# On the Challenge of Learning Complex Functions

Yoshua Bengio
Dept. IRO, Université de Montréal
P.O. Box 6128, Downtown Branch, Montreal, H3C 3J7, Qc, Canada
{bengioy}@iro.umontreal.ca

**Abstract**

A common goal of computational neuroscience and of artificial intelligence research based on statistical learning algorithms is the discovery and understanding of computational principles that could explain what we consider adaptive intelligence, in animals as well as in machines. This chapter focuses on what is required for the learning of complex behaviors. We believe it involves the learning of highly varying functions, in a mathematical sense. We bring forward two types of arguments which convey the message that many currently popular machine learning approaches to learning flexible functions have fundamental limitations that render them inappropriate for learning highly varying functions. The first issue concerns the representation of such functions with what we call shallow model architectures. We discuss limitations of *shallow architectures*, such as so-called kernel machines, boosting algorithms, and one-hidden-layer artificial neural networks. The second issue is more focused and concerns kernel machines with a *local kernel* (the type used most often in practice), that act like a collection of template matching units. We present mathematical results on such computational architectures showing that they have a limitation similar to those already proved for older non-parametric methods, and connected to the so-called curse of dimensionality. Though it has long been believed that efficient learning in deep architectures is difficult, recently proposed computational principles for learning in deep architectures may offer a breakthrough.

## 1 Introduction

Much research in Artificial Intelligence (AI) and in computational neuroscience has focused on *how to perform a "function"*[1]. This is tedious work that is being done in both AI and computational neuroscience because of the very large number of tasks, sub-tasks, and concepts that need to be considered to explain the rich array of behaviors that are observed or desired. Research in learning algorithms started from the premise that much more robust and adaptive behaviors would result if we focused on how to *learn "function"*, i.e., the development of procedures that apply more general-purpose knowledge (how to learn to perform a task) to the specific examples encountered by

[1] e.g., object recognition in vision, motor control for grasping, etc.: we quote "function" to talk about its biological sense, and otherwise use the word in its mathematical sense.

the machine or the animal in order to yield behaviors tuned to the specifics of the environment. However, as argued here, we believe that something important has been missing from most accounts of how brains or machines learn. Expressing complex behaviors requires highly varying mathematical functions, or high-level abstractions, i.e., mathematical functions that are highly non-linear, in complex ways, in terms of the raw sensory inputs of the machine or animal. Consider for example the task of interpreting an input image such as the one in Figure 1. A high-level abstraction such as the ones illustrated in that figure has the property that it corresponds to a very large set of possible raw inputs, which may be very different from each other from the point of view of simple Euclidean distance in pixel space. Hence that set (e.g., the set of images to which the label "man" could be attributed) forms a highly convoluted region in pixel space, which we call a *manifold*.

The raw input to the learning system is a high dimensional entity, made of many observed variables, related by unknown intricate statistical relationships. Over the years, research in the field of statistical machine learning has revealed fundamental principles for learning predictive models from data, in addition to a plethora of useful machine learning algorithms (Duda, Hart, & Stork, 2001; Hastie, Tibshirani, & Friedman, 2001; Bishop, 2006). The last quarter century has given us a set of flexible (i.e., non-parametric) statistical learning algorithms that can, at least in principle, learn any continuous input-output mapping, if provided with enough computing resources and training data. In practice, traditional machine learning and statistics research has focused on relatively simple problems (in comparison to full-fledged AI). In these simple cases, the data distribution was explicitly *or implicitly* assumed to have a rather simple form: it was either assumed known ahead of time up to a few parameters, or smooth, or to involve only a few interactions between variables.

One long-term goal of machine learning research is to produce methods that are applicable to highly complex tasks, such a perception, reasoning, intelligent control, and other artificially intelligent behaviors. However, despite impressive progress on both the academic and technological sides, these long-term goals remain elusive. What makes these learning tasks challenging and difficult for the computer and presumably for animals as well is that not enough is known ahead of time about the generating distribution (or process) from which the data come. There are currently two major modes of performing statistical machine learning: one using rich explicit prior knowledge about the generating process (such as using so-called probabilistic graphical models (Jordan, 1998), Bayes nets, etc.); and another in which prior knowledge is implicit in the choice of a metric or similarity function that compares examples (e.g., kernel-based models – see `www.kernel-machines.org` – and other non-parametric models such as artificial neural networks (Rumelhart, Hinton, & Williams, 1986) and boosting (Freund & Schapire, 1997)). Non-parametric models also implicitly assume a notion of smoothness of the function to be learned, i.e., if $x$ is close to $y$ then we favor a function $f$ such that $f(x)$ is close to $f(y)$. These principles work quite well and have been employed in sophisticated ways in the last few years, but they also have limits. This chapter aims at studying them in order to orient future research towards learning algorithms that can plausibly learn the kind of intelligent behaviors that animals and humans can learn.

An example of a complex task in which the smoothness prior is insufficient is the

