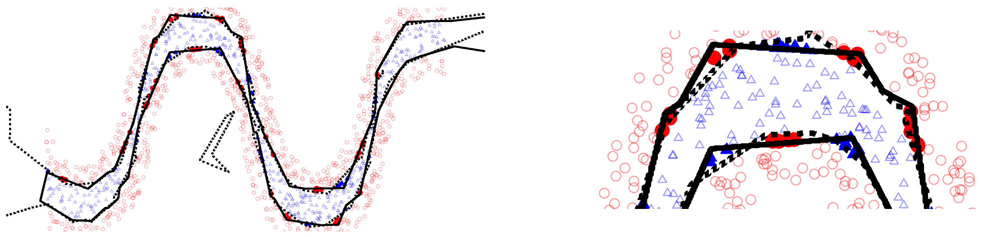


Figure 1: Binary classification using a shallow model with 20 hidden units (solid line) and a deep model with two layers of 10 units each (dashed line). The right panel shows a close-up of the left panel. Filled markers indicate errors made by the shallow model.

These previous theoretical results, however, do not trivially apply to the types of deep neural networks that have seen success in recent years. Conventional neural networks often employ either hidden units with a bounded smooth activation function, or Boolean hidden units. On the other hand, recently it has become more common to use piecewise linear functions, such as the *rectifier* activation $g(a) = \max\{0, a\}$ (Glorot et al. 2011, Nair and Hinton 2010) or the *maxout* activation $g(a_1, \ldots, a_k) = \max\{a_1, \ldots, a_k\}$ (Goodfellow et al. 2013). The practical success of deep neural networks with piecewise linear units calls for the theoretical analysis specific for this type of neural networks.

In this respect, Pascanu et al. (2013) reported a theoretical result on the complexity of functions computable by deep feedforward networks with rectifier units. They showed that, in the asymptotic limit of many hidden layers, deep networks are able to separate their input space into exponentially more linear response regions than their shallow counterparts, despite using the same number of computational units.

Building on the ideas from (Pascanu et al. 2013), we develop a general framework for analyzing deep models with piecewise linear activations. The intermediary layers of these models are able to map several pieces of their inputs into the same output. The layer-wise composition of the functions computed in this way re-uses low-level computations exponentially often as the number of layers increases. This key property enables deep networks to compute highly complex and structured functions. We underpin this idea by estimating the number of linear regions of functions computable by two important types of piecewise linear networks: with rectifier units and with maxout units.

Our results for the complexity of deep rectifier networks yield a significant improvement over the previous results on rectifier networks mentioned above, showing a favourable behavior of deep over shallow networks even with a moderate number of hidden layers. Our analysis of deep rectifier and maxout networks serves as plattform to study a broad variety of related networks, such as convolutional networks.

The number of linear regions of the functions that can be computed by a given model is a measure of the model's flexibility. An example of this is given in Fig. 1, which compares the learnt decision boundary of a single-layer and a two-layer model with the same number of hidden units (see details in Appendix F). This illustrates the advantage of depth; the deep model captures the desired boundary more accurately, approximating it with a larger number of linear pieces.

As noted earlier, deep networks are able to *identify* an exponential number of input neighborhoods by mapping them to a common output of some intermediary hidden layer. The computations carried out on the activations of this intermediary layer are replicated many times, once in each of the identified neighborhoods. This allows the networks to compute very complex looking functions even when they are defined with relatively few parameters.

The number of parameters is an upper bound for the dimension of the set of functions computable by a network, and a small number of parameters means that the class of computable functions has a low dimension. The set of functions computable by a deep feedforward piecewise linear network, although low dimensional, achieves exponential complexity by re-using and composing features from layer to layer.

2 Feedforward Neural Networks and their Compositional Properties

In this section we discuss the ability of deep feedforward networks to re-map their input-space to create complex symmetries by using only relatively few computational units. The key observation of our analysis is that each layer of a deep model is able to map different regions of its input to a common output. This leads to a compositional structure, where computations on higher layers are effectively replicated in all input regions that produced the same output at a given layer. The capacity to replicate computations over the input-space grows exponentially with the number of network layers. Before expanding these ideas, we introduce basic definitions needed in the rest of the paper. At the end of this section, we give an intuitive perspective for reasoning about the replicative capacity of deep models.

2.1 Definitions

A *feedforward neural network* is a composition of layers of computational units which defines a function $F\colon \mathbb{R}^{n_0} \to \mathbb{R}^{\text{out}}$ of the form

$$F(\mathbf{x};\theta) = f_{\text{out}} \circ g_L \circ f_L \circ \cdots \circ g_1 \circ f_1(\mathbf{x}), \tag{1}$$

where f_l is a linear pre-activation function and g_l is a nonlinear activation function. The parameter θ is composed of *input* weight matrices $\mathbf{W}_l \in \mathbb{R}^{k\cdot n_l \times n_{l-1}}$ and *bias* vectors $\mathbf{b}_l \in \mathbb{R}^{k\cdot n_l}$ for each layer $l \in [L]$.

The output of the l-th layer is a vector $\mathbf{x}_l = [\mathbf{x}_{l,1}, \dots, \mathbf{x}_{l,n_l}]^\top$ of activations $\mathbf{x}_{l,i}$ of the units $i \in [n_l]$ in that layer. This is computed from the activations of the preceding layer by $\mathbf{x}_l = g_l(f_l(\mathbf{x}_{l-1}))$. Given the activations $\mathbf{x}_{l-1}$ of the units in the $(l-1)$-th layer, the pre-activation of layer l is given by

$$f_l(\mathbf{x}_{l-1}) = \mathbf{W}_l \mathbf{x}_{l-1} + \mathbf{b}_l,$$

where $f_l = [f_{l,1}, \dots, f_{l,n_l}]^\top$ is an array composed of n_l pre-activation vectors $f_{l,i} \in \mathbb{R}^k$, and the activation of the i-th unit in the l-th layer is given by

$$\mathbf{x}_{l,i} = g_{l,i}(f_{l,i}(\mathbf{x}_{l-1})).$$

We will abbreviate $g_l \circ f_l$ by h_l. When the layer index l is clear, we will drop the corresponding subscript. We are interested in piecewise linear activations, and will consider the following two important types.

- Rectifier unit: $g_i(f_i) = \max\{0, f_i\}$, where $f_i \in \mathbb{R}$ and $k = 1$.
- Rank-k maxout unit: $g_i(f_i) = \max\{f_{i,1}, \dots, f_{i,k}\}$, where $f_i = [f_{i,1}, \dots, f_{i,k}] \in \mathbb{R}^k$.

The *structure* of the network refers to the way its units are arranged. It is specified by the number n_0 of input dimensions, the number of layers L, and the number of units or *width* n_l of each layer.

We will classify the functions computed by different network structures, for different choices of parameters, in terms of their number of linear regions. A *linear region* of a piecewise linear function $F\colon \mathbb{R}^{n_0} \to \mathbb{R}^m$ is a maximal connected subset of the input-space $\mathbb{R}^{n_0}$, on which F is linear. For the functions that we consider, each linear region has full dimension, n_0.

2.2 Shallow Neural Networks

Rectifier units have two types of behavior; they can be either constant 0 or linear, depending on their inputs. The boundary between these two behaviors is given by a hyperplane, and the collection of all the hyperplanes coming from all units in a rectifier layer forms a *hyperplane arrangement*. In general, if the activation function $g\colon \mathbb{R} \to \mathbb{R}$ has a distinguished (i.e., irregular) behavior at zero (e.g., an inflection point or non-linearity), then the function $\mathbb{R}^{n_0} \to \mathbb{R}^{n_1};\ \mathbf{x} \mapsto g(\mathbf{W}\mathbf{x} + \mathbf{b})$ has a distinguished behavior at all inputs from any of the hyperplanes $H_i := \{\mathbf{x} \in \mathbb{R}^{n_0} \colon \mathbf{W}_{i,:}\mathbf{x} + \mathbf{b}_i = 0\}$ for $i \in [n_1]$. The hyperplanes capturing this distinguished behavior also form a hyperplane arrangement.

