**DIMACS Technical Report 99-49**
**October 1999**

# On the Computational Power of Analog Neural Networks

by

Bhaskar DasGupta[1,2]
Dept. of Computer Science
Rutgers University
Camden, New Jersey 08102
Email: bhaskar@crab.rutgers.edu

Georg Schnitger
Fachbereich 20, Informatik
Universität Frankfurt
60054 Frankfurt, Germany
Email: georg@thi.informatik.uni-frankfurt.de

[1]DIMACS Visitor
[2]Supported by NSF Grant CCR-9800086.

---

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, Telcordia Technologies (formerly Bellcore) and NEC Research Institute.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

## ABSTRACT

We survey the computational power of analog neural networks.

# 1 Introduction

Artificial neural networks are proposed as a tool for machine learning and many results have been obtained regarding their application to practical problems in robotics control, vision, pattern recognition, grammatical inferences and other areas. Typically, a neural network is trained during a *supervised* training session to recognize complex associations between inputs and outputs. These associations are incorporated into the weights of the network, which encode a distributed representation of the information contained in the input patterns. Once trained, the network will compute an input/output mapping which, if the training data was representative enough, will closely match the unknown rule producing the original data. The applicability of learning algorithms such as the backpropagation algorithm, the massive parallelism of computation, as well as noise and fault tolerance, are some of the prominent features of neural nets.

Feedforward circuits composed of AND, OR, NOT or threshold gates have been thoroughly studied in theoretical computer science. However, due to the backpropagation algorithm and its variants, differentiable activation functions such as the *standard sigmoid* are more commonly used. The last two decades have seen a resurgence of theoretical techniques to design and analyze the performance of neural nets as well as novel applications of neural nets to various applied areas. Theoretical studies towards the computational capabilities of neural nets have given valuable insights into the mechanisms of these models.

In subsequent discussions, we distinguish between feedforward neural nets and recurrent neural nets. The architecture of a feedforward net $\mathcal{N}$ is described by an interconnection graph and the activation functions of $\mathcal{N}$. A node (processor or neuron) $v$ of $\mathcal{N}$ computes a function

$$\gamma_v \left( \sum_{i=1}^{k} a_{v_i,v} u_{v_i} + b_v \right) \tag{1}$$

of its inputs $u_{v_1}, \ldots, u_{v_k}$. These inputs are either external (i.e., representing the input data) or internal (i.e., representing the outputs of the immediate predecessors of $v$). The coefficients $a_{v_i,v}$ (resp. $b_v$) in (1) are the *weights* (resp. *threshold*) of node $v$, and the function $\gamma_v$ is the *activation function* of $v$. No cycles are allowed in the interconnection graph and the output of designated nodes provides the output of the network. A recurrent neural net, on the other hand, allows cycles, thereby providing potentially higher computational capabilities. The *state* $u_v$ of node $v$ in a recurrent net is updated over time according to

$$u_v(t+1) = \gamma_v \left( \sum_{i=1}^{k} a_{v_i,v} u_{v_i}(t) + b_v. \right) \tag{2}$$

In this chapter, we emphasize the exact and approximate representational power of feedforward and recurrent neural nets. This line of research can be traced back to Kolmogorov (Kolmogorov 1957), who essentially proved the first existential result on the (exact) representation capabilities of neural nets. The need to work with "well-behaved" activation functions however enforces approximative representations of target functions and the question of the approximation power (within limited resources) becomes fundamental.

Certainly there are immediate consequences of representation power of neural nets for learning, since we cannot learn (approximately) what we cannot represent (approximately). On the other hand, learning complexity increases with increasing representational power of the underlying neural model and care has to be exercised to strike a balance between representational power on one hand and learning complexity on the other.