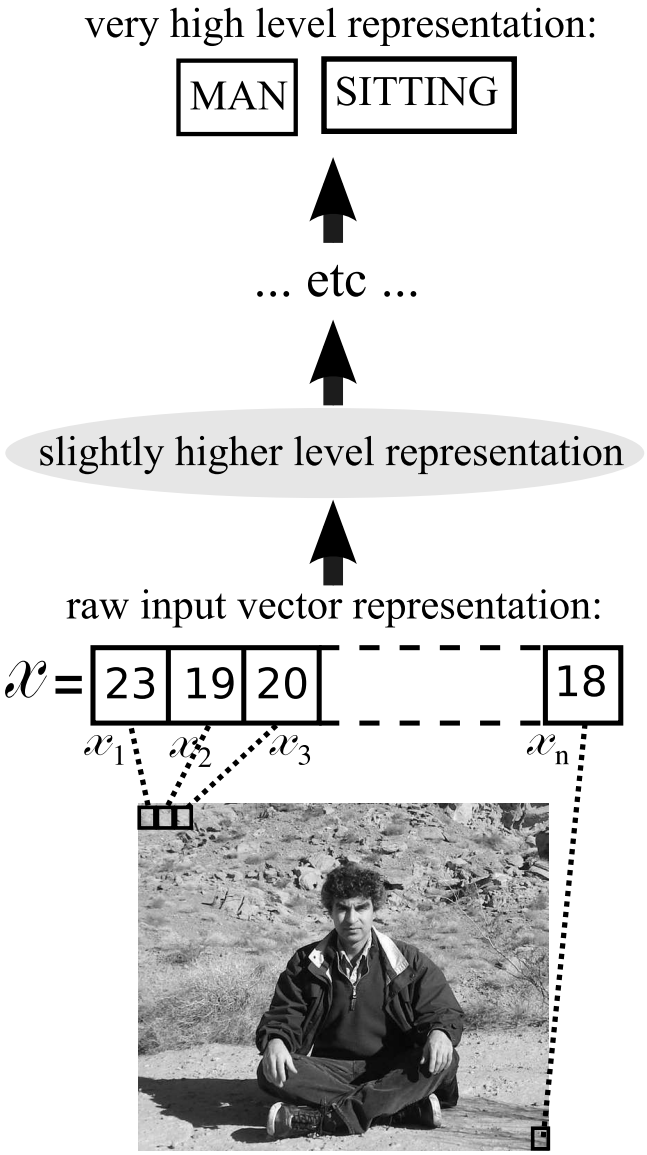


Figure 1: The raw input image is transformed into gradually higher levels of representation, representing more and more abstract functions of the raw input. In practice, we do not know ahead of time what the "right" representation (in the AI or in the biological senses) should be for all these levels of abstractions, although linguistic concepts sometimes help us to imagine what the higher levels might implicitly represent. We need learning algorithms that can discover most of these abstractions, from mostly unlabeled data.

recognition of multiple objects in an image, when each of the objects in the image can come in many variations of shape, geometry (location, scale, angle), and when backgrounds can also vary. This task involves a number of factors that can co-vary, creating a combinatorial explosion of possible variations. This setting involves a highly varying functions in the sense that if we were to parametrize all the instances of a particular object, we would find that many pixel values would oscillate between the same colors as we, for example, translated the object from left to right across the scene. This setting has been studied in Bengio and Le Cun (2007), which draws many comparisons between different learning algorithms with deep vs shallow architectures, and finds that shallow architectures are fundamentally limited in their capacity to learn efficient representations of this sort of function.

Most of the current non-parametric statistical learning techniques, studied by AI researchers and computational neuroscientists, rely almost exclusively, implicitly or explicitly, on the smoothness prior. We discuss two fundamental limitations of these computational architectures. The first limitation, discussed in section 2, is more general and concerns the representation of a function with an architecture of one or two levels of adaptive elements (elements which one can think of as neurons, or groups of neurons, and that we sometimes call units, here). We call such architectures *shallow*. We give several examples suggesting that such architectures can be very inefficient, in the sense that many more units are required to represent some functions than in a *deep architecture*. An example of a deep architecture is that of a multi-layer neural network with many layers (e.g., 5 to 10 layers). The second limitation is more specific, and concerns architectures of *kernel machines*. Most learning algorithms for kernel machines compute linear combinations of the outputs of template matchers, units that produce a large output [activation] when the raw input matches a specific template associated with the unit, and a small output otherwise. The mathematical form of the Gaussian kernel permits the theoretical analysis summarized in section 3, which gives us insight into the limitations of more general forms of template matcher, which we call local kernels. This second limitation might also plague feed-forward artificial neural networks with one hidden layer, which become kernel machines in the mathematical limit as the number of neurons becomes very large (Bengio, Le Roux, Vincent, Delalleau, & Marcotte, 2006).

These limitations both lead us to the same conclusion: in order to learn and represent highly-varying functions (such as those we believe are required in the computations involved with complex behaviors) with a shallow architecture, one would need a very large number of units. This number may even grow exponentially with the number of factors influencing the variations that can occur in the sensory inputs. With a shallow architecture, the number of examples required to learn a task to a given degree of accuracy would also be very large (in contradiction with what is observed with animals and humans) with the consequence that it would be impractical to implement an artificial intelligence that can learn truly complex behaviours.

One way around the limitations of shallow or local architectures is to embed a lot of prior knowledge in the architecture. For example, if the hard-wired form of the first layer of units in such an architecture already computes an appropriate representation for the task at hand, then the number of units required may remain small: in the right space, any learning problem becomes easy. However, such hard-wired non-linear trans-

formations would have to be *designed* by human engineers in the case of machines, and evolved in the case of biological brains. It would seem much more efficient, for human designers, as well as for evolution, to take advantage of broad priors about a large set of tasks, such as those that humans solve easily. Therefore, we set our goals towards learning algorithms that do not require very detailed priors on each task, but yet, can learn the kind of complex functions required for intelligent behavior.

In section 5, we put the requirement for deep architectures in a broader context: we present a number of computational principles which we believe should be present, either in an attempt to explain biological learning of complex behaviors or in an attempt to achieve artificial intelligence through machine learning. These include: a deep architecture, on-line learning (not having to store all examples and return to them many times), semi-supervised learning (dealing with mostly unlabeled examples), multi-task learning (capitalizing on the common processing involved in a large number of tasks), reinforcement learning (learning using reinforcement signals rather than supervised signals, which may be delayed in time), and probably active learning (choosing actions to acquire more information about the data-generating process) as well.

## 2 The Problem with Shallow Architectures

Here we consider a limitation of the general class of architectures that are *shallow*, i.e., have one or two levels of adaptive elements. In addition to kernel machines, this includes ordinary feed-forward artificial neural networks with one hidden layer.

Any specific function can be implemented by a suitably designed shallow architecture or by a deep architecture. Furthermore, when parameterizing a family of functions, we have the choice between shallow or deep architectures. The important questions are: 1. how large is the corresponding architecture (with how many parameters, how many neurons, how much computation to compute the output); 2. how much manual labor / evolutionary time is required in specializing the architecture to the task.

