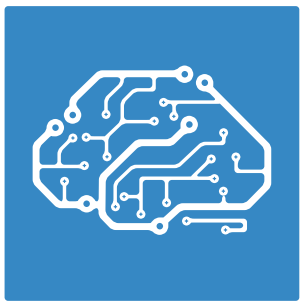

Center for Brains, Minds & Machines

CBMM Memo No. 037 **29 September 2015**

Notes on Hierarchical Splines, DCLNs and i-theory

by

Tomaso Poggio, Lorenzo Rosasco,Amnon Shashua, Nadav Cohen and Fabio Anselmi

Abstract

We define an extension of classical additive splines for multivariate function approximation that we call hierarchical splines. We show that the case of hierarchical, additive, piece-wise linear splines includes present-day Deep Convolutional Learning Networks (DCLNs) with linear rectifers and pooling (sum or max). We discuss how these observations together with i-theory may provide a framework for a general theory of deep networks.

This work was supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF - 1231216.

Notes on Hierarchical Splines, DCLNs and i-theory

Tomaso Poggio, Lorenzo Rosasco,Amnon Shashua,
Nadav Cohen and Fabio Anselmi

September 29, 2015

Abstract

We define an extension of classical additive splines for multivariate function approximation that we call *hierarchical splines*. We show that the case of hierarchical, additive, piece-wise linear splines includes present-day Deep Convolutional Learning Networks (DCLNs) with linear rectifiers and pooling (sum or max). We discuss how these observations together with i-theory may provide a framework for a general theory of deep networks.

1 Introduction

We interpret present-day DCLNs as an interesting hierarchical extension of classical function approximation techniques, in particular additive linear splines. Other classical approximations techniques such as tensor product splines and radial basis functions can be extended to hierarchical architectures. Our framework builds upon the close connections between DCLNs and past results in i-theory which was developed to characterize a class of neurophysiologically plausible algorithms for invariant pattern recognition. Many of our observations and results are in the Remarks part of the sections. We assume knowledge of i-theory results and definitions [1].

2 DCLNs

Consider the network module depicted in Figure 1, a). We give a description of standard DCLNs using i-theory notation [2].

The open circle represents a complex cell; $\bigwedge$ represents its receptive field and pooling range. Within each receptive field $\bigwedge$ there are several simple cells. Each performs an inner product of the patch x^1 (which is a vector of dimensionality D^1) of the image x (a vector of dimensionality D) within $\bigwedge$ with another vector t^q (called a template or a filter, or a kernel) with $q = 1, \cdots, Q$, followed by a non linearity; the dot product followed by the nonlinearity is indicated by a

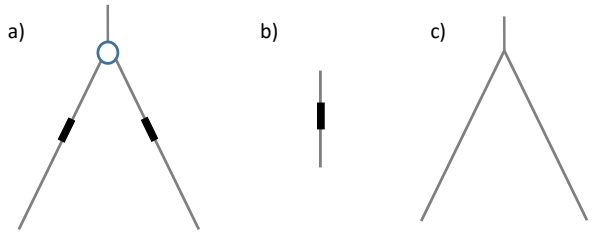

Figure 1: *Basic motifs of the DCLNs networks described in the paper (note that for present-day DCLNs all the filters in the motifs below are "convolutional" across the whole image): a) A pooling module with linear rectifier as nonlinearities. Subsampling after pooling is usual but is not mandatory; it is suggested here by the graph which has two lines in and one out. Its output is* $\phi_q(x) = \sum_g |\langle x, gt^q \rangle + b^q|_+$ *or* $(\phi_q(x) = (\sum_g |\langle x, gt^q \rangle + b^q|_+)^p)^{\frac{1}{p}} \approx \phi_q(x) = max_{g \in G} |\langle x, gt^q \rangle + b^q|_+$. *The dot products are denoted by the thin lines, the nonlinearity by the black rectangle; the nodes (open circles) represent pooling* $\sum_g$ *or* max_g. *In the language of i-theory each open circle represents a complex cell pooling the outputs of the associated simple cells – typically many of them within each* $\bigwedge$. *Each simple cell computes a nonlinear transformation of the dot product between a patch of the image in the receptive field of the complex cell and a template. There are several different types of simple cells (different* t^q *per each complex cell). The* $\bigwedge$ *represents the receptive field of the complex cell – the part of the (neural) image visible to the module (for translations this is also the pooling range). b) A degenerate pooling module (in a* 1×1 *convolution layer) of a DCLN network. The dot products are followed by nonlinearities; the latter are denoted by the black rectangles. In this case there is no complex cell and no pooling but a set of* Q *types of simple cells, each representing a different template (untransformed). Thus the operation performed is* $|(\langle x, gt^q \rangle + b^q)|_+, q = 1, \cdots, Q$. *c) shows the "unfolding" of b) in the case of* $Q = 2$ *channels. The unit represented by the vertical line at the top is a (linear) simple cell which combines linearly through the associated template the two incoming inputs* $(x_1, x_2) = x$ *to yield:* $\langle x, gt^q \rangle + b_q, q = 1, 2$. *The t filter in this case combines linearly the two different components of* x.

black rectangle. The nonlinearity in most of today's DCLNs is a linear rectifier: thus $|\langle x, t\rangle + b|_+$. These first two steps can be seen to roughly correspond to the neural response of a so called simple cell [3, 4]. The complex cell performs a *pooling* operation on the simple cells outputs, aggregating in a single output the values of the different inner products that correspond to transformations of the same template. Usually there are fewer "complex cells" than simple cells; in other words, pooling is usually followed by subsampling. Complex cells use mostly a max [4] (called maxout in the deep learning community) operation over a set of transformations (see for instance [1]) $g \in G$ (such as x, y translations in standard DCLNs) of templates (called weights) t:

$$\max_{g \in G} |\langle x, gt\rangle + b|_+, \tag{1}$$

As described in a later section the max can be computed as

$$\mu(I) = h(\frac{1}{|G|}\sum_{g \in G} \eta(\langle x, gt\rangle)), \tag{2}$$

where $h(z) = (z^p)^{\frac{1}{p}}$, which is close to the max for large p.
An alternative pooling is provided by the average

$$\frac{1}{|G|}\sum_{g \in G} |\langle x, gt\rangle + b|_+. \tag{3}$$

A more flexible alternative is the softmax pooling (which approximates the max for "large" n) written as

- $$\sum_{g \in G} \frac{|\langle x, gt\rangle + b|_+^n}{\sum_{g'} 1 + |\langle x, gt\rangle + b|_+^{n-1}}, \tag{4}$$