The hyperplanes in the arrangement split the input-space into several regions. Formally, a *region* of a hyperplane arrangement $\{H_1, \dots, H_{n_1}\}$ is a connected component of the complement $\mathbb{R}^{n_0} \setminus (\cup_i H_i)$, i.e., a set of points delimited by these hyperplanes (possibly open towards infinity). The number of regions of an arrangement can be given in terms of a characteristic function of the arrangement, as shown in a well-known result by Zaslavsky (1975). An arrangement of n_1 hyperplanes in $\mathbb{R}^{n_0}$ has at most $\sum_{j=0}^{n_0} \binom{n_1}{j}$ regions. Furthermore, this number of regions is attained if and only if the hyperplanes are in general position. This implies that the maximal number of linear regions of functions computed by a shallow rectifier network with n_0 inputs and n_1 hidden units is $\sum_{j=0}^{n_0} \binom{n_1}{j}$ (see Pascanu et al. 2013; Proposition 5).

2.3 Deep Neural Networks

We start by defining the identification of input neighborhoods mentioned in the introduction more formally:

Definition 1. A map F *identifies* two neighborhoods S and T of its input domain if it maps them to a common subset $F(S) = F(T)$ of its output domain. In this case we also say that S and T are *identified* by F.

For example, the four quadrants of 2-D Euclidean space are regions that are identified by the absolute value function $g\colon \mathbb{R}^2 \to \mathbb{R}^2$;

$$g(x_1, x_2) = \left[|x_1|\,,\ |x_2|\right]^\top . \tag{2}$$

The computation carried out by the l-th layer of a feedforward network on a set of activations from the $(l-1)$-th layer is effectively carried out for all regions of the input space that lead to the same activations of the $(l-1)$-th layer. One can choose the input weights and biases of a given layer in such a way that the computed function behaves most interestingly on those activation values of the preceding layer which have the largest number of preimages in the input space, thus replicating the interesting computation many times in the input space and generating an overall complicated-looking function.

For any given choice of the network parameters, each hidden layer l computes a function $h_l = g_l \circ f_l$ on the output activations of the preceding layer. We consider the function $F_l\colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$; $F_l := h_l \circ \cdots \circ h_1$ that computes the activations of the l-th hidden layer. We denote the image of F_l by $S_l \subseteq \mathbb{R}^{n_l}$, i.e., the set of (vector valued) activations reachable by the l-th layer for all possible inputs. Given a subset $R \subseteq S_l$, we denote by P_R^l the set of subsets $\bar{R}_1, \dots, \bar{R}_k \subseteq S_{l-1}$ that are mapped by h_l onto R; that is, subsets that satisfy $h_l(\bar{R}_1) = \cdots = h_l(\bar{R}_k) = R$. See Fig. 2 for an illustration.

The number of separate input-space neighborhoods that are mapped to a common neighborhood $R \subseteq S_l \subseteq \mathbb{R}^{n_l}$ can be given recursively as

$$\mathcal{N}_R^l = \sum_{R' \in P_R^l} \mathcal{N}_{R'}^{l-1}, \qquad \mathcal{N}_R^0 = 1, \text{ for each region } R \subseteq \mathbb{R}^{n_0}. \tag{3}$$

For example, P_R^1 is the set of all disjoint input-space neighborhoods whose image by the function computed by the first layer, $h_1\colon \mathbf{x} \mapsto g(\mathbf{W}\mathbf{x} + \mathbf{b})$, equals $R \subseteq S_1 \subseteq \mathbb{R}^{n_1}$.

The recursive formula (3) counts the number of identified sets by moving along the branches of a tree rooted at the set R of the j-th layer's output-space (see Fig. 2 (c)). Based on these observations, we can estimate the maximal number of linear regions as follows.

Lemma 2. *The maximal number of linear regions of the functions computed by an L-layer neural network with piecewise linear activations is at least $\mathcal{N} = \sum_{R \in P^L} \mathcal{N}_R^{L-1}$, where $\mathcal{N}_R^{L-1}$ is defined by Eq.* (3)*, and P^L is a set of neighbordhoods in distinct linear regions of the function computed by the last hidden layer.*

Here, the idea to construct a function with many linear regions is to use the first $L-1$ hidden layers to identify many input-space neighborhoods, mapping all of them to the activation neighborhoods P^L of the $(L-1)$-th hidden layer, each of which belongs to a distinct linear region of the last hidden layer.

We will give the detailed analysis of rectifier and maxout networks in Secs. 3 and 4.

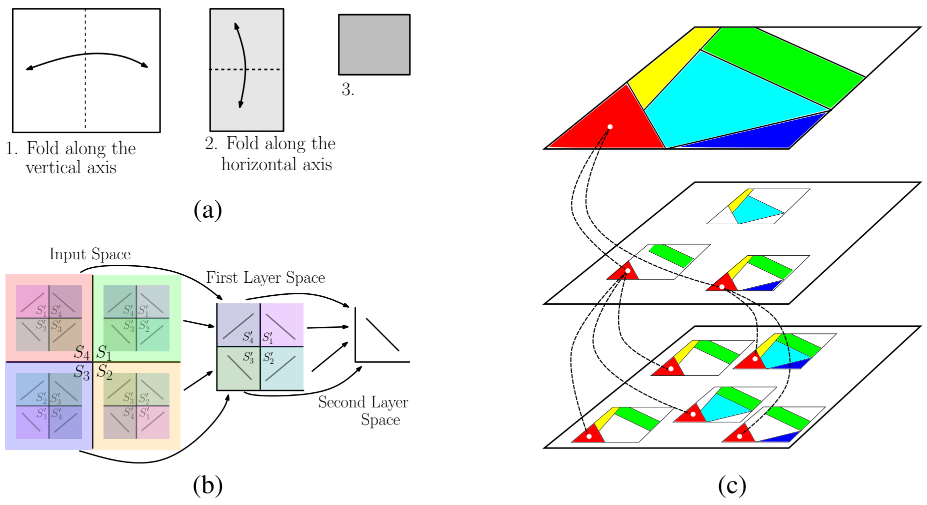


Figure 2: (a) Space folding of 2-D Euclidean space along the two axes. (b) An illustration of how the top-level partitioning (on the right) is replicated to the original input space (left). (c) Identification of regions across the layers of a deep model.

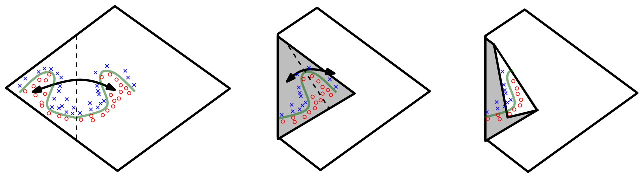


Figure 3: Space folding of 2-D space in a non-trivial way. Note how the folding can potentially identify symmetries in the boundary that it needs to learn.

2.4 Identification of Inputs as Space Foldings

In this section, we discuss an intuition behind Lemma 2 in terms of *space folding*. A map F that identifies two subsets $\mathcal{S}$ and $\mathcal{S}'$ can be considered as an operator that *folds* its domain in such a way that the two subsets $\mathcal{S}$ and $\mathcal{S}'$ coincide and are mapped to the same output. For instance, the absolute value function $g\colon \mathbb{R}^2 \to \mathbb{R}^2$ from Eq. (2) folds its domain twice (once along each coordinate axis), as illustrated in Fig. 2 (a). This folding identifies the four quadrants of 2-D Euclidean space. By composing such operations, the same kind of map can be applied again to the output, in order to re-fold the first folding.