Using a number of examples, we shall demonstrate that deep architectures are often more efficient (more compact) for representing many functions. Let us first consider the task of adding two $N$-bit binary numbers. The most natural digital circuit for doing so involves adding the bits pair by pair and propagating the carry. The carry propagation takes a number of steps and circuit elements proportional to $N$. Hence a natural architecture for binary addition is a deep one, with $N$ layers and on the order of $N$ elements in total. It is also possible to implement it efficiently with a less deep circuit, with $\log N$ layers, with less than $N$ elements per layer, for a total of about $N \log N$ computations and elements. On the other hand, a shallow architecture with two layers can implement any boolean formula expressed in disjunctive normal form (DNF), by computing the min-terms (applying AND functions on the inputs) in the first layer, and applying an OR on the second layer. Unfortunately, even for primitive boolean operations such as binary addition and multiplication, the number of terms in the intermediate layer can be extremely large (up to $2^N$ for $N$-bit inputs in the worst case). In fact, most functions in the set of all possible boolean functions of $N$ bits require an exponential number of min-terms (i.e., an exponentially large number of basis functions) (Bengio & Le Cun, 2007). Only an exponentially small fraction of

possible boolean functions require a less than exponential number of min-terms. The computer industry has devoted a considerable amount of effort to optimize the implementation of 2-level boolean functions with an exponential number of terms, but the largest it can put on a single chip has only about 32 input bits (a 4 Gbit RAM chip). This is why practical digital circuits, e.g., for adding or multiplying two numbers are built with deep circuits, i.e., with multiple layers of logic gates: their 2-layer implementation would be prohibitively expensive. There are many theoretical results from circuits complexity analysis which clearly indicate that circuits with a small number of layers can be extremely inefficient at representing functions that can otherwise be represented compactly with a deep circuit (Hastad, 1987; Allender, 1996). See (Utgoff & Stracuzzi, 2002; Bengio & Le Cun, 2007) for discussions of this question in the context of learning architectures.

Another interesting example is the boolean parity function. The $N$-bit boolean parity function can be implemented in at least these three ways:

1. with $N$ daisy-chained XOR gates (an $N$-layer architecture or a recurrent circuit with one XOR gate and $N$ time steps);
2. with $N-1$ XOR gates arranged in a tree (a $\log_2 N$ layer architecture);
3. a DNF formula (i.e. two layers) with a number of min-terms proportional to $2^N$.

In theorem 3.4 (Section 3) we state a similar result for learning architectures: an exponential number of terms is required with a Gaussian kernel machine in order to represent the parity function. In many instances, space can be traded for time (or layers) with considerable advantage. In the case of boolean circuits, a similar theorem shows that an exponential number of elements (logical AND, OR, or NOT gates) are required to implement parity with a 2-level circuit (Ajtai, 1983).

Another interesting example in which adding layers is beneficial is the fast Fourier transform algorithm (FFT). Since the discrete Fourier transform is a linear operation, it can be performed by a matrix multiplication with $N^2$ scalar multiplications, which can all be performed in parallel, followed by on the order of $N^2$ additions to collect the sums. However the FFT algorithm can reduce the total cost to $\frac{1}{2}N \log_2 N$, multiplications, with the trade-off of requiring $\log_2 N$ sequential steps involving $\frac{N}{2}$ multiplications each. This example shows that, even with linear functions, adding layers allows us to take advantage of the intrinsic regularities in the task.

These exemples and the complexity theory circuits highlight a severe limitation of shallow architectures, i.e., of kernel machines (grandmother cells followed by linear predictors) and one-hidden-layer neural networks. They may need to be exponentially large to represent functions that may otherwise be represented compactly with a deep architecture, e.g. a deep neural network.

# 3 The Problem with Template Matching and Local Kernels

This section focusses more mathematically than the previous one on specific shallow architectures, kernel machines with *local kernels*, corresponding to a layer of template-

matching units followed by linear aggregation. It shows that when the function to be learned has many variations (twists and turns), which can come about because of the interaction of many explanatory factors, such architectures may require a very large number of training examples and computational elements. This section can be skipped without losing the main thread of the chapter.

Kernel machines compute a function of their input that has the following structure:

$$f(x) = b + \sum_i \alpha_i K(x, x_i) \tag{1}$$

where $K(\cdot, \cdot)$ can be understood as a matching function. It is chosen a priori and represents a notion of similarity between two input objects. A typical kernel function is the Gaussian kernel,

$$K_\sigma(u, v) = e^{-\frac{1}{\sigma^2}||u-v||^2}, \tag{2}$$

in which the width $\sigma$ controls the locality of the kernel. In biological terms, this approach corresponds to two levels of processing: first the estimation of similarity of the current sensory pattern $x$ with many previously-encountered patterns, and then some form of voting between the matching patterns, in order to come up with a decision or a prediction $f(x)$. When $K(x, x_i)$ is substantially greater than its base output level, this would be like a grandmother cell for the "grandmother image" $x_i$ firing in response to input $x$. This architecture can be summarized as follows: at the bottom, a matching level and at the top, a linear classification or linear regression level that aggregates all the matches into one prediction or decision. Only the top level is fully tuned to the task (the bottom one is learned in a simple and unsupervised way by copying raw inputs). In an artificial neural network with a single hidden layer, trained in a supervised way or by reinforcement learning, there are also two levels but both can be fully tuned to the task. Both are shallow architectures. A more shallow architecture is the linear classifier or linear regressor, corresponding to a single layer of neurons, as in Rosenblatt's Perceptron (Rosenblatt, 1957). The limits of the Perceptron are well understood (Minsky & Papert, 1969). In this chapter, we emphasize the less obvious but nonetheless serious limits of shallow architectures such as a kernel machine. One reason why these limitations may have been overlooked is that unlike the Perceptron, such architectures are universal approximators; with enough [training] data, they can approximate any continuous function arbitrarily closely. However, the number of required elements (i.e., the number of training examples, or grandmother cells) could be very large. As we show below, in many cases it the requirement may be exponential with respect to the size of the input pattern.