  (which can be implemented by simple circuits of the lateral inhibition type (see [5])) or as

- by the MEX (with appropriate parameter values, see Appendix)

$$\frac{1}{\xi}\log\Big(\frac{1}{n}\sum_{i=1}^{n}\exp(\xi|\langle x, g_i t\rangle + b_i|_+)\Big).$$

The set of operations – dot products, nonlinearity, pooling – is iterated at each layer. Note that there are Q channels with different t^q for each input and edge of the graph, where Q varies from layer to layer (in today's DCLNs $Q = 3$ for each image pixel in the first layer). If G contains only the identity, the pooling is degenerate: a special case of degenerate pooling, called 1×1 convolution, is shown in Figure 1, b). The corresponding number of simple cells

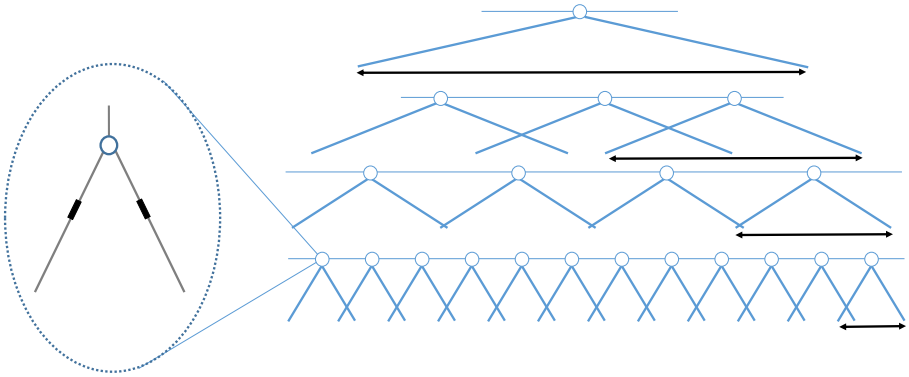

Figure 2: *A hierarchical network in which all the layers are pooling layers. The "image" is at the bottom. The final layer may consist of a small array of cells not just one.*

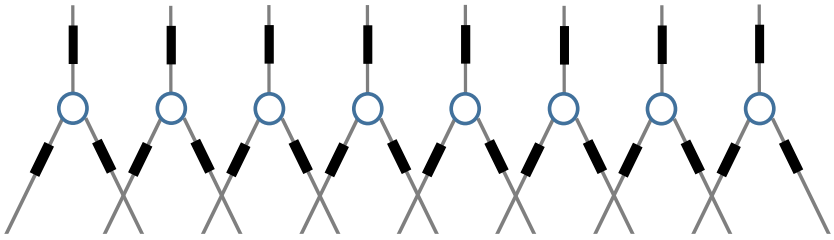

Figure 3: *Two layers of a hierarchical network: the first layer is a pooling layer, the second is not (it could be a* 1×1 *convolutional layer).*

is Q, whereas in the case of non-degenerate "convolution", the number of simple cells within a $\bigwedge$ is $|G| \times Q$, where $|G|$ is the group cardinality.

Remarks

- In the field of function approximation the rectifier nonlinearity was called ramp by Breiman [6].
- The collection of inner products of a given input with a template and its *transformations* in i-theory corresponds to a so called convolutional layer in DCLNs. More precisely, given a template t and its transformations $g_i t$, where $g_i \in G, i = 1, \cdots, |G|$ is a finite set of transformations (in DCLNs the only transformations presently used are translations), each input x is mapped to $\langle x, g_i t \rangle, \quad g_i \in G$. The values are hence processed via a non linear *activation* function, e.g. a sigmoid $(1 + e^{-s})^{-1}$, or a rectifier $|s + b|_+ = \max\{-b, s\}$ for $s, b \in \mathbb{R}$.
- Neither the max of a kernel nor the softmax of a kernel are kernels ([7]), *while the average is a kernel*. However, one may use instead of the softmax the numerator $\sum_g \langle x, gt \rangle^n$ (or $\sum_g e^{\langle x, gt \rangle^n}$) as a "proxy" for the maximum. These expressions are called ([7]) *softmaximal kernels*.
- From the point of view of an implementation in terms of networks using only sums and rectification, the following expression is interesting (see Figure 4)

$$max_2(x_1, x_2) = x_2 + |(x_1 - x_2)|_+. \tag{5}$$

Iteration of the expression provides max pooling for d inputs as

$$max(x_1, \cdots, x_d) = max_2(x_d, (max_2(x_{d-1}, (\cdots (max_2(x_2, x_1) \cdots)) \tag{6}$$

3 Main example: the sum-of-squares network

Suppose to choose $\phi(z) = (b + z)^2$ as the nonlinearity acting on the dot product $z = \langle t, x \rangle$ in each of the edges of a network such as in Figure 2. Assume, for simplicity that there is no pooling. Then the output at the second layer can be written as $\phi(\phi(x))$, or more explicitly being

$$\phi_i(x) = (\langle x, t^{i1} \rangle - b^{i1})^2$$

we can write

$$\phi_j(z) = \phi_j(\phi(x)) = (\sum_i (\langle x, t^{i1} \rangle - b^{i1})^2 t_i^{2,j} - b^{2,j})^2 \tag{7}$$

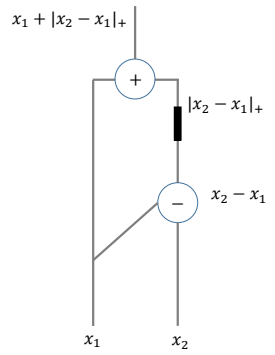

Figure 4: *Possible network module implementing of* $max(x_1, x_2)$.

where the indexes $1, 2$ refers to the first and second layer.