Due to space limitations and limitations on the number of references, we discuss only a small subset of the vast literature on this topic. The rest of the chapter is organized as follows. In section 2, we introduce the basic notation. The representational power of feedforward, respectively recurrent neural nets is discussed in sections 3 and 4. Section 3.2 gives a brief discussion of networks of spiking neurons and their relation to sigmoidal nets. Section 5 concludes the chapter.

# 2 Models and Basic Definitions

We present notation and basic definitions for subsequent sections. For real-valued functions we measure the approximation quality of function $f$ by function $g$ (over a domain $D \subseteq \mathbb{R}^n$) by the Chebychev norm

$$||f - g||_D = \sup\{|f(x) - g(x)| \ : \ x \in D\}$$

(the subscript $D$ will be omitted when clear from the context). To emphasize the selection of activation functions we introduce the concept of $\Gamma$-nets for a class $\Gamma$ of real-valued activation functions. A $\Gamma$-net $\mathcal{N}$ assigns only functions in $\Gamma$ to nodes. We assume that each function in $\Gamma$ is defined on some subset of $\mathbb{R}$ and require that $\Gamma$ contains the identity function by default (thus allowing weighted additions as node outputs). Finally we restrict our attention to $\Gamma$-nets with a single output node.

The *depth* of a feedforward net $\mathcal{N}$ is the length of the longest path of the (acyclic) interconnection graph of $\mathcal{N}$, and the *size* of $\mathcal{N}$ is the number of nodes. The *hidden nodes* are all nodes excluding all input nodes and the output node.

The class of important activation functions is rather large and includes among others the binary threshold function $\mathcal{H}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$, the cosine squasher,the gaussian, the standard sigmoid $\sigma(x) = 1/(1 + e^{-x})$, the hyperbolic tangent, (generalized) radial basis functions, polynomials and trigonometric polynomials, splines and rational functions.

Care must be exercised when using a neural net with continuous activation functions to compute a Boolean-valued function, since in general the output node computes a real number. A standard output convention in this case is as follows (see (Maass 1994)):

**Definition 2.1** *A $\Gamma$-net $\mathcal{N}$ computes a Boolean function $F : \mathbb{R}^n \rightarrow \{0, 1\}$ with separation $\epsilon > 0$ if there is some $t \in \mathbb{R}$ such that for any input $x \in \mathbb{R}^n$ the output node of $\mathcal{N}$ computes a value which is at least $t + \epsilon$ if $F(x) = 1$, and at most $t - \epsilon$ otherwise.*

Recurrent neural nets, unlike their feedforward counterparts, allow loops in their interconnection graph. Certainly *asynchronous* recurrent nets are an important neural model, but we assume in (2) that all nodes update *synchronously* at each time step. Typically, besides internal and external data lines, some of the inputs and outputs are validation lines, indicating if there is any input or output present at the time.

# 3 Computational Power of Feedforward Nets

The simple perceptron as a feedforward neural net with one layer has only limited computational abilities. For instance, if we restrict ourselves to one-node simple perceptrons and assume monotone, but otherwise arbitrary, activation functions, then the XOR-function $\mathrm{XOR}(x_1, x_2) = x_1 \oplus x_2$ cannot be computed.

On the other hand, if we choose the binary threshold function $\mathcal{H}$ as activation function, then the learning problem for simple perceptrons is efficiently solvable by linear programming. This positive result is also extendable to a large class of activation functions including the standard sigmoid. But simple perceptrons should not be underestimated, since the problem of approximately minimizing the miss-classification ratio (when the target function is not representable as a simple perceptron) has been shown to be (probably) intractable (Arora et. al. 1997).

However, the power of feedforward nets increases significantly when networks of more layers are considered. In fact, a result of Kolmogorov (refuting Hilbert's 13th problem for continuous functions), when translated into neural net terminology, shows that any continuous function can be computed *exactly* by a feedforward net of depth 3.