Each hidden layer of a deep neural network can be associated with a folding operator. Each hidden layer folds the space of activations of the previous layer. In turn, a deep neural network effectively folds its input-space recursively, starting with the first layer. The consequence of this recursive folding is that any function computed on the final folded space will apply to all the collapsed subsets identified by the map corresponding to the succession of foldings. This means that in a deep model any partitioning of the last layer's image-space is replicated in all input-space regions which are identified by the succession of foldings. Fig. 2 (b) offers an illustration of this replication property.

Space foldings are not restricted to foldings along coordinate axes and they do not have to preserve lengths. Instead, the space is folded depending on the orientations and shifts encoded in the input weights $\mathbf{W}$ and biases $\mathbf{b}$ and on the nonlinear activation function used at each hidden layer. In particular, this means that the sizes and orientations of identified input-space regions may differ from each other. See Fig. 3.

2.5 Stability to Perturbation

Our bounds on the complexity attainable by deep models (Secs. 3 and 4) are based on suitable choices of the network weights. However, this does not mean that the indicated complexity is only attainable in singular cases.

The parametrization of the functions computed by a neural network is continuous. More precisely, the map $\psi\colon \mathbb{R}^N \to C(\mathbb{R}^{n_0}; \mathbb{R}^{n_L}); \theta \mapsto F_\theta$, which maps input weights and biases $\theta = \{\mathbf{W}_i, \mathbf{b}_i\}_{i=1}^L$ to the continuous functions $F_\theta\colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ computed by the network, is continuous. Our analysis considers the number of linear regions of the functions F_θ. By definition, each linear region contains an open neighborhood of the input-space $\mathbb{R}^{n_0}$. Given any function F_θ with a finite number of linear regions, there is an $\epsilon > 0$ such that for each ϵ-perturbation of the parameter θ, the resulting function $F_{\theta+\epsilon}$ has at least as many linear regions as F_θ. The linear regions of F_θ are preserved under small perturbations of the parameters, because they have a finite volume.

If we define a probability density on the space of parameters, what is the probability of the event that the function represented by the network has a given number of linear regions? By the above discussion, the probability of getting a number of regions at least as large as the number resulting from any particular choice of parameters (for a uniform measure within a bounded domain) is nonzero, even though it may be very small. This is because there exists an epsilon-ball of non-zero volume around that particular choice of parameters, for which at least the same number of linear regions is attained.

For future work it would be interesting to study the partitions of parameter space $\mathbb{R}^N$ into pieces where the resulting functions partition their input-spaces into isomorphic linear regions, and to investigate how many of these pieces of parameter space correspond to functions with a given number of linear regions.

2.6 Empirical Evaluation of Folding in Rectifier MLPs

We empirically examined the behavior of a trained MLP to see if it folds the input-space in the way described above. First, we make the observation that tracing the activation of each hidden unit in this model gives a piecewise linear map $\mathbb{R}^{n_0} \to \mathbb{R}$ (from inputs to activation values of that unit). Hence, we can analyze the behavior of each unit by visualizing the different weight matrices corresponding to the different linear pieces of this map. The weight matrix of one piece of this map can be found by tracking the linear piece used in each intermediary layer, starting from an input example. This visualization technique, a byproduct of our theoretical analysis, is similar to the one proposed by Zeiler and Fergus (2013), but is motivated by a different perspective.

After computing the activations of an intermediary hidden unit for each training example, we can, for instance, inspect two examples that result in similar levels of activation for a hidden unit. With the linear maps of the hidden unit corresponding to the two examples we perturb one of the examples until it results in exactly the same activation. These two inputs then can be safely considered as points in two regions identified by the hidden unit. In Appendix G we provide details and examples of this visualization technique. We also show inputs identified by a deep MLP.

3 Deep Rectifier Networks

In this section we analyze deep neural networks with rectifier units, based on the general observations from Sec. 2. We improve upon the results by Pascanu et al. (2013), with a tighter lower-bound on the maximal number of linear regions of functions computable by deep rectifier networks.

First, let us note the following upper-bound, which follows directly from the fact that each linear region of a rectifier network corresponds to a pattern of hidden units being active:

Proposition 3. *The maximal number of linear regions of the functions computed by any rectifier network with a total of N hidden units is bounded from above by 2^N.*

3.1 Illustration of the Construction

Consider a layer of n rectifiers with n_0 input variables, where $n \geq n_0$. We partition the set of rectifier units into n_0 (non-overlapping) subsets of cardinality $p = \lfloor {}^{n}\!/_{n_0} \rfloor$ and ignore the remainder

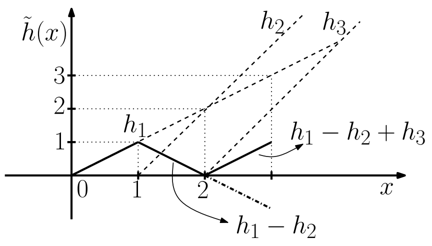


Figure 4: Folding of the real line into equal-length segments by a sum of rectifiers.

units. Consider the units in the j-th subset. We can choose their input weights and biases such that

$$\begin{aligned} h_1(\mathbf{x}) &= \max\{0, \ \mathbf{wx}\}, \\ h_2(\mathbf{x}) &= \max\{0, 2\mathbf{wx} - 1\}, \\ h_3(\mathbf{x}) &= \max\{0, 2\mathbf{wx} - 2\}, \\ &\vdots \\ h_p(\mathbf{x}) &= \max\{0, 2\mathbf{wx} - (p-1)\}, \end{aligned}$$

where $\mathbf{w}$ is a row vector with j-th entry equal to 1 and all other entries set to 0. The product $\mathbf{wx}$ selects the j-th coordinate of $\mathbf{x}$. Adding these rectifiers with alternating signs, we obtain following scalar function:

$$\tilde{h}_j(\mathbf{x}) = \left[1, -1, 1, \dots, (-1)^{p-1}\right] \left[h_1(\mathbf{x}), h_2(\mathbf{x}), h_3(\mathbf{x}), \dots, h_p(\mathbf{x})\right]^\top. \tag{4}$$

Since $\tilde{h}_j$ acts only on the j-th input coordinate, we may redefine it to take a scalar input, namely the j-th coordinate of $\mathbf{x}$. This function has p linear regions given by the following intervals:

$$(-\infty, 0], [0, 1], [1, 2], \dots, [p-1, \infty).$$

Each of these intervals has a subset that is mapped by $\tilde{h}_j$ onto the interval $(0, 1)$, as illustrated in Fig. 4. The function $\tilde{h}_j$ identifies the input-space strips with j-th coordinate $\mathbf{x}_j$ restricted to the intervals $(0, 1), (1, 2), \dots, (p-1, p)$. Consider now all the n_0 subsets of rectifiers and the function $\tilde{h} = \left[\tilde{h}_1, \tilde{h}_2, \dots, \tilde{h}_p\right]^\top$. This function is locally symmetric about each hyperplane with a fixed j-th coordinate equal to $\mathbf{x}_j = 1, \dots, \mathbf{x}_j = p-1$ (vertical lines in Fig. 4), for all $j = 1, \dots, n_0$. Note the periodic pattern that emerges. In fact, the function $\tilde{h}$ identifies a total of p^{n_0} hypercubes delimited by these hyperplanes.

Now, note that $\tilde{h}$ arises from h by composition with a linear function (alternating sums). This linear function can be effectively absorbed in the pre-activation function of the next layer. Hence we can treat $\tilde{h}$ as being the function computed by the current layer. Computations by deeper layers, as functions of the unit hypercube output of this rectifier layer, are replicated on each of the p^{n_0} identified input-space hypercubes.