It is easy to show (see for instance [8] and the work in Germany in the 70's on industrial OCR, see [9]) that the second layer can synthesize an arbitrary second order polynomial – and the n^{th} layer an arbitrary $2n^{th}$ order polynomial – in the d variables that are components of x [8], provided there are a sufficient number of linear combinations with weights that can be chosen (or learned). Weierstrass theorem then ensures that the network can provide an arbitrary good approximation of a continuous function of d variables on an interval (see also [10, 8]). Each unit at layer i can represent a specific quadratic polynomial in the m variables of the previous layer. A polynomial of degree $2d$ represented by a hierarchical network of depth d with m variables requires a flat, one-layer representation with $\frac{(2d+m-1)!}{(m-1)!}$ units, each representing a monomial of the polynomial. The complexity of a shallow networks representing a specific small complexity hierarchical polynomial is thus much larger. Similar observations with much more detail and with formal proofs can be found in an elegant paper by Shalev-Shwartz [11]. Quadratic networks are very closely related to sum-product nets studied by Bengio [12] who also reports the observation that specific deep sum-product networks in general require equivalent shallow networks with an exponential number of units.
The kernel associated with ϕ, $K(x, y) = (1 + \langle x, y \rangle)^2$ is a kernel and

$$K(x, y) = |\langle t, x - y \rangle|^2 = |\langle t, x \rangle - b|^2$$

is also a kernel (see [13]). The reason that we use it as our main example is that it is a a good proxy for several other hierarchical kernels, in particular for the absolute value kernel (think about the similar "shape" of x^2 and $|x|$) that we discuss in the next section. To be precise: $\sum_i c_{ij} |\langle t^j, x \rangle - t_i^j|$ replaces $(\langle t^j, x \rangle - b_i^j)^2$ (where with subscripts we indicate elements of a vector and with superscripts different vectors). Consider the particular "pyramidal" architecture of Figure 5 with the square kernel, no convolution, 4 input variables x_1, x_2, x_3, x_4 at the bottom, 3 layers, no nonlinearity in the first layer (for simplicity): this

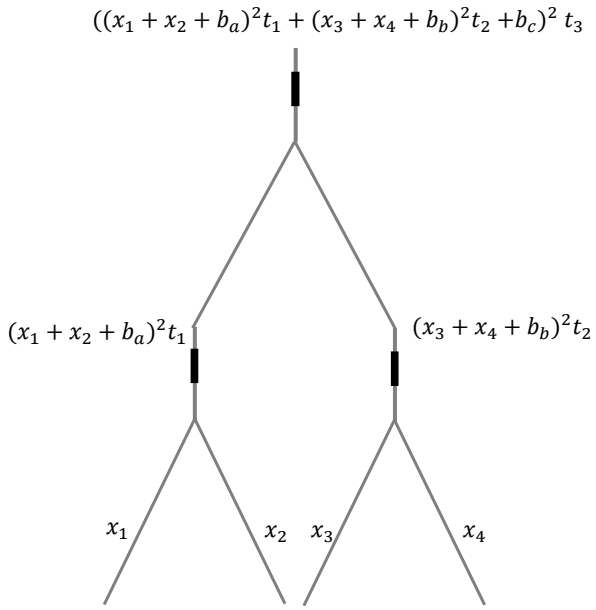

Figure 5: *A sum-of-squares network.*

leads to hierarchical terms of the type

$$t_3(b_c + t_2(b_a + x_1 + x_2)^2 + t_1(b_b + x_3 + x_4)^2)^2.$$

Note that this network corresponds to fewer units than a generic, flat implementation of the same degree i (a shallow network with a unit for each different monomial). This is similar in spirit – but different in from – to the compact calculations of a polynomial kernel that computes $K(x, z) = \langle z, x \rangle^2$ without the need of explicitly representing all the (quadratic) monomials.

Remarks

- Notice that like in all deep networks the quadratic network deals in each layer with approximation of functions in one variable (the linear combination of the previous variables in the inputs). In spirit, this is the main motivation for additive splines (see Appendix 8.2).
- The R-convolution of Haussler (see Appendix 8.2) use kernels that are linear combinations of tensor products of possibly one dimensional kernels. They can be defined on a variety of data structure. In particular, they can be defined on the structure defined by the graph representing the connectivity of the specific deep network.

- The square kernel has the property that the composition is equivalent to the product, which is a kernel (this is strictly true for $K(x,y) = \langle x,y\rangle^2$; for polynomials the composition is is still a polynomial and a kernel). Thus the compositions associated with a deep network using the square nonlinearity are equivalent to linear combinations of (tensor) products.

- The iterated linear combination of tensor products corresponding to the network graph can be represented iterating the construction of Haussler (see Appendix 8.2) at each layer of the network:

$$K(x,y) = \sum_{\mathbf{x}\in\mathcal{R}^{-1}(x),\mathbf{y}\in\mathcal{R}^{-1}(y)} \prod_{d=1}^{D} K_d(x^d, y^d) \tag{8}$$

  where $K_d(x_d, y_d) = \langle x_d, y_d\rangle^2$ and $\mathcal{R}$ is defined by the network graph. The second layer can be obtained by observing that the corresponding kernel is the square of sum of products of kernels and thus still a kernel. This particular use the of Haussler construction works for the quadratic kernel but not for the absolute value case since the composition of absolute value kernels cannot be written as a product.

- The above may be related to the tensor rank argument (Hierarchical Tucker vs CP rank) (see [14]). Note that the Tucker decomposition of Hackbush and Kuhn [15] is inspired by the multiresolution spaces of wavelets.

- If the pooling is a max, and not a sum, then some of the properties – such as the kernel property – do not hold anymore. For the polynomial softmax kernel however most of the proofs still hold.

4 Present-day DCLNs are hierarchical, additive linear splines

This section is about the main observation of this paper. Consider an additive approximation scheme (see Appendix) in which a function of d variables is approximated by an expression such as

$$f(x) = \sum_{i}^{d} \phi_i(x^i) \tag{9}$$

where x^i is the i-th component of the input vector x and the ϕ_i are one-dimensional spline approximations. For linear piecewise splines $\phi_i(x_i) = \sum_j c_{ij}|x^i - b_i^j|$. Obviously such an approximation is not universal: for instance it cannot approximate the function $f(x,y) = xy$. The classical way to deal with the problem is to use tensor product splines (a particularly efficient special case of which is tensor product of Gaussians – if we are willing to say that a Gaussian

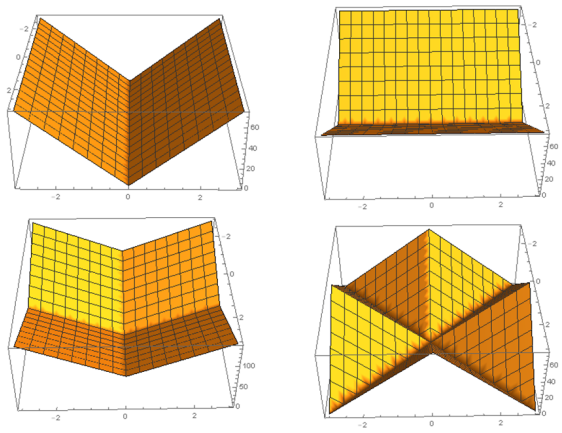

Figure 6: *A plot of the iterated absolute value function* $\phi(c_1\phi(x)) + c_2\phi(y)) = \frac{\pi^4}{4}|c_1|x| + c_2|y||$ *for* $(c_1, c_2) = (0,1), (1,0), (1,1), (1,-1)$.

is a spline). The new alternative that we propose here is *hierarchical additive splines*, which in the case of a 2-layers hierarchy has the form

$$f(x) = \sum_{j}^{K} \phi_j(\sum_{i}^{d} \phi_i(x^i)). \tag{10}$$

and which can be clearly extended to an arbitrary depth. The intuition is that in this way, it is possible to obtain approximation of a function of several variables from functions of one variable because interaction terms such as xy in a polynomial approximation of a function $f(x, y)$ can be obtained from terms such as $e^{log(x)+log(y)}$.
We start with a lemma about the relation between linear rectifiers, which do not correspond to a kernel, and absolute value, which is a kernel.