To establish the intuition motivating the mathematical results below, consider the apparently simple problem of pattern recognition and classification when the input images are handwritten characters with various backgrounds. One of the fundamental problems in pattern recognition is how to handle intra-class variability. For example, we can picture the set of all the possible '2' images as a continuous manifold in the pixel space. Any pair of '2' images on this manifold can be connected by a path, along which every point corresponds to some image of a '2'. The dimensionality of this manifold at one location corresponds to the number of independent distortions that can can be applied to a shape while preserving its category. For handwritten character

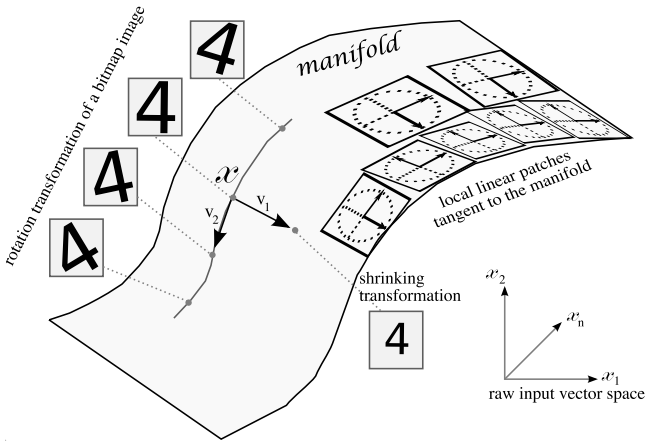


Figure 2: The set of images asssociated with the same object class forms a manifold, i.e. a region of lower dimension than the original space of images. By rotating, translating, or shrinking the image we get other images of the same class, i.e., on the same manifold. Since the manifold is locally smooth, it can in principle be approximated locally by linear patches. However, if the manifold is highly curved, many such patches will be needed.

categories, the manifold has a high dimension: characters can be distorted using affine transforms (6 parameters), distorted using an elastic sheet deformation (high dimension), or modified so as to cover the range of possible writing styles (with or without a loop, with tick or thin stroke,...), and the backgrounds can change (high dimension). Even for simple character images, the manifold is highly non-linear, with high curvature. Moreover, manifolds for different categories are closely intertwined. Consider the shape of a capital U and an O at the same location. They have many pixels in common, many more pixels in fact than with a shifted version of the same U. Another insight about the high curvature of these manifolds can be obtained from the example of translating a high-contrast image. The tangent of a manifold is a locally linear approximation of the manifold, appropriate around a particular point of the manifold, as illustrated in Figure 2. Consider the one-dimensional manifold that is the set of images generated by taking a particular image and allowing it to be rotated or translated left or right by different amounts. Analyzing such a manifold shows that the tangent vector of this manifold (which can be seen as an image itself) changes abruptly as we translate the image only one pixel to the right, indicating high curvature of the manifold. As discussed earlier, many kernel algorithms make an implicit assumption of a locally smooth function around each training example $x_i$. They approximate the manifold with a locally linear patch around each $x_i$. For Support Vector Machines or SVMs (Boser, Guyon, & Vapnik, 1992; Cortes & Vapnik, 1995; Vapnik, 1998), as discussed in section 3.2, the function is locally linear. Hence a high curvature implies the necessity of a large number of training examples in order to cover all the desired turns with locally constant or locally linear patches. The basic problem is that we have many factors of

variation that can be combined in order to give rise to a rich set of possible patterns. With pattern matching architectures such as kernel machines with a local kernel (e.g., the Gaussian), one must cover the space of these variations, with at least one grandmother cell for each combination of values, especially if a change in values can give rise to a change in the desired response. Even more variations than those outlined above could be obtained by simply combining multiple objects in each image. The number of possible variations then grows exponentially with the number of objects and with the number of dimensions of variation per object. An even more dire situation occurs if the background is not uniformly white, but can contain random clutter. To cover all the important combinations of variations the kernel machine will need many different templates containing each motif with a wide variety of different backgrounds.

## 3.1 Minimum Number of Bases Required

The following theorem informs us about the number of sign changes that a Gaussian kernel machine can achieve, when it has $k$ bases (i.e., $k$ support vectors, or at least $k$ training examples).

**Theorem 3.1 (Theorem 2 of Schmitt (2002)).** *Let $f : \mathbb{R} \to \mathbb{R}$ computed by a Gaussian kernel machine (eq. 1) with $k$ bases (non-zero $\alpha_i$'s). Then $f$ has at most $2k$ zeros.*

We would like to say something about kernel machines in $\mathbb{R}^d$, and we can do this simply by considering a straight line in $\mathbb{R}^d$ and the number of sign changes that the solution function $f$ can achieve along that line.

**Corollary 3.2.** *Suppose that the learning problem is such that in order to achieve a given error level for samples from a distribution $P$ with a Gaussian kernel machine (eq. 1), $f$ must change sign at least $2k$ times along some straight line (i.e., in the case of a classifier, a good decision surface must be crossed at least $2k$ times by that straight line). Then the kernel machine must have at least $k$ bases (non-zero $\alpha_i$'s).*

A proof can be found in Bengio, Delalleau, and Le Roux (2006).

**Example 3.3.** *Consider the decision surface shown in figure 3, which is a sinusoidal function. One may take advantage of the global regularity to learn it with few parameters (thus requiring few examples). By contrast, with an affine combination of Gaussians, corollary 3.2 implies one would need at least $\lceil \frac{m}{2} \rceil = 10$ Gaussians. For more complex tasks in higher dimension, the complexity of the decision surface could quickly make learning impractical when using such a local kernel method.*

Of course, one only seeks to approximate the decision surface $S$, and does not necessarily need to learn it perfectly: corollary 3.2 says nothing about the existence of an easier-to-learn decision surface approximating $S$. For instance, in the example of figure 3, the dotted line could turn out to be a good enough estimated decision surface if most samples were far from the true decision surface, and this line can be obtained with only two Gaussians.