3.2 Formal Result

We can generalize the construction described above to the case of a deep rectifier network with n_0 inputs and L hidden layers of widths $n_i \geq n_0$ for all $i \in [L]$. We obtain the following lower bound for the maximal number of linear regions of deep rectifier networks:

Theorem 4. *The maximal number of linear regions of the functions computed by a neural network with n_0 input units and L hidden layers, with $n_i \geq n_0$ rectifiers at the i-th layer, is lower bounded by*

$$\left(\prod_{i=1}^{L-1} \left\lfloor \frac{n_i}{n_0} \right\rfloor^{n_0}\right) \sum_{j=0}^{n_0} \binom{n_L}{j}.$$

The next corollary gives an expression for the asymptotic behavior of these bounds. Assuming that $n_0 = O(1)$ and $n_i = n$ for all $i \geq 1$, the number of regions of a single layer model with Ln hidden units behaves as $O(L^{n_0} n^{n_0})$ (see Pascanu et al. 2013; Proposition 10). For a deep model, Theorem 4 implies:

Corollary 5. *A rectifier neural network with n_0 input units and L hidden layers of width $n \geq n_0$ can compute functions that have $\Omega\left(({n}/{n_0})^{(L-1)n_0} n^{n_0} \right)$ linear regions.*

Thus we see that the number of linear regions of deep models grows exponentially in L and polynomially in n, which is much faster than that of shallow models with nL hidden units. Our result is a significant improvement over the bound $\Omega\left(({n}/{n_0})^{L-1} n^{n_0} \right)$ obtained by Pascanu et al. (2013). In particular, our result demonstrates that even for small values of L and n, deep rectifier models are able to produce substantially more linear regions than shallow rectifier models. Additionally, using the same strategy as Pascanu et al. (2013), our result can be reformulated in terms of the number of *linear regions per parameter*. This results in a similar behaviour, with deep models being exponentially more efficient than shallow models (see Appendix C).

4 Deep Maxout Networks

A maxout network is a feedforward network with layers defined as follows:

Definition 6. A *rank-k maxout layer* with n input and m output units is defined by a pre-activation function of the form $f : \mathbb{R}^n \to \mathbb{R}^{m\cdot k}$; $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$, with input and bias weights $\mathbf{W} \in \mathbb{R}^{m\cdot k\times n}$, $\mathbf{b} \in \mathbb{R}^{m\cdot k}$, and activations of the form $g_j(\mathbf{z}) = \max\{\mathbf{z}_{(j-1)k+1}, \dots, \mathbf{z}_{jk}\}$ for all $j \in [m]$. The layer computes a function

$$g \circ f\colon \quad \mathbb{R}^n \to \mathbb{R}^m; \quad \mathbf{x} \mapsto \begin{pmatrix} \max\{f_1(\mathbf{x}), \dots, f_k(\mathbf{x})\} \\ \vdots \\ \max\{f_{(m-1)k+1}(\mathbf{x}), \dots, f_{mk}(\mathbf{x})\} \end{pmatrix}. \tag{5}$$

Since the maximum of two convex functions is convex, maxout units and maxout layers compute convex functions. The maximum of a collection of functions is called their *upper envelope*. We can view the graph of each linear function $f_i\colon \mathbb{R}^n \to \mathbb{R}$ as a supporting hyperplane of a convex set in $(n+1)$-dimensional space. In particular, if each f_i, $i \in [k]$ is the unique maximizer $f_i = \max\{f'_i\colon i' \in [k]\}$ at some input neighborhood, then the number of linear regions of the upper envelope $g_1 \circ f = \max\{f_i\colon i \in [k]\}$ is exactly k. This shows that the maximal number of linear regions of a maxout unit is equal to its rank.

The linear regions of the maxout layer are the intersections of the linear regions of the individual maxout units. In order to obtain the number of linear regions for the layer, we need to describe the structure of the linear regions of each maxout unit, and study their possible intersections.

Voronoi diagrams can be lifted to upper envelopes of linear functions, and hence they describe input-space partitions generated by maxout units. Now, how many regions do we obtain by intersecting the regions of m Voronoi diagrams with k regions each? Computing the intersections of Voronoi diagrams is not easy, in general. A trivial upper bound for the number of linear regions is k^m, which corresponds to the case where all intersections of regions of different units are different from each other. We will give a better bound in Proposition 7.

Now, for the purpose of computing lower bounds, here it will be sufficient to consider certain well-behaved special cases. One simple example is the division of input-space by $k-1$ parallel hyperplanes. If $m \leq n$, we can consider the arrangement of hyperplanes $H_i = \{\mathbf{x} \in \mathbb{R}^n\colon \mathbf{x}_j = i\}$ for $i = 1, \dots, k-1$, for each maxout unit $j \in [m]$. In this case, the number of regions is k^m. If $m > n$, the same arguments yield k^n regions.

Proposition 7. *The maximal number of regions of a single layer maxout network with n inputs and m outputs of rank k is lower bounded by $k^{\min\{n,m\}}$ and upper bounded by $\min\{\sum_{j=0}^{n} \binom{k^2 m}{j}, k^m\}$.*

Now we take a look at the deep maxout model. Note that a rank-2 maxout layer can be simulated by a rectifier layer with twice as many units. Then, by the results from the last section, a rank-2 maxout network with $L-1$ hidden layers of width $n = n_0$ can identify $2^{n_0(L-1)}$ input-space regions, and, in turn, it can compute functions with $2^{n_0(L-1)}2^{n_0} = 2^{n_0 L}$ linear regions.

For the rank-k case, we note that a rank-k maxout unit can identify k cones from its input-domain, whereby each cone is a neighborhood of the positive half-ray $\{r\mathbf{W}_i \in \mathbb{R}^n\colon r \in \mathbb{R}_+\}$ corresponding to the gradient $\mathbf{W}_i$ of the linear function f_i for all $i \in [k]$. Elaborating this observation, we obtain:

Theorem 8. *A maxout network with L layers of width n_0 and rank k can compute functions with at least $k^{L-1}k^{n_0}$ linear regions.*

Theorem 8 and Proposition 7 show that deep maxout networks can compute functions with a number of linear regions that grows exponentially with the number of layers, and exponentially faster than the maximal number of regions of shallow models with the same number of units. Similarly to the rectifier model, this exponential behavior can also be established with respect to the number of network parameters.

We note that although certain functions that can be computed by maxout layers can also be computed by rectifier layers, the rectifier construction from last section leads to functions that are not computable by maxout networks (except in the rank-2 case). The proof of Theorem 8 is based on the same general arguments from Sec. 2, but uses a different construction than Theorem 4 (details in Appendix D).

5 Conclusions and Outlook

We studied the complexity of functions computable by deep feedforward neural networks in terms of their number of linear regions. We specifically focused on deep neural networks having piecewise linear hidden units which have been found to provide superior performance in many machine learning applications recently. We discussed the idea that each layer of a deep model is able to identify pieces of its input in such a way that the composition of layers identifies an exponential number of input regions. This results in exponentially replicating the complexity of the functions computed in the higher layers of the model. The functions computed in this way by deep models are complicated, but still they have an intrinsic rigidity caused by the replications, which may help deep models generalize to unseen samples better than shallow models.

This framework is applicable to any neural network that has a piecewise linear activation function. For example, if we consider a convolutional network with rectifier units, as the one used in (Krizhevsky et al. 2012), we can see that the convolution followed by max pooling at each layer identifies all patches of the input within a pooling region. This will let such a deep convolutional neural network recursively identify patches of the images of lower layers, resulting in exponentially many linear regions of the input space.

The parameter space of a given network is partitioned into the regions where the resulting functions have corresponding linear regions. This correspondence of the linear regions of the computed functions can be described in terms of their adjacency structure, or a poset of intersections of regions. Such combinatorial structures are in general hard to compute, even for simple hyperplane arrangements. One interesting question for future analysis is whether many regions of the parameter space of a given network correspond to functions which have a given number of linear regions.