Lemma 1 *Any given superposition of linear rectifiers* $\sum_i c'_i|x - b'^i|_+$ *with* c'_i, b'^i *given, can be represented over a finite interval in terms of the absolute value kernel with appropriate weights. Thus there exist* c_i, b^i *such that* $\sum_i c'_i|x - b'^i|_+ =$

$\sum_i c_i|x - b^i|$.

The proof follows from the facts that a) the superpositions of ramps is a piecewiselinear function, b) piecewise linear functions can be represented in terms of linear splines and c) the kernel corresponding to linear splines in one dimension is the absolute value $K(x, y) = |x - y|$.

Now consider two layers in the network of Figure 2 in which we assume degenerate pooling for simplicity of the argument. Because of Lemma 1, and because weights and biases are arbitrary we assume that the the nonlinearity in each edge is the absolute value. Under this assumption, unit j in the first layer, before the non linearity, computes

$$f^j(x) = \sum_{i=1} c_i^j |\langle t^i, x\rangle - b^i|, \tag{11}$$

where x and w are vectors and the t^i are real numbers. Then the second layer output can be calculated as in eq. (7) with the nonlinearity $|\cdots|$ instead of $(\cdot)^2$.

In the case of a network with two inputs x, y the effective output after pooling at the first layer may be $\phi^{(1)}(x, y) = t_1|x + b_1| + t_2|y + b_2|$, that is the linear combination of two "absolute value" functions. At the second layer terms like $\phi^{(2)}(x, y) = |t_1|x+b_1|+t_2|y+b_2|+b_3|$ may appear, as shown in Figure 6 (where $b_1 = b_2 = b_3 = 0$). The output of a second layer still consists of hyperplanes, since the layer is a kernel machine with an output which is always a piecewise linear spline.

Networks implementing tensor product splines are universal in the sense that they approximate any continuous function in an interval, given enough units. Additive splines on linear combinations of the input variables of the form in eq. (11) are also universal (use Theorem 3.1 in [16]). However additive splines on the individual variables are not universal while hierarchical additive splines are:

Theorem *Hierarchical additive splines networks are universal (on an interval).*

Proof sketch: We use Arnold's and Kolmogorov proof of the converse of Hilbert's 13th conjecture: *a continuous function of two or more variables can be represented in terms of a two-layer network of one dimensional function.* Informally their result is as follows. Let $f : \mathbb{R}^d \to \mathbb{R}$ and $x = (x^1, \ldots, x^d) \in \mathbb{R}^d$ and consider

$$f(x) = \sum_{i=1}^{2d+1} g_i(\sum_{j=1}^{d} h_{i,j}(x^j)) \tag{12}$$

where $g_i, h_{i,j} : \mathbb{R} \to \mathbb{R}$, $\forall i, j$. The functions $h_{i,j}$ can be chosen to be univariate and are independent of f. Further refinements of the theorem show that $g_i =$

$c_i g$, for $c_i \in \mathbb{R}$. Let us use the result in the simple case of two variables:

$$f(x,y) = \sum_{i=1}^{5} g_i(h_{i,1}(x) + h_{i,2}(y)). \tag{13}$$

The functions g and $h_{i,j}$ are not "nice" but are continuous and thus, given arbitrary resources, can be approximated arbitrarily well by classical, one-dimensional, additive splines (see Sprecher implementation of Kolmogorov solution). In a no-pooling network represented by a binary graph (such as in Figure 2) , the output of a node after the nonlinearity is a function of the two inputs and can be represented by Equation 13 (this also shows that a RLU network of depth $d \geq n$ with n being the dimensionality of the input can approximate arbitrarily well any continuous function).

Remarks

- One is naturally led to the idea that networks composed of PLS subnetworks can approximate any reasonable network (sigmoidal, radial, etc.), consistently with the idea that multiple layers networks are equivalent to McCullogh-Pitts networks and to finite state machines (see [17, 18]).

- Related to the sketch of the proof of Theorem above here are a few additional comments. The Taylor series representation of $|x+t| = \sqrt{(x+t)^2} = \sqrt{(t(\frac{x}{t}+1))^2} = |t|\sqrt{(\frac{x}{t}+1)^2}$ is

$$|x+t| = |t|\sqrt{(\frac{x}{t}+1)^2} = |t|\sum_{n=0} \frac{(-1)^n(2n)!}{(1-2n)(n!)^2(4n)}(\frac{x}{t})^n \tag{14}$$

  and converges for $|\frac{x}{t}| \leq 1$. For a network of depth d the expansion above can be reused to provide a series representation of the whole network valid for a certain range of convergence depending on the parameters at each layer. Appropriate renormalization operations at each layer may ensure convergence of the representation for any input to a network containing such operations.

  The proof suggested in the main part of the section can be replaced by the following argument. Recall that one-layer subnetworks can perform piecewise linear spline approximation (PLS) of one-dimensional functions. Then construct two-layers PLS network modules that approximate in the first layer $log(x)$ function and in the second layer the e^x function. With these modules two-layers networks can represent

$$\sum_i g_i(\sum_j h_{i,j}(x^j)), \tag{15}$$

where the g_i are powers of exponential and the $h_{i,j}$ are log with appropriate coefficient, thus obtaining terms such as $e^{log(x)+log(y)}$. In this way a multilayer network can approximate a polynomial in d variables of arbitrary degree. Notice that in this proof the minimum number of required hidden layers is two, though more layers may give more efficient representations for specific functions.

- The proof of the universality of hierarchical additive linear splines requires more than one hidden layer unlike the existing proofs about universality of Gaussians and other functions (see [19] and [20]).