The above theorem tells us that if we are trying to represent a function that locally varies a lot (in the sense that its sign along a straight line changes many times), then

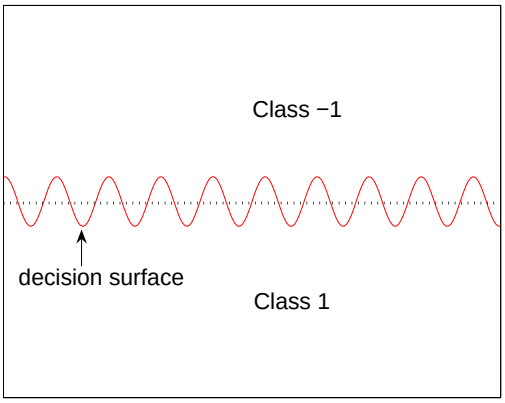


Figure 3: The dotted line crosses the decision surface 19 times: one thus needs at least 10 Gaussians to learn it with an affi ne combination of Gaussians with same width.

we need many training examples to do so with a Gaussian kernel machine. Note that it says nothing about the dimensionality of the space, but we might expect to have to learn functions that vary more when the data is high-dimensional. The next theorem confi rms this suspicion in the case of a highly-varying function, the $d$-bits parity function, which changes value whenever any one of its inputs bits is flipped:

$$\text{parity}: \ (b_1, \ldots, b_d) \in \{0,1\}^d \mapsto \begin{cases} 1 \text{ if } \sum_{i=1}^{d} b_i \text{ is even} \\ -1 \text{ otherwise.} \end{cases}$$

This function is interesting because it varies a lot as we move around in input space. Learning this apparently simple function with Gaussians centered on any of the possible input bit patterns: it requires a number of Gaussians exponential in $d$ (for a fi xed Gaussian width). Note that our earlier corollary 3.2 does not apply to this function, so it represents another type of local variation (not along a line). However, it is also possible to prove a strong result about that case.

**Theorem 3.4.** *Let $f(x) = b + \sum_{i=1}^{2^d} \alpha_i K_\sigma(x_i, x)$ be an affine combination of Gaussians with same width $\sigma$ centered on points $x_i \in X_d$. If $f$ solves the parity problem, then there are at least $2^{d-1}$ non-zero coefficients $\alpha_i$.*

A proof can be found in Bengio et al. (2006).

The bound in theorem 3.4 is tight, since it is possible to solve the parity problem with exactly $2^{d-1}$ Gaussians and a bias, for instance by using a negative bias and putting a positive weight on each example satisfying $\text{parity}(x_i) = 1$.

One may argue that parity is a simple discrete toy problem of little interest. But even if we have to restrict the analysis to discrete samples in $\{0,1\}^d$ for mathematical

reasons, the parity function can be extended to a smooth function on the $[0,1]^d$ hypercube depending only on the continuous sum $b_1 + \ldots + b_d$. Theorem 3.4 is thus a basis to argue that the number of Gaussians needed to learn a function with many variations in a continuous space may scale linearly with these variations, and thus possibly exponentially in the dimension.

## 3.2 Smoothness vs Locality

Consider a Gaussian SVM and how that estimator changes as one varies $\sigma$, the hyperparameter of the Gaussian kernel. For large $\sigma$ one would expect the estimated function to be very smooth, whereas for small $\sigma$ one would expect the estimated function to be more local, in the sense discussed earlier: the near neighbors of $x$ have dominating influence in the shape of the predictor at $x$.

The following proposition tells us what happens when $\sigma$ is large, or when we consider what happens in a ball of training examples with small radius (compared with $\sigma$). It considers the space of possible input examples, and a sphere in that space, that contains all the training examples. The proposition focusses on what happens when the Gaussian width $\sigma$ becomes large compared to that sphere.

**Proposition 3.5.** *For the Gaussian kernel classifier, as $\sigma$ increases and becomes large compared with the diameter of the data, within the smallest sphere containing the data the decision surface becomes linear if $\sum_i \alpha_i = 0$ (e.g., for SVMs), or else the normal vector of the decision surface becomes a linear combination of two sphere surface normal vectors, with each sphere centered on a weighted average of the examples of the corresponding class.*

A proof can be found in Bengio et al. (2006).

This proposition shows that when $\sigma$ becomes large, a kernel classifier becomes non-local (it approaches a linear classifier). However, this non-locality is at the price of constraining the decision surface to be very smooth, making it difficult to model highly varying decision surfaces. This is the essence of the trade-off between smoothness and locality in many similar non-parametric models (including the classical ones such as k-nearest-neighbor and Parzen windows algorithms).

Now consider in what senses a Gaussian kernel machine is local (thinking about $\sigma$ small). Consider a test point $x$ that is near the decision surface. We claim that the orientation of the decision surface is dominated by the Euclidean neighbors $x_i$ of $x$ in the training set, making the predictor *local in its derivative*. *It means that variations around an input $x$ of the decision surface represented by the kernel machine $f$ are mostly determined by the training examples in the neighborhood of $x$.* If we consider the coefficients $\alpha_i$ fixed (i.e., ignoring their dependence on the training $x_i$'s), then it is obvious that the prediction $f(x)$ is dominated by the near neighbors $x_i$ of $x$, since $K(x, x_i) \to 0$ quickly when $||x - x_i||/\sigma$ becomes large. However, the $\alpha_i$ can be influenced by all the $x_j$'s. The following proposition skirts that issue by looking at the first derivative of $f$.

**Proposition 3.6.** *For the Gaussian kernel classifier, the normal of the tangent of the decision surface at $x$, with $x$ on the decision surface, is constrained to approximately*

*lie in the span of the vectors* $(x - x_i)$ *with* $||x - x_i||$ *not large compared to* $\sigma$ *and* $x_i$ *in the training set.*

See Bengio and Le Cun (2007) for a proof.

The constraint of $\frac{\partial f(x)}{\partial x}$ being in the span of the vectors $x - x_i$ for neighbors $x_i$ of $x$ is not strong if the the region of the decision surface where data concentrate (a manifold of possibly lower dimension than the decision surface itself) has low dimensionality. Indeed if that dimensionality is smaller or equal to the number of dominating neighbors, then the kernel machine decision surface is not strongly constrained by the neighboring examples. However, when modeling complex dependencies involving many factors of variation, the region of interest may have high dimension (e.g., consider the effect of variations that have arbitrarily large dimension, such as changes in clutter, background , etc. in images). For such a complex highly-varying target function, we also need a local predictor ($\sigma$ small) in order to accurately represent all the desired variations. With a small $\sigma$, the number of dominating neighbors will be small compared to the dimension of the manifold of interest, making this locality in the derivative a strong constraint, and allowing the following curse of dimensionality argument.