**Theorem 3.1** *(Kolmogorov 1957) Let $n$ be a natural number. Then there are continuous functions $h_1, \ldots, h_{2n+1} : [0,1] \to \mathbb{R}$ such that any continuous function $f : [0,1]^n \to \mathbb{R}$ can be represented as*

$$f(x) = \sum_{j=1}^{2n+1} g\left(\sum_{i=1}^{n} \alpha_i h_j(x_i)\right),$$

*where the function $g$ as well as the weights $\alpha_1, \ldots, \alpha_n$ depend on $f$.*

But, unfortunately, the function $g$ depends on the function to be represented. Moreover, the functions $h_j$ are non-differentiable and hence cannot be used by current learning algorithms. For a further discussion we refer the reader to the survey (Poggio and Girosi 1989).

However, if we only allow everywhere differentiable activation functions (such as the standard sigmoid), then we can only represent everywhere differentiable target functions. Thus one has to relax the requirement of exact representation, and demand only that the approximation error (in an appropriate norm) is small. Applying the Stone-Weierstrass theorem one obtains for instance (trigonometric) polynomials as universal approximators and hence we get neural nets with one hidden layer as universal approximators.

Cybenko (Cybenko 1989) considers activation functions from the class of continuous *discriminatory* functions. This class contains for instance *sigmoidal* functions, i.e., continuous functions $\sigma$ satisfying

$$\sigma(t) \rightarrow \begin{cases} 1 & \text{as } t \rightarrow +\infty \\ 0 & \text{as } t \rightarrow -\infty \end{cases}$$

**Theorem 3.2** *Let $\sigma$ be a continuous discriminatory function and let $f : [0,1]^n \rightarrow \mathbb{R}$ be a continuous target function. Then, for every $\epsilon > 0$ and for sufficiently large $N$ (where $N$ depends on the target function $f$ and $\epsilon$), there exist weights $\alpha_{ij}, w_j$ and thresholds $\beta_j$, such that $||f - g|| < \epsilon$, where $g = \sum_{j=1}^{N} w_j \cdot \sigma(\sum_{i=1}^{n} \alpha_{ij} \cdot x_i + \beta_j)$.*

In particular, one hidden layer suffices to approximate any continuous function by sigmoidal nets within arbitrarily small error. Further results along this line are shown by Hornik, Stinchcombe and White, Funahashi, Moore and Poggio as well as Poggio and Girosi to mention just a few. Whereas most arguments in the above mentioned results are nonconstructive, Carroll and Dickinson describe a method using Radon transforms to approximate a given $L_2$ function to within a givenmean square error.

Barron (Barron 1993) discusses the approximation quality achievable by sigmoidalnets of small size. In particular, let $B_r^n(0)$ denote the $n$-dimensional ball with radius $r$ around 0 and let $f : B_r^n(0) \rightarrow \mathbb{R}$ be the target function. Assume that $F$ is the magnitude distribution of the Fourier transform of $f$.

**Theorem 3.3** *(Barron 1993) Let $\sigma$ be any sigmoidal function. Then for every probability measure $\mu$ and for every $N$ there exist weights $\alpha_{ij}, w_j$ and thresholds $\beta_j$, such that*

$$\int_{B_r^n(0)} (f(x) - \sum_{j=1}^{N} w_j \cdot \sigma(\sum_{i=1}^{n} \alpha_{ij} \cdot x_i + \beta_j))^2 \, \mu(dx) \leq \frac{(2 \int r \cdot |w| \cdot F(dw))^2}{N}.$$

Set $C_f = \int r \cdot |w| \cdot F(dw)$ and the approximation error achievable by sigmoidal nets of size $N$ is bounded by $\frac{(2 \cdot C_f)^2}{N}$. However, $C_f$ may depend superpolynomially on $n$ and the curse of dimensionality may strike. As an aside, the best achievable squared error for linear combinations of $N$ *basis functions* will be at least $\Omega(\frac{C_f}{n \cdot N^{1/n}})$ for certain functions $f$ (Barron 1993), and hence neural networks are superior to conventional approximation methods from this point of view.