- Approximating with additive splines the functions $log(x)$ and e^x makes it in principle possible to extend the results of [14] from product networks to the more usual networks of one-variable functions, such as ramps, in current use.

- The equivalence of compositions with products is lost in the case of the absolute value kernel. It is thus impossible to use Haussler representation and the implied equivalence with linear combinations of tensor products.

- The equivalence can be recovered by approximating the absolute value with its Taylor representation which, when truncated, corresponds to a polynomial kernel. The approximation suggests that the behavior of the absolute value is similar to the square and that a similar argument based on HT decomposition may hold. Note however that the convergence domain depends on parameters at each layer.

- A result related to Lemma 1 follows from an integral evaluation deals with "uniform" linear combinations of ramps (see Figure 6):

  Lemma 2 *Consider the "feature"* $\phi(x) = \int_{-\pi}^{\pi} dw|(w(x-t))|_+$ *and the iteration of it* $\phi(c_1\phi(x-t)+c_2\phi(y-q))$. *The calculations provide* $\phi(x) = \frac{\pi^2}{2}|x-t|$ *and* $\phi(c_1\phi(x-t)+c_2\phi(y-q)) = \frac{\pi^4}{4}|c_1|x-t|+c_2|y-q||$. *The calculation is recursive and can be used for layers of arbitrary depth.*

- If pooling is the average and the nonlinearity (synthesized from linear rectifiers) is the absolute value then each layer of a DCLN is a kernel machine.

- Results indirectly related to the Lemma are due to Saul [21]. Also note that specific choice of the weights can synthesize the absolute value from linear rectifiers ($|\langle t,x\rangle - b| = |\langle t,x\rangle - b|_+ + |-\langle t,x\rangle - b|_+$). In addition, sigmoids and Gaussian-like one-dimensional functions can also be synthesized as linear combinations of ramps[13].

- Notice that $\phi(\phi)$ is not a kernel though the kernel that describes the similarity criterion induced by the features computed by the 2-layer network can be written as $K(x,y) = \left\langle \phi^{(2)}(x), \phi^{(2)}(y) \right\rangle$.

- All kind of nonlinearities $\phi(x)$ yield universality for networks of the form $f(x) = \sum_i^d c_i \phi_i(\langle w_i, x \rangle)$. The key condition is that the nonlinearity cannot be a polynomial [16]. Interestingly, this restriction disappears in hierarchical architectures as shown by the example of quadratic networks.

5 Why hierarchies

In i-theory the reasons for a hierarchy follow from the need to compute invariant representations. They are

1. *Optimization of local connections* and optimal reuse of computational elements. Despite the high number of synapses on each neuron it would be impossible for a complex cell to pool information across all the simple cells needed to cover an entire image, as needed by a single hidden layer network..

2. *Compositionality, wholes and parts.* A hierarchical architecture provides signatures of larger and larger patches of the image in terms of lower level signatures. Because of this, it can access memory in a way that matches naturally with the linguistic ability to describe a scene as a whole and as a hierarchy of parts.

3. *Approximate factorization.* In architectures such as the network of Figure 2, approximate invariance to transformations specific for an object class can be learned and computed in different stages. Thus the computation of invariant representations can, in some cases, be "factorized" into different steps corresponding to different transformations.

This paper adds a few additional reasons for hierarchies:

- As described in section 6 the typical convolutional architecture which looks like a hierarchical pyramid is likely to be an optimal way to approximate signals that have certain symmetry properties related to shift and scale invariance.

- Supervised learning of the coefficients of filters over channels for each input and at each stage can be regarded as supervised PCA that helps reduce dimensionality.

5.1 Exponential complexity of n-layers nets

The main argument can be inferred from the square kernel example, in the case of the pyramidal architecture of Figure 2. The result is that *multilayer representations of additive one-dimensional approximations have a representations of exponentially higher complexity if constrained to one layer.* The argument [8] is as follows

- assume that the following modules are available: square operation, linear combinations with arbitrary coefficients. Then it is possible to compute in one layer (square and linear combinations) all the individual monomial, in n variables, such as $x \cdot y$, x^2...

- each of the monomial can be represented by a unit which can be weighted appropriately in the next layer in order to approximate an arbitrary function

For the absolute value kernel, a very similar argument can be used for the Taylor representation of the iterated network. Related results are Bengio's [22] bounds on the number of linear regions that a d layer network can generate relative to a one layer network with the same number of units. A more powerful approach, because it allows the use of classical results about sample complexity, generalization error etc) is to characterize the capacity of such multilayer networks in terms of classical measures such as VC-dimension, Radamacher averages and Gaussian averages [23, 24, 11]. For the hierarchical quadratic networks described in [11] (see section 4 there and also section 3 in this paper) a coarse VC-dimension bound (assuming binary output for the network) is $O(\gamma(\Delta + d))$ where γ is the number units per layer, Δ is the degree of the polynomial, d is the dimensionality of the input space $\mathbb{R}^d$, whereas the VC-dimension of a one layer is $\frac{(d+\Delta)!}{\Delta! d!}$ which grows much more quickly with d and Δ. Also relevant here is the work on SimNets [25] and related tensor analysis of their complexity[14].

It is amusing to notice that the above complexity observations apply to (specific implementations of) the networks with linear rectifiers discussed in this note because *hierarchical additive splines networks can approximate multipliers, squares, MEX (and other functions) at a complexity cost in terms of depth and number of units which is usually a (small) multiplicative constant.*

6 Theory and DCLNs: a summary

The body of previous work that we called i-theory is studying representations for new images (not previously seen) that are selective and invariant to transformations previously experienced (for different objects). The theory applies to the HW modules in Figure 1 and to the hierarchical architecture of Figure 2. The theory suggests directly such an architecture as a natural alternative to the single HW module, which we indicate with $\bigwedge$, for the computation of invariance. In particular i-theory can be used to characterize properties of convolutional/pooling layers in DCLNs. The output of the basic HW module, corresponding to complex cell k, is

$$\mu^k(I) = h(\frac{1}{|G|}\sum_{g\in G}\eta(\langle I, gt^k\rangle)), \quad k = 1, ..., K, \tag{16}$$

where h is a monotonic nonlinear function (often the identity). In today's DCLNs $\mu^k(I) = (\frac{1}{|G|}\sum_{g\in G} |\langle I, gt^k\rangle|_+^p)^{\frac{1}{p}}$, which is close to the max for large p.