This notion of locality in the sense of the derivative allows us to defi ne a ball around each test point $x$, containing neighbors that have a dominating influence on $\frac{\partial f(x)}{\partial x}$. Smoothness within that ball constrains the decision surface to be approximately either linear (case of SVMs) or a particular quadratic form. Let $N$ be the number of such balls necessary to cover the region $\Omega$ where the value of the predictor is desired (e.g., near the target decision surface, in the case of classifi cation problems). Let $k$ be the smallest number such that one needs at least $k$ examples in each ball to reach error level $\epsilon$. The number of examples thus required is $kN$. To see that $N$ can be exponential in some dimension, consider the maximum radius $r$ of all these balls and the radius $R$ of $\Omega$. If $\Omega$ has intrinsic dimension $d$, then $N$ could be as large as the number of radius-$r$ balls that can tile a $d$-dimensional manifold of radius $R$, which is on the order of $\left(\frac{R}{r}\right)^d$. This means that *in order to cover the possibly variations in the data that matter to define the decision surface, one may need a number of examples that grows as* $\left(\frac{R}{r}\right)^d$.

# 4 Learning Abstractions One on Top of the Other

The analyses of the previous sections point to the diffi culty of learning *highly-varying functions*, i.e., functions that have a large number of *variations* in the domain of interest. These analyses focus on shallow architectures and learning algorithms that generalize only locally, such as kernel machines with Gaussian kernels, i.e. grandmother cells. This problem also plagues many non-parametric unsupervised learning algorithms which attempt to cover the manifold near which the data concentrate. Such models can in principle cover the space of variations of the input examples by a large number of locally linear patches. Since the number of locally linear patches can be made to grow exponentially with the number of input variables, this problem is directly connected with the well-known curse of dimensionality for classical non-parametric learning algorithms (for regression, classifi cation and density estimation). If the shapes of all these linear patches are unrelated, one needs enough examples for

each patch in order to generalize properly. However, if these shapes are related and can be predicted from each other, *non-local learning algorithms* have the potential to generalize to regions not covered by the patches arising from the training set. Such ability would seem necessary for learning in complex domains such as those involved in intelligent behavior.

One way to represent a highly-varying function compactly (with few parameters) is through the composition of many non-linearities. Such multiple composition of non-linearities appear to grant non-local properties to the estimator, in the sense that the value of $f(x)$ or $f'(x)$ can be strongly dependent on training examples far from $x_i$ while at the same time allowing $f(\cdot)$ to capture a large number of variations. We have already discussed the example of parity in the previous two sections. Other arguments can be brought to bear to strongly suggest that learning of more abstract functions is much more efficient when it is done sequentially, composing previously learned concepts in order to represent and learn more abstract concepts (Utgoff & Stracuzzi, 2002).

The raw input object, e.g., the vector of gray-scale values of all the pixels in an image constitutes an initial *low level representation* of the input. This is to be contrasted with *high level representations* such as a set of symbolic categories for that input object (e.g., in an image: `man, sitting...`). Many intermediate levels of representation can exist in between these two extremes. Indeed, deep multi-layered systems, with each layer extracting a slightly higher level representation of its input patterns, have long been believed to be the key to building the ultimate intelligent learning systems (Utgoff & Stracuzzi, 2002). Unfortunately, we generally do not know a set of intermediate and high-level concepts which would appropriate to explain the data. It would therefore be important to have algorithms that can discover such abstractions. However this vision has proved difficult to actualize; learning deep-layered architecture is known to be problematic (Tesauro, 1992) because it is a difficult **optimization** problem. For this reason, many recent successful approaches in machine learning (Schölkopf, Burges, & Smola, 1999) seem to have given up on the notion of multiple levels of transformations altogether, in favor of analytical simplicity and theoretical guarantees. Recently, however, Hinton, Osindero, and Teh (2006) have demonstrated algorithms that suggest that the difficulties of learning with many layers can be overcome.

Deep architectures are perhaps best exemplified by multi-layer neural networks with several hidden layers. In general terms, deep architectures form a **composition** of non-linear modules, each of which can be adapted. Deep architectures rarely appear in the literature. Indeed, the vast majority of neural network research has focused on shallow architectures with a single hidden layer, because of the difficulty of training networks with more than 2 or 3 layers (Tesauro, 1992). Notable exceptions include work on convolutional networks (LeCun, Boser, Denker, Henderson, Howard, Hubbard, & Jackel, 1989; LeCun, Bottou, Bengio, & Haffner, 1998), and recent work on Deep Belief Networks (Hinton et al., 2006; Bengio, Lamblin, Popovici, & Larochelle, 2007) or DBNs. The latter are probabilistic generative models of the data, along with a fast approximation of the posterior (determining what higher-level causes or abstractions are likely to be involved in explaining the current input), and a greedy layer-wise training algorithm that works by training one layer at a time in an unsupervised fashion. Each layer is trained to model the distribution of its input, which is the output

of the previous layer. Upper layers of a DBN are supposed to represent more abstract concepts that explain the input observation $x$, whereas lower layers extract low-level features from $x$. They learn simpler concepts first, and more abstract concepts are learned by composing them. Although DBNs have been introduced only recently, it has already been shown that they can learn efficient high-level representations. They have also been shown to beat state-of-the-art methods by a comfortable margin (Hinton et al., 2006; Bengio & Le Cun, 2007) on MNIST (a well known benchmark task involving classification of digit images), when no prior knowledge on images is allowed. Deep Belief Networks are described in more detail in this book, in Hinton's chapter.