The above results show that depth-2 feedforward nets are universal approximators. This dramatically increased computing power however has a rather negative consequence. Kharitonov (Kharitonov 1993) shows under certain cryptographic assumptions that no efficient learning algorithm will be able to predict the input-output behavior of binary threshold nets with a fixed number of layers. This result even holds, when experimentation is allowed, that is when the learning algorithm is allowed to submit inputs for classification.

In the next section we compare important activation functions in terms of their approximation power, when limiting resources such as depth and size. The subsequent section discusses networks of spiking neurons and lower size-bounds for sigmoidal nets will be mentioned, when comparing networks of spiking neurons and sigmoidalnets.

## 3.1 Efficient Approximation by Feedforward Nets

Our discussion will be informal and we refer the reader to (DasGupta and Schnitger 1993) for details. Our goal is to compare activation functions in terms of the size and the depth required to obtain tight approximations. Another resource of interest is the *Lipschitz-bound* of the net, which is a measure of the numerical stability of the circuit. Informally speaking, for a net $\mathcal{N}$ to have Lipschitz bound $L$ we first demand that all weights and thresholds of $\mathcal{N}$ are bounded in absolute value by $L$. Moreover we require that each activation function of $\mathcal{N}$ has (the conventional) Lipschitz-bound $L$ on the inputs it receives. Finally, the actually received inputs have to be bounded away from regions with higher Lipschitz-bounds.

We formalize the notion of having *essentially the same approximation power.*

**Definition 3.1** *Let $\Gamma_1$ and $\Gamma_2$ be classes of activation functions.*
**(a)** *We say that $\Gamma_2$ simulates $\Gamma_1$ (denoted by $\Gamma_1 \leq \Gamma_2$) if and only if there is a constant $k \geq 1$ such that for all $\Gamma_1$-nets $C_2$ of size at most $s$, depth at most $d$ and Lipschitz-bound $2^s$ there is*
*a $\Gamma_2$-circuit $C_1$ of size at most $(s+1)^k$, depth at most $k \cdot (d+1)$ and Lipschitz-bound $2^{(s+1)^k}$, such that*

$$||\mathcal{C}_1 - \mathcal{C}_2||_{[-1,1]^n} \leq 2^{-s}.$$

**(b)** *We say that $\Gamma_1$ and $\Gamma_2$ are equivalent if and only if $\Gamma_1 \leq \Gamma_2$ and $\Gamma_2 \leq \Gamma_1$.*

In other words, when simulating classes of gate functions, we allow depth to increase by a constant factor, size and the logarithm of the Lipschitz-bound to increase polynomially. The relatively large Lipschitz-bounds should not come as a surprise, since the negative exponential error $2^{-s}$ requires correspondingly large weights in the simulating circuit.

Splines (i.e., piecewise polynomial functions) have turned out to be powerful approximators and they are our benchmark class of activation functions; in particular we assume that a spline net of size $s$ has as its activation functions splines of degree at most $s$ with at most one knot. Which properties does a class $\Gamma$ of activation functions need to reach the approximation power of splines? The activation functions should be able to approximate polynomials as well as the binary threshold $\mathcal{H}$ with few layers and relatively few nodes!

Tightly approximating polynomials is not difficult as long as there is at least one "sufficiently smooth" non-trivial function $\gamma \in \Gamma$. The crucial problem is to obtain a tight approximation of $\mathcal{H}$. It turns out that $\gamma$-nets achieve tight approximations of $\mathcal{H}$, whenever

$$|\, \gamma(x) - \gamma(x+\varepsilon)\, | = O(\varepsilon/x^2) \;\text{ for }\; x \geq 1, \varepsilon \geq 0 \;\text{ and }\; |\int_1^\infty \gamma(u^2)\, du| \neq 0.$$

Call a function with these two properties *strongly sigmoidal.* We are actually demanding too much, since it suffices to tightly approximate a strongly sigmoidal function $\gamma$ by small $\Gamma$-nets with few layers. Let us call a class $\Gamma$ *powerful* if there is at least one "sufficiently smooth" non-trivial function in $\Gamma$ and if a strongly sigmoidal function can be approximated as demanded above.