References

M. Anthony and P. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

D. Ciresan, U. Meier, J. Masci, and J. Schmidhuber. Multi column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

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

O. Delalleau and Y. Bengio. Shallow vs. deep sum-product networks. In *NIPS*, 2011.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011.

I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *Proceedings of The 30th International Conference on Machine Learning (ICML'2013)*, 2013.

G. Hinton, L. Deng, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov. 2012.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

O. Krause, A. Fischer, T. Glasmachers, and C. Igel. Approximation properties of DBNs with binary hidden units and real-valued visible units. In *Proceedings of The 30th International Conference on Machine Learning (ICML'2013)*, 2013.

A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*. 2012.

N. Le Roux and Y. Bengio. Deep belief networks are compact universal approximators. *Neural Computation*, 22(8):2192–2207, Aug. 2010.

G. Montúfar. Universal approximation depth and errors of narrow belief networks with discrete units. *Neural Computation*, 26, July 2014.

G. Montúfar and N. Ay. Refinements of universal approximation results for deep belief networks and restricted Boltzmann machines. *Neural Computation*, 23(5):1306–1319, May 2011.

V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In L. Bottou and M. Littman, editors, *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*, pages 807–814. ACM, 2010.

R. Pascanu and Y. Bengio. Revisiting natural gradient for deep networks. In *International Conference on Learning Representations*, 2014.

R. Pascanu, G. Montúfar, and Y. Bengio. On the number of inference regions of deep feed forward networks with piece-wise linear activations. *arXiv:*`1312.6098 [cs.LG]`, Dec. 2013.

R. Stanley. An introduction to hyperplane arrangements. In *Lect. notes, IAS/Park City Math. Inst.*, 2004.

J. Susskind, A. Anderson, and G. E. Hinton. The Toronto face dataset. Technical Report UTML TR 2010-001, U. Toronto, 2010.

T. Zaslavsky. *Facing Up to Arrangements: Face-Count Formulas for Partitions of Space by Hyperplanes*. Number no. 154 in Memoirs of the American Mathematical Society. American Mathematical Society, 1975.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. Technical Report Arxiv 1311.2901, 2013.

A Identification of Input-Space Neighborhoods

Proof of Lemma 2. Each output-space neighborhood $R \in P^L$ has as preimages all input-space neighborhoods that are R-identified by η_L (i.e., the input-space neighborhoods whose image by η_L, the function computed by the first L-layers of the network, equals R). The number of input-space preimages of R is denoted $\mathcal{N}_R^L$. If each $R \in P^L$ is the image of a distinct linear region of the function $h_L = g_L \circ f_L$ computed by the last layer, then, by continuity, all preimages of all different $R \in P^L$ belong to different linear regions of η_L. Therefore, the number of linear regions of functions computed by the entire network is at least equal to the sum of the number of preimages of all $R \in P^L$, which is just $\mathcal{N} = \sum_{R \in P^L} \mathcal{N}_R^{L-1}$. □

B Rectifier Networks

Proof of Theorem 4. The proof is done by counting the number of regions for a suitable choice of network parameters. The idea of the construction is to divide the first $L-1$ layers of the network into n_0 independent parts; one part for each input neuron. The parameters of each part are chosen in such a way that it folds its one-dimensional input-space many times into itself. For each part, the number of foldings per layer is equal to the number of units per layer. See Fig. 5.

As outlined above, we organize the units of each layer into n_0 non-empty groups of units of sizes $p_1, \dots, p_{n_0}$. A simple choice of these sizes is, for example, $p_1 = \cdots = p_{n_0} = \lfloor n_l / n_0 \rfloor$ for a layer of width n_l, dropping the remainder units. We define the input weights of each group in such a way that the units in that group are sensitive to only one coordinate of the n_0-dimensional input-space. By the discussion from Sec. 3.1, choosing the input and bias weights in the right way, the alternating sum of the activations of the p units within one group folds their input-space coordinate p times into itself. Since the alternating sum of activations is an affine map, it can be absorbed in the pre-activation function of the next layer. In order to make the arguments more transparent, we view the alternating sum $\tilde{h} = h_1 - h_2 + \cdots \pm h_p$ of the activations of the p units in a group as the activation of a fictitious intermediary unit. The compound output of all these n_0 intermediary units partitions the input-space, $\mathbb{R}^{n_0}$, into a grid of $\prod_{i=1}^{n_0} p_i$ identified regions. Each of these input-space regions is mapped to the n_0-dimensional unit cube in the output-space of the intermediary layer. We view this unit cube as the *effective* inputs for the next hidden layer, and repeat the construction. In this way, with each layer of the network, the number of identified regions is multiplied by $\prod_{i=1}^{n_0} p_i$, according to Lemma 2. In the following we discuss the folding details explicitly.

Consider one of the network parts and consider the weights used in Sec. 3.1. The function $\tilde{h}$ computes an alternating sum of the responses h_k for $k \in [p]$. It is sufficient to show that this sum folds the input-coordinate p times into the interval $(0, 1)$. Inspecting the values of h_k, we see that we only need to explore the intervals $(0, 1), (1, 2), \dots, (p-1, p)$.

Consider one of these intervals, $(k-1, k)$, for some $k \in [p]$. Then, for all $x \in (k-1, k)$, we have $\tilde{h}(x) = x + 2\sum_{i=1}^{k-1}(-1)^i(x-i) = (-1)^{k-1}(x-(k-1)) - \frac{1}{2}((-1)^{k-1} - 1)$. Hence $\tilde{h}(k-1) = -\frac{1}{2}((-1)^{k-1} - 1)$ and $\tilde{h}(k) = (-1)^{k-1} - \frac{1}{2}((-1)^{k-1} - 1)$. One of the two values is always zero and the other one, and so $\tilde{h}(\{k-1, k\}) = \{0, 1\}$. Since the function is linear between $k-1$ and k, we obtain that $\tilde{h}$ maps the interval $(k-1, k)$ to the interval $(0, 1)$.

In total, the number of input-space neighborhoods that are mapped by the first $L-1$ layers onto the (open) unit hypercube $(0, 1)^{n_0}$ of the (effective) output space of the $(L-1)$-th layer is given by

$$\mathcal{N}_{(0,1)^{n_0}}^{L-1} = \prod_{l=1}^{L-1} \prod_{i=1}^{n_0} p_{l,i}, \tag{6}$$

where $p_{l,i}$ is the number of units in the i-th group of units in the l-th layer.

The inputs and bias of the last hidden layer can be chosen in such a way that the function h_L partitions its (effective) input neighborhood $(0, 1)^{n_0}$ by an arrangement of n_L hyperplanes in general position, i.e., into $\sum_{j=0}^{n_0} \binom{n_L}{j}$ regions (see Sec. 2.2).