However, the idea that learning occurs in stages, with different levels of concepts, dates back at least to Piaget (Piaget, 1952). Humans do not learn very abstract mathematical concepts until the end of adolescence. For example, they start by learning the notion of objects in the world, they learn to count and doing simple mathematical operations on them, and gradually build on these early concepts in order to learn to represent more abstract concepts. This strategy makes a lot of sense from a mathematical point of view: the optimization problem of learning the more abstract concepts directly would appear too difficult (e.g., training a deep neural network gets stuck in poor local minima), whereas sequentially breaking the problem into simpler ones (e.g., learning less abstract concepts first, and gradually more abstract ones on top) is a common type of optimization heuristic. Biological evidence for maturation one stage after the other is less clear (Guillery, 2005), but some observations support a hierarchical sequence of maturations.

# 5 What is Needed

To design learning algorithms that handle more complex data distributions such as those presumably involved in artificial and natural intelligence, we believe that bold steps are required, not just fine-tuning of existing algorithms. We hypothesize that significant progress can be achieved through a small set of computational and mathematical general-purpose principles, by opposition to a large set of engineered special-purpose tricks. By general-purpose or special-purpose we want to distinguish methods that apply to a large class versus a tiny class of tasks. Keeping in mind that there exists no *completely universal* statistical learning algorithm (Wolpert, 1996), it suffices that such broadly applicable generalization principles be relevant to the type of learning tasks that we care about, such as those solved by humans and animals.

Another hypothesis on which this chapter has focused, is that we are are better off using *deep architectures* than *shallow architectures* in order to learn complex highly-varying functions such as those involved in intelligent behaviors. In particular one of our objectives is to investigate deep architectures that are obtained by the principle of composition: more abstract and more non-linear functions are represented as the composition of simpler ones, and this is done at multiple levels. The number of such levels is the depth of the architecture, and corresponds to depth of a circuit if that function is represented as a circuit. To achieve generalization on examples that are truly novel, and to generalize across tasks, it is important that the components get to be re-used in different places and different tasks.

In order to be able to learn highly-varying complex functions, we believe that one needs learning algorithms that can cope with very large datasets, that can take advantage of a large number of inter-related tasks, that have a deep architecture obtained by composition of simpler components, and that can learn even when most of the examples are unlabeled, and when the inputs are very high-dimensional.

We put together below a set of requirements which we believe necessary in learning algorithms for intelligent behaviors.

- *Learning complex abstractions through composition of simpler ones.* Our work already strongly suggests that high-level abstractions require deep architectures, but there remains the question of optimizing these architectures. Learning complex abstractions (highly-varying functions) is likely to involve a difficult optimization problem, but a promising strategy is to break this problem into simpler (even possibly convex) sub-problems. Successful examples of this strategy are found in recent work, following the work on Deep Belief Networks (Hinton et al., 2006), i.e., with greedy layer-wise training of deep networks (Bengio et al., 2007; Hinton & Salakhutdinov, 2006; Ranzato, Poultney, Chopra, & LeCun, 2006).

- *Unsupervised and semi-supervised learning are key.* To learn complex abstractions requires a lot of data, and tagging it would be prohibitively expensive. Most current state-of-the-art unsupervised and semi-supervised learning algorithms are of the memorizing type (local non-parametric models) and the mathematical results outlined here show that they will suffer from the curse of dimensionality, i.e., they are unlikely to have the ability to learn abstractions that actually have a simple expression but appear complex because they correspond to a great variety of examples or give rise to highly variable functions.

- *The more variables and task, the better?* Although common belief (that has some justifications) is that learning is harder when there are more variables involved (high-dimensional spaces, curse of dimensionality, etc.), we hypothesize that one can take advantage of the presence of many variables, as long as their relations are not arbitrary but relate to shared underlying realities. If we consider predicting one variable from the others to be one of a series of tasks, and we apply the principles of *multi-task learning*, there should be an advantage to working with more random variables, as long as they are meaningfully related. This idea also means that instead of separately tackling each task, we devote a great part of our effort on learning concepts that are relevant to a large number of tasks, e.g., concepts that help to make sense of the world around us.

- *Great quantities of data call for on-line learning.* If a large number of examples are required to learn complex concepts then we should strive to develop learning algorithms whose computational requirements scale linearly with the number of examples. On-line learning algorithms visit each example only once. Other variants are possible, but the overall training time should not scale much worse than linearly in the number of examples.

- *Predictive, reinforcement and active learning.* Although most data are unlabeled, the task of predicting what comes next can be achieved with supervised learning

algorithms, as components in the unsupervised learning task. Because what we are trying to learn is complex, passively observing it for a lifetime may not be sufficient to collect and process enough data. Active learning algorithms (Cohn, Ghahramani, & Jordan, 1995; Fukumizu, 1996) suggest actions that influence what examples are seen next, i.e., in which direction to explore in order to acquire data that brings more information. They can potentially give rise to exponentially faster learning. In this context, the learning algorithms must consider the optimization of a *sequence of decisions*, as in reinforcement learning.

- *Learning to represent context at multiple levels.* Another challenge for learning algorithms is that many of the statistical dependencies that matter in the performance of intelligent tasks involve events at different times and are greatly influenced by temporal context. This means that learning algorithms must involve models of the dynamics in the data, and that an unobserved state representation must be learned, which necessarily involves *long-term dependencies*, that are unfortunately hard to learn with currently known techniques (Bengio, Simard, & Frasconi, 1994; Bengio & Frasconi, 1995). We believe that one of the keys to achieving this goal is to represent context at different levels of abstraction and different time scales (ElHihi & Bengio, 1996).

# 6 Conclusion

Because the brain may be seen as a deep network, computational neuroscience research on the learning mechanisms that involve many layers could serve as very useful inspiration for AI research. Conversely, the algorithmic and mathematical development of ideas in statistical machine learning geared towards training deep networks could also provide hypotheses to inspire computational neuroscience research into learning mechanisms.

The main messages of this chapter are the following.

1. Shallow architectures such as those of linear predictors, kernel machines (grandmother cells and template matching) and single-hidden-layer neural networks are not efficient enough in terms of representation to address the learning of complex functions such as those involved in intelligent behavior. Computational research should pay particular attention to possible learning mechanisms involving many layers of processing together.