A similar output which we will consider in our analisys for the pooling layers is $\mu^k(I) = \frac{1}{|G|}\sum_{g\in G} |\langle I, gt^k\rangle|_+$ i-theory shows that μ^k is invariant and *selective* as much as desired depending on K *if* the nonlinearity η is appropriate and if h is monotonic. The theory was developed for the *unsupervised* case but it applies to the convolutional layers of DCLNs because of the hardwired convolution there (the group G implicit in DCLNs is the translation group in x, y). As a side note, it suggests a possible extension to scale of the convolutional layers of DCLNs. The previous invariance and selectivity results of i-theory also apply to the 1×1 convolutional layers in terms of selectivity (there is no transformation, no pooling, no invariance) but without any useful insight. The invariance and selectivity results can be used for the pooling layers in mixed networks such as in Figure 3 but not for the nonpooling ones. More importantly, networks of the type shown in Figure 2 are outside the scope of the theorems proved in earlier i-theory papers (see Appendix 8.1.1).

The approach in this paper applies to supervised networks without pooling, such as the case of Figure 2. Together with the previous invariance results, this extended i-theory applies to supervised networks with mixed pooling and non-pooling layers.

Thus this paper extends i-theory to supervised and non-pooling networks. This extended theory can be applied to the current DCLNs architectures. It also suggests other similar networks and several variations of them. For instance, weight sharing does not need to be over the whole image: it can be restricted to the pooling regions. The nonlinearity allowed are rather general but must yield universal approximations (like ramps or cubic splines for hierarchical splines and ramps for one-step, non-hierarchical approximations, ideally, of a one-dimensional pdf via the group average). An obvious alternative choice is a Gaussian function instead of a ramp: the corresponding architecture is a hierarchical Gaussian RBF network. The architecture of Figure 2, similar to a binary tree, is almost implied by convolution, pooling and subsampling which are a special case of it. It corresponds to a particular decomposition of the computations represented by a function of several variables – in functions of functions of subsets – which is optimal for signals that have certain symmetry properties reflecting shift and scale invariance. A forthcoming paper will explore the reasons for the claim that much of *the power of the architecture of Figure 2 derives from the particular hierarchical combination of inputs variables and is rather independent of the details of the nonlinear operations (whether linear rectifiers or sigmoids or Gaussians).*

Remarks

- The two main contributions of the original i-theory, before the extensions of this paper, are:
  - theorems on invariance and selectivity of pooling for transformations belonging to a group and related results on approximative invariance for non-group transformations

– "unsupervised" learning of invariance to transformations by memorizing transformations of templates either directly or in the form of PCs, because $\langle x, gt \rangle = \langle g^{-1}x, t \rangle$.

The second contribution is potentially quite relevant especially for neuroscience.

- Consider the invariant

$$\mu^k(I) = h(\frac{1}{|G|} \sum_{g \in G} \eta(\langle I, gt^k \rangle)), \quad k = 1, ..., K, \tag{17}$$

The property that μ^k is *invariant and selective if* h is monotonic is a direct extension of Theorem 5 in [2].

6.1 Biological implications

There are several properties that follow from the theory here which are attractive from the point of view of neuroscience. A main one is the robustness of the results with respect to the choice of nonlinearities (linear rectifiers, sigmoids, Gaussians etc.) and pooling (to yield either moments or pdf or equivalent signatures).

An somewhat puzzling question arises in the context of neuroscience plausibility about weight-sharing. A biological learning rule that enforces weight-sharing across the visual field seems implausible. In the context of i-theory a plausible alternative is to consider the problem of weight sharing only within a complex cell receptive field. During development within the receptive field of each complex cell the simple cells tuning may be due to Hebb-like plasticity. There are then two possibilities: a) after development the simple cells tuning is refined by supervised SGD in a non-shared way or b) after developments the tuning of the simple cells does change but the weight vector of the complex cells outputs at one position is tuned by SGD.

7 Discussion

The main observation of this note is that present-day DCLNs can be regarded as *hierarchical additive splines*. This point of view establishes a connection with classical approximation theory and may thereby open the door for additional formal results and for extensions of the basic network architecture.

Connection with i-theory. Loosely speaking i-theory characterizes invariance and associated selectivity obtained by convolutional layers. It explains how to extend the convolutional stage to other transformations beyond translation. I-theory suggests why hierarchies are desirable for stage-by-stage invariance. The non-convolutional layers may be characterized by formal results within the framework introduced here – of hierarchical additive splines.

Tacit assumption: stochastic gradient descent works. Throughout this note, we discuss the potential properties of multilayer networks, that is the properties they have with the "appropriate" sets of weights. The assumption we make is therefore that training using greedy SGD on very large sets of labeled data, can find the "appropriate sets of weights". Characterizing the reasons of the success of SGD in training DCLNs and its generalization properties is probably the second main problem, together with the problem of the properties of the architecture which is the subject of this note.

A model of universal neural network computation. In the traditional "digital" model of computation, universality means the capability of running any program that is computable by a Turing machine. Concepts such as polynomial vs non-polynomial complexity emerge from this point of view. For neural networks the model of computations that seems to take form now is closely related to function approximation. Here universality may be defined in terms of universal approximation of a function of n variables (the inputs). Appropriate definitions of complexity may need refinements beyond existing constructs such as stability and covering numbers to include networks depth, number of units and local connectivity (eg size of pooling regions).

8 Appendices

8.1 Kernels and features

Any set of features such as

$$\phi_{t,b}(x) = |\langle x, gt\rangle + b|_+$$

is associated to the kernel

$$K_0(x,x') = \phi^T(x)\phi(x') = \int db\, dt\, |\langle t, x\rangle + b|_+ |\langle t, x'\rangle + b|_+ = \sum_{t,b} \phi^T(x)\phi(x')$$

where the $\int$ reduces to the finite sum $\sum_{t,b} \phi^T(x)\phi(x')$ for finite networks. If the sum is finite, the kernel is always defined; otherwise its existence depends on the convergence of the sum or the existence of the integral.

Features and kernels, when they are well defined, are equivalent. One has to be careful, however, about what this means, especially in the case of multilayer networks.

Suppose the output of a network is a set of features $\Phi_i(x)$ for each input x. This the common situation with a DCLN. A kernel that measure similarities between the input x and a stored y can be defined using the set of features $\sum \Phi_i(x)\Phi_i(y) = K(x,y)$ *irrespectively of how the features were computed.* Alternatively the network could already provide as an output the similarity $K(x,y)$ between input x and a stored y.