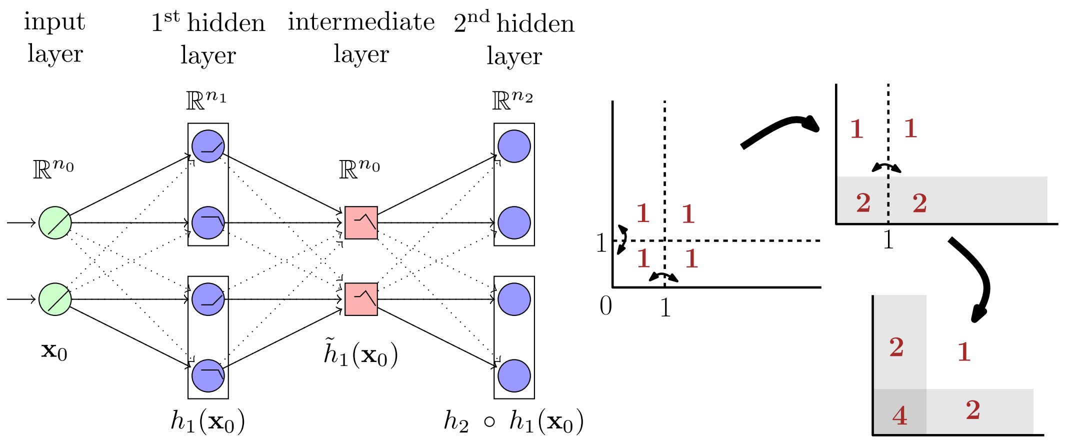


Figure 5: Illustration of the proof of Theorem 8. The figure shows a rectifier network that has been divided into n_0 independent parts. Each part is sensitive only to one coordinate of the input-space. Each layer of each part is fed to a fictitious intermediary affine unit (computed by the preactivation function of the next layer), which computes the activation value that is passed to the next layer. Illustration of a function computed by the depicted rectifier network for $n_0 = 2$, at the intermediary layer. The function is composed of two foldings; the first pair of hidden units fold the input-space $\mathbb{R}^2$ along a line parallel to the x-axis, and the second pair, along a line parallel to the y-axis.

Let m_l denote the remainder of n_l divided by n_0. Choosing $p_{l,1} = \cdots = p_{l,n_0-m_l} = \lfloor n_l/n_0 \rfloor$ and $p_{l,n_0-m_l+1} = \cdots = p_{l,n_0} = \lfloor n_l/n_0 \rfloor + 1$, we obtain a total of linear regions

$$\left(\prod_{l=1}^{L-1} \left\lfloor \frac{n_l}{n_0} \right\rfloor^{n_0-m_l} \left(\left\lfloor \frac{n_l}{n_0} \right\rfloor + 1 \right)^{m_l} \right) \sum_{j=0}^{n_0} \binom{n_L}{j}. \tag{7}$$

This is equal to the bound given in theorem when all remainders $m_l = n_l - n_0\lfloor n_l/n_0 \rfloor$ are zero, and otherwise it is larger. This completes the proof. □

C Our Bounds in terms of Parameters

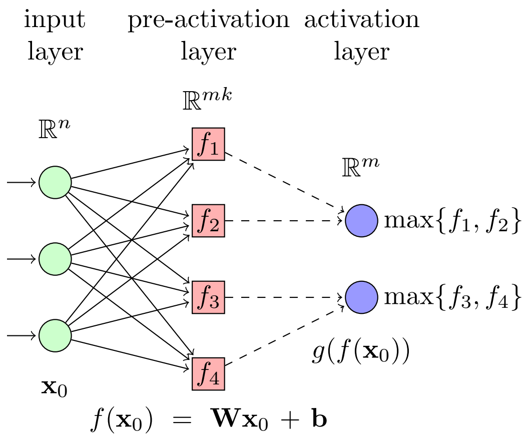
We computed bounds for the maximal number of linear regions of the functions computable by different networks in terms of their number of hidden units. It is not difficult to express these results in terms of the number of parameters of the networks and to derive expressions for the asymptotic rate of growth of the number of linear regions per added parameter. This kind of expansions have been computed in Pascanu et al. (2013; Proposition 8). The number of parameters of a deep model with L layers of width n behaves as $\Theta(Ln^2)$, i.e., it is bounded above and below by Ln^2, asymptotically. The number of parameters of a shallow model with Ln hidden units behaves as $\Theta(Ln)$. Our Theorem 4 and the discussion of shallow networks given in Sec. 2.2, imply the following asymptotic rates (number of linear regions per parameter):

- For a deep model: $\Omega\left((n/n_0)^{n_0(L-1)} \frac{n^{n_0-2}}{L} \right)$.
- For a shallow model: $O\left(L^{n_0-1} n^{n_0-1} \right)$.

This shows that, for deep models, the maximal number of linear regions grows exponentially fast with the number of parameters, whereas, for shallow models, it grows only polynomially fast with the number of parameters.

D Maxout Networks

Proof of Proposition 7. Here we investigate the maximal number of linear regions of a rank-k maxout layer with n inputs and m outputs. In the case of rectifier units, the solutions is simply the



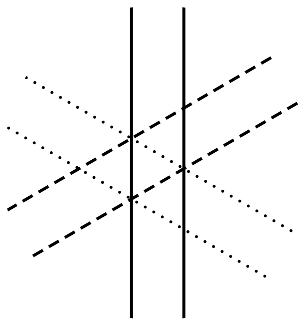


Figure 6: Illustration of a rank-2 maxout layer with $n = 3$ inputs and $m = 2$ outputs. The pre-activation function maps the input into mk-dimensional space, where $k = 2$ is the rank of the layer. The activation function maximizes over groups of k preactivation values. Illustration of the 3-dimensional Shi arrangement $\mathcal{S}_3 \subset \mathbb{R}^3$ (depicted is the intersection with $\{\mathbf{x} \in \mathbb{R}^3 \colon \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 = 1\}$). This arrangement corresponds an input-space partition of a rank-3 maxout layer with $n = 3$ inputs and $m = 3$ outputs (for one choice of the parameters). Each pair of parallel hyperplanes delimits the linear regions of one maxout unit.

maximal number of regions of a hyperplane arrangement. In the case of maxout units, we do not have hyperplane arrangements. However, we can upper bound the number of linear regions of a maxout layer by the number of regions of a hyperplane arrangement. The arguments are as follows.

As mentioned in Sec. 4, each maxout unit divides its input into the linear regions of an upper envelope of k real valued linear functions. In other words, the input space is divided by pieces of hyperplanes defining the boundaries between inputs where one entry of the pre-activation vector is larger than another. There are at most k^2 such boundaries, since each of them corresponds to the solution set of an equation of the form $f_i(\mathbf{x}) = f_j(\mathbf{x})$. If we extend each such boundary to a hyperplane, then the number of regions can only go up.

The linear regions of the layer are given by the intersections of the regions of the individual units. Hence, the number of linear regions of the layer is upper bounded (very loosely) by the number of regions of an arrangement of $k^2 \cdot m$ hyperplanes in n-dimensional space. By Zaslavsky (1975), the latter is $\sum_{j=0}^{n} \binom{k^2 m}{j}$, which behaves as $O((k^2 m)^n)$, i.e., polynomially in k and in m. □

Proof of Theorem 8. Consider a network with $n = n_0$ maxout units of rank k in each layer. See Fig. 5. We define the seeds of the maxout unit q_j such that $\{\mathbf{W}_{i,:}\}_i$ are unit vectors pointing in the positive and negative direction of $\lfloor k/2 \rfloor$ coordinate vectors. If k is larger than $2n_0$, then we forget about $k - 2n_0$ of them (just choose $\mathbf{W}_{i,:} = 0$ for $i > 2n_0$). In this case, q_j is symmetric about the coordinate hyperplanes with normals e_i with $i \leq \lfloor k/2 \rfloor$ and has one linear region for each such i, with gradient e_i. For the remaining q_j we consider similar functions, whereby we change the coordinate system by a slight rotation in some independent direction.

This implies that the output of each $q_j \circ (f_1, \ldots, f_k)$ is an interval $[0, \infty)$. The linear regions of each such composition divide the input space into r regions $R_{j,1}, \ldots, R_{j,k}$. Since the change of coordinates used for each of them is a slight rotation in independent directions, we have that $R_i := \cap_j R_{j,i}$ is a cone of dimension n_0 for all $i \in [k]$. Furthermore, the gradients of $q_j \circ f_j$ for $j \in [n_0]$ on each R_i are a basis of $\mathbb{R}^{n_0}$. Hence the image of each R_i by the maxout layer contains an open cone of $\mathbb{R}^{n_0}$ which is identical for all $i \in [k]$. This image can be shifted by bias terms such that the effective input of the next layer contains an open neighbourhood of the origin of $\mathbb{R}^{n_0}$.