Example of powerful singleton classes include for instance $\frac{1}{x}$ as a prime example and more generally any rational function which is not a polynomial, $\exp(x)$ (since $\exp(-x)$ is strongly sigmoidal) and $\ln(x)$ (since $\ln(x+1) - \ln(x)$ is strongly sigmoidal), any power $x^\alpha$ provided $\alpha$ is not a natural number, and the standard sigmoid as well as the gaussian $\exp(-x^2)$.

**Theorem 3.4** **(a)** *Assume that* $\Gamma$ *is powerful. Then splines* $\leq \Gamma$.
**(b)** *The following classes of activation function have equivalent approximation power:*

> *splines (of degree* $s$ *for nets of size* $s$*), any rational function which is not a polynomial, any power* $x^\alpha$ *(provided* $\alpha$ *is not a natural number), the logarithm (for any base),* $\exp(x)$*, and the gaussian* $exp(-x^2)$.

Notable exceptions from the list of equivalent activation functions are polynomials, trigonometric polynomials and the binary threshold function $\mathcal{H}$ (or more generally low-degree splines). Low-degree splines turn out to be properly weaker. The same applies to polynomials, even if we allow any polynomial of degree $s$ as activation function for nets of size $s$. Finally sine nets cannot be simulated (as defined in Definition 3.1) for instance by nets of standard sigmoids and we conjecture that the reverse is also true, namely that nets of standard sigmoids cannot be simulated efficiently by sine nets.

What happens if we relax the required approximation quality from $2^{-s}$ to $s^{-d}$, when simulating nets of depth $d$ and size $s$? Linear splines and the standard sigmoid are still not equivalent, but the situation changes completely, if we *count* the number of inputs when determining size and if we restrict the Lipschitz-bound of the target function to be at most $s^{-d}$. With this modification an even larger class of important functions, including linear splines, polynomials as well as the sine-function, turn out to be equivalent with the standard sigmoid.

The situation for Boolean input and output is somewhat comparable. Maass, Schnitger and Sontag and subsequently DasGupta and Schnitger construct Boolean functions that are computed by sigmoidal nets of constant size (i.e., independent of the number of input bits), whereas $\mathcal{H}$-nets of constant size do not suffice. (See (Maass 1994) for a more detailed discussion.) However (Maass 1993) shows that spline nets of constant degree, constant depth and polynomial size (in the number of input bits) can be simulated by $\mathcal{H}$-nets of constant depth and polynomial size. This simulation holds without any restriction on the weights used by the spline net.

Thus analog computation does help for discrete problems, but apparently by not too much. For a thorough discussion of discrete neural computation we refer to (Siu et. al. 1994).

## 3.2 Sigmoidal Nets and Nets of Spiking Neurons

A formal model of networks of spiking neurons is defined in (Maass 1997). The architecture is described by a directed graph $G = (V, E)$ with $V$ as the set of nodes and $E$ as the set of edges. We interpret nodes as neurons, edges as synapses and assign to each neuron $v$ a threshold function $\Theta_v : \mathbb{R}^+ \to \mathbb{R}^+$. ($\mathbb{R}^+$ denotes the set of non-negative reals.) The value of $\Theta_v(t-t')$ measures the "reluctance" (or the threshold to be exceeded) of neuron $v$ to fire at time $t$ ($t > t'$), assuming that $v$ has fired at time $t'$. This reluctance can be overcome only if the potential $P_v(t)$ of neuron $v$ at time $t$ is at least correspondingly as large.