Remark
We briefly discuss here the relation between units in a network, features and kernels. We begin by noticing that, because of the generalization to virtual centers (not corresponding to data point), there is quite a bit of flexibility in how to interpret a network. What I mean is the following. In the classical definition a shift invariant kernel such as the absolute value $K(x,y) = |x-y|$ corresponds to features that are Fourier components. In the example of linear rectifiers we can interpret $|\langle t, x\rangle + b|_+$ as a feature (for a specific t, b). However I can also consider the linear combination $|\langle t, x\rangle + y|_+ + |-\langle t, x\rangle - y|_+| = |x-y|$ as a virtual unit computing the kernel similarity computation for the absolute value.

8.1.1 Invariance, convolution and selectivity

In the following we summarize the key steps of [26] in proving that HW modules are kernel machines:

1. The feature map
$$\phi(x,t,b) = |\langle t, x\rangle + b|_+$$
(that can be associated to the output of a simple cell, or the basic computational unit of a deep learning architecture) can also be seen as a kernel in itself. The kernel can be a universal kernel. The feature ϕ leads to a kernel
$$K_0(x,x') = \phi^T(x)\phi(x') = \int db\, dt\, |\langle t, x\rangle + b|_+ |\langle t, x'\rangle + b|_+$$
which is a universal kernel being a kernel mean embedding (w.r.t. t, b, see [27]) of the a product of universal kernels.

2. If we explicitly introduce a group of transformations acting on the feature map input i.e. $\phi(x,g,t,b) = |\langle gt, x\rangle + b|_+$ the associated kernel can be written as
$$\begin{aligned}\tilde{K}(x,x') &= \int dg\, dg' \int dt\, db\, |\langle gt, x\rangle + b|_+ |\langle g't, x'\rangle + b|_+ \\ &= \int dg\, dg'\, K_0(x,g,g').\end{aligned}$$
$\tilde{K}(x,x')$ is the group average of K_0 (see [28]) and can be seen as the mean kernel embedding of K_0 (w.r.t. g, g', see [27]).

3. The kernel $\tilde{K}$ is invariant and, if G is compact, selective i.e.
$$\tilde{K}(x,x') = 1 \;\Leftrightarrow\; x \sim x'.$$
The invariance follows from the fact that any $G-$group average function is invariant to G transformations. Selectivity follows from the fact that $\tilde{K}$ a universal kernel being a kernel mean embedding of K_0 which is assumed to be universal (see [27]).

Remark 1. *If the distribution of the templates t follows a gaussian law the kernel K_0, with an opportune change of variable, can be seen as a particular case of the nth order arc-cosine kernel in [21] for $n = 1$.*

8.2 Additive and Tensor Product Splines

Additive and tensor product splines are two alternatives to radial kernels for multidimensional function approximation. It is well known that the three techniques follow from classical Tikhonov regularization and correspond to one-hidden layer networks with either the square loss or the SVM loss.

I recall the extension of classical splines approximation techniques to multidimensional functions. The setup is due to Jones et al. (1995) [13].

8.2.1 Tensor product splines

The best-known multivariate extension of one-dimensional splines is based on the use of radial kernels such as the Gaussian or the multiquadric radial basis function. An alternative to choosing a radial function is a *tensor product* type of basis function, that is a function of the form

$$K(x) = \Pi_{j=1}^{d} k(x_j)$$

where x_j is the j-th coordinate of the vector x and $k(x)$ is the inverse Fourier transform associated with a Tikhonov stabilizer (see [13]).
We notice that the choice of the Gaussian basis function for $k(x)$ leads to a Gaussian radial approximation scheme with $K(x) = e^{-\|x\|^2}$.

8.2.2 Additive splines

Additive approximation schemes can also be derived in the framework of regularization theory. With additive approximation we mean an approximation of the form

$$f(x) = \sum_{\mu=1}^{d} f_\mu(x^\mu) \tag{18}$$

where x^μ is the μ-th component of the input vector x and the f_μ are one-dimensional functions that will be defined as the *additive components* of f (from now on Greek letter indices will be used in association with components of the input vectors). Additive models are well known in statistics (at least since Stone, 1985) and can be considered as a generalization of linear models. They are appealing because, being essentially a superposition of one-dimensional functions, they have a low complexity, and they share with linear models the feature that the effects of the different variables can be examined separately. The resulting scheme is very similar to Projection Pursuit Regression. We refer to [13] for references and discussion of how such approximations follow from regularization.

Girosi et al. [13] derive an approximation scheme of the form (with i corresponding to spline knots – which are free parameters found during learning as in free knots splines - and μ corresponding to new variables as linear combinations of the original components of x):

$$f(x) = \sum_{\mu=1}^{d'} \sum_{i=1}^{n} c_i^\mu K(\langle t^\mu, x \rangle - b^\mu) = \sum_{\mu=1} \sum_{i=1} c_i^\mu K(\langle t^\mu, x \rangle - b_i^\mu) \ . \qquad (19)$$

Note that the above can be called spline only with a stretch of the imagination: not only the w_μ but also the t^μ depend on the data in a very nonlinear way. In particular, the t^μ may not correspond at all to actual data point. The approximation could be called ridge approximation and is related to projection pursuit. When the basis function K is the absolute value that is $K(x-y) = |x-y|$ the network implements piecewise linear splines.

8.3 Networks and R-Convolution Kernels

Suppose that $x \in X$ is an image (more in general a signal) composed of patches $x^1, \cdots, x^D$ where $x^d \in X^d$ for $1 \leq d \leq D$. Following Haussler we assume that $X, X^1, \cdots, X^D$ are separable metric spaces and in particular that they are countable finite sets, such as pixels. The patches are implicitly defined by the architecture of the first layer of the networks we will discuss (see Fig 2). In particular the patches of the image are defined by the receptive fields of the first layer complex cells.

We represent the property "$x^1, \cdots, x^D$ are patches of x" by a relation $\mathcal{R}$ on the set $X \times X^1, \cdots, X^D$ where $\mathcal{R}(x^1, \cdots, x^D, x)$ is true if $x^1, \cdots, x^D$ are parts of x^d. We denote $x = (x^1, \cdots, x^D)$ and $\mathcal{R}(x, x) = \mathcal{R}(x^1, \cdots, x^D, x)$. Let $\mathcal{R}^{-1}(x) = x : \mathcal{R}(\S, \S)$.