The above arguments show that a maxout layer of width n_0 and rank k can identify at least k regions of its input. A network with $L - 1$ layers with therefore identify k^{L-1} regions of the input. □

In Sec. 4 we mentioned that maxout layers can compute functions whose linear regions correspond to intersections of Voronoi diagrams. Describing intersections of Voronoi diagrams is difficult, in general. There are some superpositions of Voronoi diagrams that correspond to hyperplane arrangements which are well understood. Here are two particularly nice examples:

Example 9. Consider a layer with n inputs and $m = n(n-1)/2$ rank-3 maxout units labeled by the pairs (i, j), $1 \leq i < j \leq n$. The input and bias weights can be chosen in such a way that the regions of unit (i, j) are delimited by the two hyperplanes $H_{(i,j),s} = \{\mathbf{x} \in \mathbb{R}^n \colon \mathbf{x}_i - \mathbf{x}_j = s\}$ for $s \in \{0, 1\}$. The intersections of the regions of all units are given by the regions of the hyperplane arrangement $\{H_{(i,j),s}\}_{1 \leq i < j \leq n, s=1,2}$, which is known as the *Shi arrangement* $\mathcal{S}_n$ and has $(n+1)^{n-1}$ regions. The right panel of Fig. 6 illustrates the Shi arrangement $\mathcal{S}_3$.

A related arrangement, corresponding to rank-4 maxout units, is the *Catalan arrangement*, which has triplets of parallel hyperplanes, and a total of $n!C_n$ regions, where $C_n := \frac{1}{n+1}\binom{2n}{n}$ is the *Catalan number*. For details on these arrangements see (Stanley 2004; Corollary 5.1 and Proposition 5.15).

E Other Networks

In the introduction we mention that our analysis of rectifier and maxout networks serves as a platform to study other types of feedforward neural networks. Without going into many details, we exemplify this for the particular case of convolutional networks. A convolutional network is a network whose units take values in a space of *features* (real valued arrays) and whose edges pass features by convolution with *filters* (real valued arrays). Since convolution is a linear map, the preactivation function of a convolutional network is of the same form as the preactivation functions considered in this paper. Its output is a feature, but it can be written as a vector, like the $f_{l,i}$'s that we considered. Hence convolutional networks with piecewise linear activations fall in the class of networks that we considered here. The only difference lies in that the corresponding input weight matrices of convolutional networks belong to restricted classes of matrices.

F Sinusoidal Boundary Experiment

Here, we describe the experiments we performed to obtain Fig. 1 in the main text.

In this experiment we considered two MLPs, of which one has a single hidden layer with 20 hidden units and the other has two hidden layers with 10 hidden unit each. The MLPs were trained on the same synthetic dataset using a conjugate natural gradient (Pascanu and Bengio 2014) which was used to minimize the effect of optimization. We plot the best of several runs. The shallow model misclassified 123 examples, whereas the deep model did only 24 examples. The two-layer model is better at capturing a sinusoidal decision boundary, because it can define more linear regions.

G Visualizing the Behaviour of Hidden Units in Higher Layers

In this section, we describe the details on how one can visualize the effect of folding in rectifier MLPs, discussed in Sec. 2.6.

Any piecewise linear function is fully defined by the different linear pieces from which it is composed. Each piece is given by its domain–a region of the input space $R_i \subseteq \mathbb{R}^{n_0}$–and the linear map f_i that describes its behaviour on R_i. Because f_i is an affine map, it can be interpreted in the same way hidden units in a shallow model are. Namely, we can write f_i as:

$$f_i(\mathbf{x}) = \mathbf{u}^\top \mathbf{x} + c, \quad \mathbf{x} \in R_i,$$

where $\mathbf{u}^\top$ is a row vector, and $\mathbf{u}^\top \in \mathbb{R}^{n_0}$. Then f_i measures the (unnormalized) cosine distance between $\mathbf{x}$ and $\mathbf{u}^\top$. If $\mathbf{x}$ is some image, $\mathbf{u}^\top$ is also an image and shows the pattern (template) to which the unit responds whenever $\mathbf{x} \in R_i$.

Given an input example $\mathbf{x}$ from an arbitrary region R_i of the input space we *can construct the corresponding linear map* f_i generated by the j-th unit at the l-th layer. Specifically, the weight $\mathbf{u}^\top$ of the linear map f_i is computed by

$$\mathbf{u}^\top = (\mathbf{W}_l)_{j:} \operatorname{diag}\left(\mathbf{I}_{f_{l-1}>0}(\mathbf{x})\right) \mathbf{W}_{l-1} \cdots \operatorname{diag}\left(\mathbf{I}_{f_1>0}(\mathbf{x})\right) \mathbf{W}_1. \quad (8)$$

A bias of the linear map can be similarly computed.

From Eq. (8), we can see that the linear map of a specific hidden unit $f_{l,j}$ at a layer l is found by keeping track of which linear piece is used at each layer until the layer l ($\mathbf{I}_{f_p>0}, p < l$ – which is the indicator function). At the end, the j-th row of the weight matrix $\mathbf{W}_l$ ($(\mathbf{W}_l)_{j:}$) is multiplied. Although we present a formula specific to the rectifier MLP, it is straightforward to adapt this to any MLP with a piecewise linear activation, such as a convolutional neural network with a maxout activation.

From the fact that the linear map computed by Eq. (8) depends on each sample/point $\mathbf{x}$, we need to traverse a set of points (e.g., training samples) to identify different linear responses of a hidden unit. While this does not give all possible responses, if the set of points is large enough, we can get sufficiently many to provide a better understanding of its behaviour.

We trained a rectifier MLP with three hidden layer on Toronto Faces Dataset (TFD) (Susskind et al. 2010). The first two hidden layers have 1000 hidden units each and the last one has 100 units.

We trained the model using stochastic gradient descent. We used, as regularization, an L_2 penalty with a coefficient of 10^{-3}, dropout on the first two hidden layers (with a drop probability of 0.5) and we enforced the weights to have unit norm column-wise by projecting the weights after each SGD step. We used a learning rate of 0.1 and the output layer is composed of sigmoid units. The purpose of these regularization schemes, and the sigmoid output layer is to obtain cleaner and sharper filters. The model is trained on fold 1 of the dataset and achieves an error of 20.49% which is reasonable for this dataset and a non-convolutional model.

Since each of the first layer hidden units only responds to a single linear region, we directly visualize the learned weight vectors of the 16 randomly selected units in the first hidden layer. These are shown on the top row of Fig. 7.

On the other hand, for any other hidden layer, we randomly pick 20 units per layer and visualize the most interesting four units out of them based on the maximal Euclidean distance between the different linear responses of each unit. The linear responses of each unit are computed by clustering the responses obtained on the training set (we only consider those responses where the activation was positive) into four clusters using K-means algorithm. We show the representative linear response in each of the clusters (see the second and third rows of Fig. 7).

Similarly, we visualize the linear maps learned by each of the output unit. For the output layer, we show the visualization of all seven units in Fig. 8

By looking at the differences among the distinct linear regions that a hidden unit responds to, we can investigate the type of invariance the unit learned. In Fig. 9, we show the differences among the four linear maps learned by the last visualized hidden unit of the third hidden layer (the last column of the visualized linear maps).

From these visualizations, we can see that a hidden unit learns to be invariant to more abstract and interesting translations at higher layers. We also see the types of invariance of a hidden unit in a higher layer clearly.

Zeiler and Fergus (2013) attempt to visualize the behaviour of units in the upper layer, specifically, of a deep convolutional network with rectifiers. This approach is to some extent similar to our approach proposed here, except that we do not make any other assumption beside that a hidden unit in a networks uses a piece-wise linear activation function.