The potential of $v$ at time $t$ depends on the recent firing history of the presynaptic neurons (or the immediate predecessors) $u$ of $v$. In particular, if the synapse between neurons $u$ and $v$ has the efficacy (or weight) $w_{uv}$, if $\varepsilon_{uv}(t-s)$ is the response to the firing of neuron $u$ at time $s$ ($s < t$) and if the presynaptic neuron $u$ has fired previously for the times in the set $\mathrm{Fire}_\mathrm{u}^\mathrm{t}$, then the potential at time $t$ is defined as

$$P_v(t) = \sum_{(u,v)\in E} \sum_{s\in \mathrm{Fire}_\mathrm{u}^\mathrm{t}} w_{uv} \cdot \varepsilon_{uv}(t-s). \tag{3}$$

Two models, namely deterministic (resp. noisy) nets of spiking neurons are distinguished. The deterministic version assumes that neuron $v$ fires, whenever its potentials $P_v(t)$ reaches $\Theta_v(t-t')$ (with most recent firing $t'$), whereas the more realistic noisy version assumes that the firing probability increases with increasing difference $P_v(t) - \Theta_v(t-t)$.

Thus we can complete the definition of the formal model, assuming that a response function $\varepsilon_{uv} : \mathbb{R}^+ \to \mathbb{R}$ as well as the weight $w_{uv}$ is assigned to the synapse between $u$ and $v$. The model computes by transforming a spike train of inputs into a spike train of outputs. For instance assuming temporal coding with constants $T$ and $c$, the output of a designated neuron with firing times $T + c \cdot t_1, \ldots, T + c \cdot \sum_{i=1}^k t_i, \ldots$ is defined as $t_1, \ldots, \sum_{i=1}^k t_i, \ldots$

The power of spiking neurons shows for the example of the element distinctness function $ED_n$ with real inputs $x_1, \ldots, x_n$, where $ED_n(x) = \begin{cases} 1 & \text{if } x_i = x_j \text{ for some } i \neq j, \\ 0 & \text{if } |x_i - x_j| \geq 1 \text{ for all } i \neq j, \\ \text{arbitrary} & \text{otherwise.} \end{cases}$
We assume that the inputs $x_1, \ldots, x_n$ are represented by $n$ input trains of single spikes. Now it is easy to choose a simple threshold function as well as simple (and indeed identical) response functions, such that even a single spiking neuron with unit weights is capable of computing $ED_n$. On the other hand, any sigmoidal net computing $ED_n$ requires at least $\frac{n-4}{2} - 1$ hidden units (Maass 1997). This result is also the strongest lower size-bound for sigmoidal nets computing a specific function; the argument builds on techniques from (Sontag 1997).

Certainly this one-neuron computation requires time due to the temporal input coding, but the same applies to sigmoidal networks, since, from the point of neurobiology, the $x_i$'s will be obtained after sampling the firing rate of their input neurons.

Nets of spiking neurons are capable of simulating $\mathcal{H}$-nets with at most the same size and hence are properly stronger than $\mathcal{H}$-nets and at least in some cases stronger than sigmoidal nets. Thus careful timing is an advantage that synchronized models cannot overcome.

# 4 Computational Power of Recurrent Nets

The computational power of recurrent nets is investigated in (Siegelmann and Sontag 1994;Siegelmann and Sontag 1995) See also (Siegelmann 1998) for a thorough discussion of recurrent nets and analog computation in general. Recurrent nets include feedforward nets and thus the results for feedforward nets apply to recurrent nets as well. But recurrent nets gain considerable more computational power with increasing computation time. In the following, for the sake of concreteness, we assume that the piecewise-linear function $\pi(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } 0 \leq x \leq 1 \\ 1 & \text{if } x \geq 1 \end{cases}$ is chosen as activation function. We concentrate on binary input and assume that the input is provided one bit at a time.

First of all, if weights and thresholds are integers, then each node computes a bit. Recurrent net with integer weights thus turn out to be equivalent to finite automata and they recognize exactly the class of regular language over the binary alphabet $\{0, 1\}$.