Our main example for $\mathcal{R}$ consists of the following: $x^1, \cdots, x^D$ are an ordered set of square patches with n^2 pixels composing without overlap the image x which has $N^2 = Dn^2$ pixels. This relation can be iterated composing larger parts defined by the higher layers of the network of Figure 2. Suppose $x, y \in X$ and for the decomposition we have defined $\mathbf{x} = x^1, \cdots, x^D$ are the patches of x and $\mathbf{y} = y^1, \cdots, y^D$ are the corresponding patches of y. Suppose now that for each $1 \leq d \leq D$ there is a kernel $K_d(x^d, y^d)$ between the patch x^d and the patch y^d. We then define the kernel K measuring the similarity between x and y as the generalized convolution

$$K(x, y) = \sum_{\mathbf{x} \in \mathcal{R}^{-1}(x), \mathbf{y} \in \mathcal{R}^{-1}(y)} \prod_{d=1}^{D} K_d(x^d, y^d) \qquad (20)$$

This defines a symmetric function on $S \times S$ where $S = x : \mathcal{R}^{-1}(x)$ is not empty. The $\mathcal{R}$-*convolution* of $K_1, \cdots, K_D$ is defined as the zero-extension of K to $X \times X$. The zero-extension of K from $S \times S$ to $X \times X$ is obtained by setting $K(x, y) = 0$ either x or y is not in S. The following theorem holds:
Theorem If $K_1, \cdots, K_D$ are kernels on $X^1 \times X^1 \times, \cdots, X^D \times X^D$, respectively,

and $\mathcal{R}$ is a finite relation on $X \times X^1, \cdots, X^D$ then K defined in Equation 8 is a kernel on $X \times X$.

8.4 SimNets and Mex

Mex is a generalization of the pooling function. From [25] eq. 1 it is defined as:

$$Mex_{(\{c_i\},\xi)} = \frac{1}{\xi} \log \Big(\frac{1}{n} \sum_{i=1}^{n} \exp(\xi c_i) \Big)$$

We have

$$Mex_{(\{c_i\},\xi)} \xrightarrow[\xi \to \infty]{} Max_i(c_i)$$
$$Mex_{(\{c_i\},\xi)} \xrightarrow[\xi \to 0]{} Meani(c_i)$$
$$Mex_{(\{c_i\},\xi)} \xrightarrow[\xi \to -\infty]{} Min_i(c_i).$$

We can also choose values of ξ in between the ones above, the interpretation is less obvious.

Acknowledgment

This work was supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF 1231216.

References

[1] F. Anselmi, J. Z. Leibo, L. Rosasco, J. Mutch, A. Tacchetti, and T. Poggio, "Unsupervised learning of invariant representations," *Theoretical Computer Science*, 2015.

[2] F. Anselmi, L. Rosasco, and T. Poggio, "On invariance and selectivity in representation learning," *arXiv:1503.05938 and CBMM memo n 29*, 2015.

[3] D. Hubel and T. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, p. 106, 1962.

[4] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition," *Nature Neuroscience*, vol. 3,11, 2000.

[5] M. Kouh and T. Poggio, "A canonical neural circuit for cortical nonlinear operations," *Neural computation*, vol. 20, no. 6, pp. 1427–1451, 2008.

[6] L. Breiman, "Hinging hyperplanes for regression, classification, and function approximation," Tech. Rep. 324, Department of Statistics University of California Berkeley, California 94720, 1991.

[7] T. Gaertner, *KERNELS FOR STRUCTURED DATA*. Worls Scientific, 2008.

[8] B. B. Moore and T. Poggio, "Representations properties of multilayer feedforward networks," *Abstracts of the First annual INNS meeting*, vol. 320, p. 502, 1998.

[9] J. Schuermann, N. Bartneck, T. Bayer, J. Franke, E. Mandler, and M. Oberlaender, "Document analysis from pixels to contents," *Proceedings of the IEEE*, vol. 80, pp. 1101–1119, 1992.

[10] T. Poggio, "On optimal nonlinear associative recall," *Biological Cybernetics*, vol. 19, pp. 201–209, 1975.

[11] R. Livni, S. Shalev-Shwartz, and O. Shamir, "A provably efficient algorithm for training deep networks," *CoRR*, vol. abs/1304.7045, 2013.

[12] O. Delalleau and Y. Bengio, "Shallow vs. deep sum-product networks," in *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pp. 666–674, 2011.

[13] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural Computation*, vol. 7, pp. 219–269, 1995.

[14] N. Cohen, O. Sharir, and A. Shashua, "On the expressive power of deep learning: a tensor analysis," *CoRR*, vol. abs/1509.0500, 2015.

[15] H. W. and S. Kuhn, "A new scheme for the tensor representation," *J. Fourier Anal. Appl.*, vol. 15, no. 5, pp. 706–722, 2009.

[16] A. Pinkus, "Approximation theory of the mlp model in neural networks," *Acta Numerica*, pp. 143–195, 1999.

[17] T. Poggio, "What if...," *CBMM Views and Reviews*, 2015.

[18] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge eBooks, 2014.

[19] F. Girosi and T. Poggio, "Networks and the best approximation property," *Biological Cybernetics*, vol. 63, pp. 169–176, 1990.

[20] G. Cybenko, "Approximation by a superpositions of a sigmoidal function," *Math. Control Signal System*, vol. 2, pp. 303–314, 1989.

[21] Y. Cho and L. K. Saul, "Kernel methods for deep learning," in *Advances in Neural Information Processing Systems 22* (Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, eds.), pp. 342–350, Curran Associates, Inc., 2009.

[22] R. Montufar, G. F.and Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," *Advances in Neural Information Processing Systems*, vol. 27, pp. 2924–2932, 2014.

[23] M. Anthony and P. Bartlett, *Neural Network Learning - Theoretical Foundations*. Cambridge University Press, 2002.

[24] A. Maurer, "A chain rule for the expected suprema of gaussian processes," in *ALT*, 2014.

[25] N. Cohen and A. Shashua, "Simnets: A generalization of convolutional networks," *CoRR*, vol. abs/1410.0781, 2014.

[26] L. Rosasco and T. Poggio, "Convolutional layers build invariant and selective reproducing kernels," *CBMM Memo, in preparation*, 2015.

[27] B. K. Sriperumbudur, A. Gretton, K. Fukumizu, B. Schölkopf, and G. R. G. Lanckriet, "Hilbert space embeddings and metrics on probability measures," *Journal of Machine Learning Research*, vol. 11, pp. 1517–1561, 2010.

[28] B. Haasdonk and H. Burkhardt, "Invariant kernel functions for pattern analysis and machine learning," *Mach. Learn.*, vol. 68, pp. 35–61, July 2007.