The perspective from which the visualization is considered is also different. Zeiler and Fergus (2013) approaches the problem of visualization from the perspective of (approximately) inverting the feedforward computation of the neural network, whereas our approach is derived by identifying a set of linear maps per hidden unit.

This difference leads to a number of minor differences in the actual implementation. For instance, Zeiler and Fergus (2013) approximates the inverse of a rectifier by simply using another rectifier. On the other hand, we do not need to approximate the inverse of the rectifier. Rather, we try to identify regions in the input space that maps to the same activation.

In our approach, it is possible to visualize an actual point in the input space that maps to the same activation of a hidden unit. In Fig. 10, we show three distinct points in the input space that activates

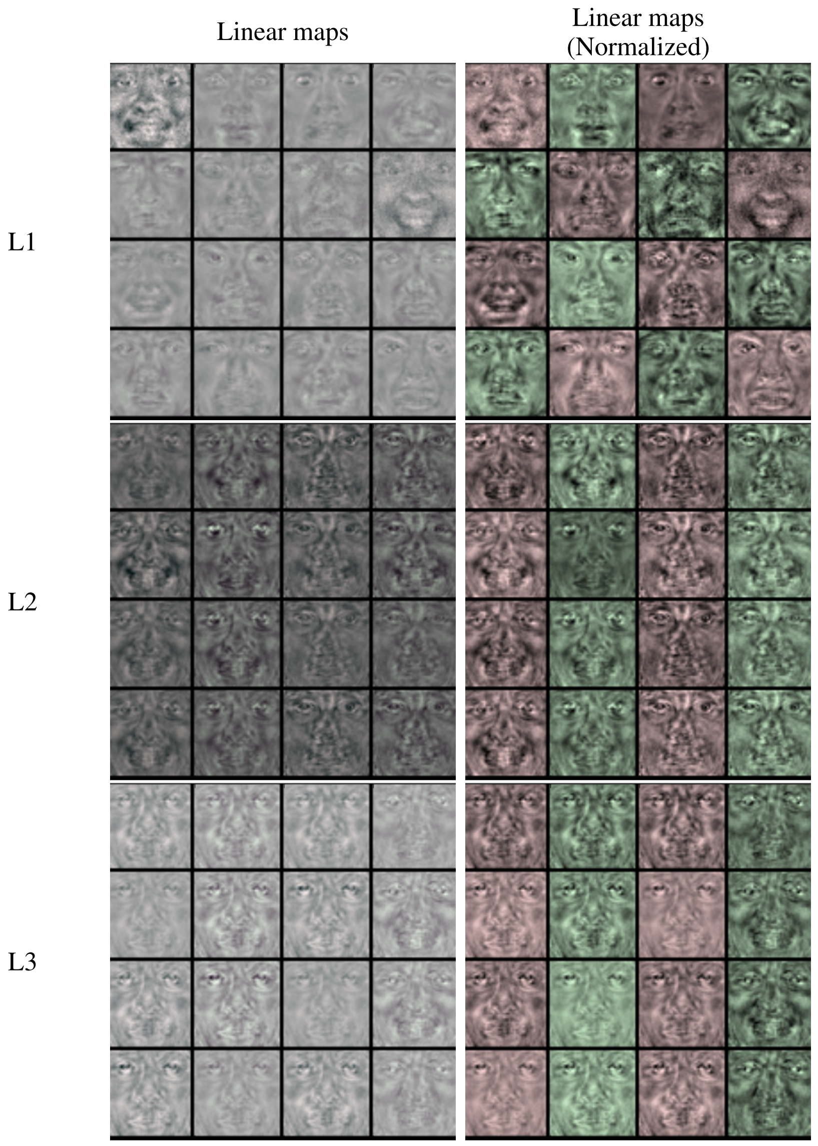


Figure 7: Visualizations of the linear maps learned by each hidden layer of a rectifier MLP trained on TFD dataset. Each row corresponds to each hidden layer. The first column shows the unnormalized linear maps, and the last column shows the normalized linear maps showing only the direction of each map. Colors are only used to improve the distinction among different filters.

a randomly chosen hidden unit in the third hidden layer to be exactly 2.5. We found these points by first finding three training samples that map to an activation close to 2.5 of the same hidden unit, and from each found sample, we search along the linear map (computed by Eq. (8)) for a point that exactly results in the activation of 2.5. Obviously, the found point is *n*ot one of the training samples. From those three points, we can see that the chosen hidden unit responds to a face with wide-open mouth and a set of open eyes while being invariant to other features of a face (e.g., eye brows). By the pertubation analysis, we can assume that there is an open set around each of these points that are identified by the hidden unit.

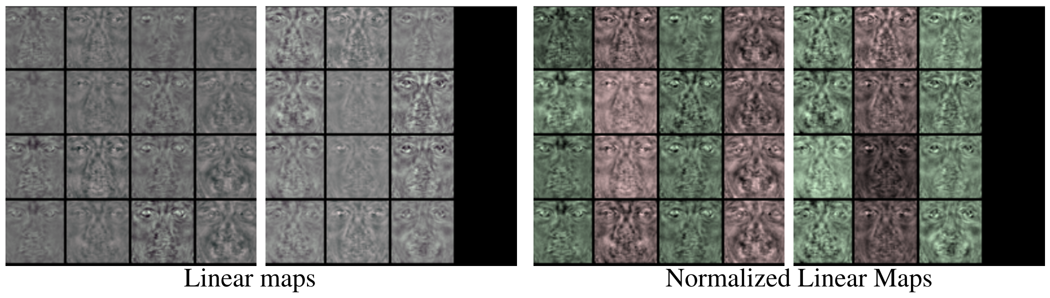


Figure 8: Linear maps of the output units of the rectifier MLP trained on TFD dataset. The corresponding class labels for the columns are (1) anger, (2) disgust, (3) fear, (4) happy, (5) sad, (6) surprise and (7) neutral.

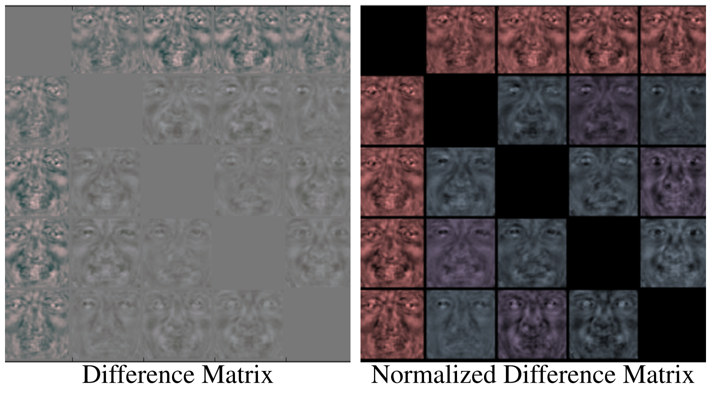


Figure 9: Differences among the distinct linear regions of a single hidden unit at the third hidden layer of the rectifier MLP trained on TFD.

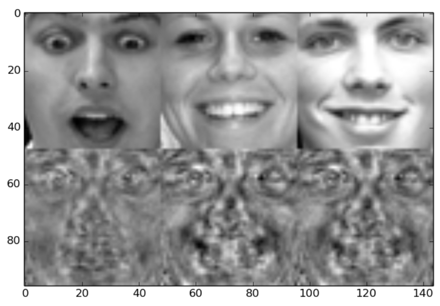


Figure 10: The visualization of three distinct points in the input space that map to the same activation of a randomly chosen hidden unit at the third hidden layer. The top row shows three points (*n*ot training/test samples) in the input space, and for each point, we plot the linear map below.