The computational power increases considerably for rational weights and thresholds. For instance, a "rational" recurrent net is, up to a polynomial time computation, equivalent to a Turing machine. In particular, a network that simulates a universal Turing machine does exist and one could refer to such a network as "universal" in the Turing sense. It is important to note that the number of nodes in the simulating recurrent net is fixed (i.e., *does not grow* with increasing input length).

Irrational weights provide a further boost in computation power. If the netis allowed exponential computation time, then arbitrary Boolean functions (including non-computable functions) are recognizable. However, if only polynomial computation time isallowed, then nets have less power and recognize exactly the languagescomputable by polynomial-size Boolean circuits.

# 5 Conclusions

We have discussed the computing power of neural nets, including universal approximation results for feedforward and recurrent neural networks as well as efficient approximation by feedforward nets with various activation functions.

We have to emphasize that this survey is everything but complete. For instance, important topics such as the Vapnik-Chervonenkis dimension of neural nets as well as the complexity of discrete neural computation had to be omitted due to space limitations.

Important open questions relate to proving better upper and lower bounds forsigmoidal nets computing (or approximating) specific functions and toprovide a better understanding of size-depth trade-offs for importantfunction classes. Other neural models, such as networks of spiking neurons,significantly change the computing power and the above questions apply to these models as well.

## References

S. Arora, L. Babai, J. Stern and Z. Sweedyk, 1997. The Hardness of Approximate Optima in Lattices, Codes and Systems of Linear Equations, Journal of Computer and System Sciences, 54, 317-331.

A. R. Barron, 1993. Universal Approximation Bounds for Superpositions of a Sigmoidal Function, IEEE Trans. on Inform. Theory, 39, 3, 930-945.

G. Cybenko, 1989. Approximation by Superposition of a Sigmoidal Function, Mathematics of Control, Signals and Systems, 2, 303-314.

B. DasGupta and G. Schnitger, 1993. The Power of Approximating: A Comparison of Activation Functions, NIPS 5, 615-622. A complete version of the paper is available as a compressed postscript file in the website
`http://crab.rutgers.edu/ ~bhaskar/resume/publ/papers/approx.ps.Z`.

M. Kharitonov, 1993. Cryptographic Hardness of Distribution Specific Learning, Proc. 25th ACM Symp. on the Theory of Computing, 372-381.

A. N. Kolmogorov, 1957. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition, Dokl. Akad. Nauk USSR, 114, 953-956.

W. Maass, 1993. Bounds for the computational power and learning complexity of analog neural nets, Proc. of the 25th Annual ACM Symposium on Theory of Computing, 335-344.

W. Maass, 1994. Sigmoids and Boolean Threshold Circuits, in Theoretical Advances in Neural Computation and Learning, V.P. Roychowdhury, K.Y. Siu, and A. Orlitsky (editors), Kluwer Academic Publishers, 127-151.

W. Maass, 1997. Networks of spiking neurons: the third generation of neural network models, Neural Networks, 10, 9, 1659-1671.

T. Poggio and F. Girosi, 1989. A theory of networks for Approximation and learning, Artificial Intelligence Memorandum, no 1140.

H. T. Siegelmann, 1998. Neural Networks and Analog Computation: Beyond the Turing Limit, Birkhäuser publishers.

H. T. Siegelmann and E. D. Sontag, 1994. Analog computation, neural networks, and circuits, Theoretical Computer Science, 131, 331-360.

H. T. Siegelmann and E. D. Sontag, 1995. On the Computational Power of Neural Nets, Journal of Computer and System Sciences, 50, 132-150.

K.-Y. Siu, V. P. Roychowdhury and T. Kailath, 1994. Discrete Neural Computation: A Theoretical Foundation, Englewood Cliffs, NJ: Prentice Hall.

E. D. Sontag, 1997. Shattering all sets of k points in general position requires $(k-1)/2$ parameters, Neural Computation, 9, 337-348.