2. Local estimators such as kernel machines with a local kernel (e.g., the Gaussian kernel, template matching) are similarly limited, because they cannot discover regularities in the data that are both fine-grained (many local variations) but span a large region of data space (globally coherent structures, principles applicable to many different types of possible inputs). Computational neuroscience models that are limited to template matching followed by linear prediction or linear classification are insufficient to explain the richness of human or animal learning.

3. Deep architectures (e.g., neural networks with many layers) appear the only way to avoid these limitations, and although they were until recently thought to be

too difficult to train, new algorithms strongly suggest that they can be trained efficiently using a greedy layer-wise unsupervised strategy.

4. Deep architectures and the greedy layer-wise strategy exploits the principle, apparently also exploited by humans (Piaget, 1952), that one can more easily learn high-level abstractions if these are defined by the composition of lower-level abstractions, with the property that these lower-level abstractions are useful by themselves to describe the data, and can thus be learned before the higher level abstractions are learned. How such a training by stages could occur in brains remains a question. However, our work (Bengio et al., 2007) suggest that *all the levels could be learning simultaneously*, even though lower levels would presumably converge near their final state earlier.

5. To achieve the learning of intelligent behaviors, this multi-level learning of abstractions should be combined with several other characteristics of learning algorithms: ability to exploit unlabeled data (unsupervised and semi-supervised learning), ability to exploit commonalities between a large number of tasks (multi-task learning) and a large number of inputs (multi-modal learning), on-line learning, learning to represent context at multiple levels, active learning, predictive learning, and reinforcement learning. Most of these have been considered separately in the machine learning community, but it is time to start putting them together in one system.

**Acknowledgments**

The author wants to acknowledge the intellectual influence and contributions to the results and ideas discussed in this chapter, primarily from Yann Le Cun, but also from Geoffrey Hinton, Pascal Lamblin, François Rivest, Olivier Delalleau, Pascal Vincent, Nicolas Le Roux, and Hugo Larochelle. The following funding agencies have also contributed to this work: NSERC, the NCE (MITACS), and the Canada Research Chairs.

# References

Ajtai, M. (1983). $\sum_1^1$-formulae on finite structures. *Annals of Pure and Applied Logic*, *24*(1), 48.

Allender, E. (1996). Circuit complexity before the dawn of the new millennium. In *16th Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pp. 1–18. Lecture Notes in Computer Science 1180.

Bengio, Y., & Frasconi, P. (1995). Diffusion of context and credit information in markovian models. *Journal of Artificial Intelligence Research*, *3*, 223–244.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Bengio, Y., Delalleau, O., & Le Roux, N. (2006). The curse of highly variable functions for local kernel machines. In Weiss, Y., Schölkopf, B., & Platt, J. (Eds.), *Advances in Neural Information Processing Systems 18*, pp. 107–114. MIT Press, Cambridge, MA.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA.

Bengio, Y., & Le Cun, Y. (2007). Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., DeCoste, D., & Weston, J. (Eds.), *Large Scale Kernel Machines*. MIT Press.

Bengio, Y., Le Roux, N., Vincent, P., Delalleau, O., & Marcotte, P. (2006). Convex neural networks. In Weiss, Y., Schölkopf, B., & Platt, J. (Eds.), *Advances in Neural Information Processing Systems 18*, pp. 123–130. MIT Press, Cambridge, MA.

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifi ers. In *Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152 Pittsburgh.

Cohn, D., Ghahramani, Z., & Jordan, M. I. (1995). Active learning with statistical models. In Tesauro, G., Touretzky, D., & Leen, T. (Eds.), *Advances in Neural Information Processing Systems 7*. Cambridge MA: MIT Press.

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning*, *20*, 273–297.

Duda, R., Hart, P., & Stork, D. (2001). *Pattern Classification, Second Edition*. Wiley and Sons, New York.

ElHihi, S., & Bengio, Y. (1996). Hierarchical recurrent neural networks for long-term dependencies. In Touretzky, D., Mozer, M., & Hasselmo, M. (Eds.), *Advances in Neural Information Processing Systems 8*, pp. 493–499. MIT Press, Cambridge, MA.

Freund, Y., & Schapire, R. E. (1997). A decision theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Science*, *55*(1), 119–139.

Fukumizu, K. (1996). Active learning in multilayer perceptrons. In Touretzky, D., Mozer, M., & Hasselmo, M. (Eds.), *Advances in Neural Information Processing Systems 8*. MIT Press, Cambridge, MA.

Guillery, R. (2005). Is postnatal neocortical maturation hierarchical?. *Trends in Neuroscience*, *28*(10), 512–517.

Hastad, J. T. (1987). *Computational Limitations for Small Depth Circuits*. MIT Press, Cambridge, MA.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning: data mining, inference and prediction*. Springer Series in Statistics. Springer Verlag.

Hinton, G. E., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–1554.

Hinton, G., & Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Jordan, M. (1998). *Learning in Graphical Models*. Kluwer, Dordrecht, Netherlands.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Minsky, M., & Papert, S. (1969). *Perceptrons*. MIT Press, Cambridge.

Piaget, J.-P. (1952). *The origins of intelligence in children*. International Universities Press, New York.

Ranzato, M., Poultney, C., Chopra, S., & LeCun, Y. (2006). Effi cient learning of sparse representations with an energy-based model. In et al., J. P. (Ed.), *Advances in Neural Information Processing Systems (NIPS 2006)*. MIT Press.

Rosenblatt, F. (1957). The perceptron — a perceiving and recognizing automaton. Tech. rep. 85-460-1, Cornell Aeronautical Laboratory, Ithaca, N.Y.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Schmitt, M. (2002). Descartes' rule of signs for radial basis function neural networks. *Neural Computation*, *14*(12), 2997–3011.

Schölkopf, B., Burges, C. J. C., & Smola, A. J. (1999). *Advances in Kernel Methods — Support Vector Learning*. MIT Press, Cambridge, MA.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*, 257–277.

Utgoff, P., & Stracuzzi, D. (2002). Many-layered learning. *Neural Computation*, *14*, 2497–2539.

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley, Lecture Notes in Economics and Mathematical Systems, volume 454.

Wolpert, D. (1996). The lack of a priori distinction between learning algorithms. *Neural Computation*, *8*(7), 1341–1390.