Continuous Neural Networks

Nicolas Le Roux
Université de Montréal
Montréal, Québec
nicolas.le.roux@umontreal.ca

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

Abstract

This article extends neural networks to the case of an uncountable number of hidden units, in several ways. In the first approach proposed, a finite parametrization is possible, allowing gradient-based learning. While having the same number of parameters as an ordinary neural network, its internal structure suggests that it can represent some smooth functions much more compactly. Under mild assumptions, we also find better error bounds than with ordinary neural networks. Furthermore, this parametrization may help reducing the problem of saturation of the neurons. In a second approach, the input-to-hidden weights are fully nonparametric, yielding a kernel machine for which we demonstrate a simple kernel formula. Interestingly, the resulting kernel machine can be made hyperparameter-free and still generalizes in spite of an absence of explicit regularization.

1 Introduction

In (Neal, 1994) neural networks with an infinite number of hidden units were introduced, showing that they could be interpreted as Gaussian processes, and this work served as inspiration for a large body of work on Gaussian processes. Neal's work showed a counterexample to two common beliefs in the machine learning community: (1) that a neural network with a very large number of hidden units would overfit and (2) that it would not be feasible to numerically optimize such huge networks. In spite of Neal's work, these beliefs are still commonly held. In this paper we return to Neal's idea and study a number of extensions of neural networks to the case where the number of hidden units is uncountable, showing that they yield implementable algorithms with interesting properties.

Consider a neural network with one hidden layer (and h hidden neurons), one output neuron and a transfer function g. Let us denote V the input-to-hidden weights, $a_i (i = 1, \ldots, h)$ the hidden-to-output weights and β the output unit bias (the hidden units biases are included in V).

The output of the network with input x is then

$$f(x) = \beta + \sum_i a_i g(\tilde{x} \cdot V_i) \tag{1}$$

where V_i is the i-th column of matrix V and $\tilde{x}$ is the vector with 1 appended to x. The output of the i-th hidden neuron is $g(\tilde{x} \cdot V_i)$. For antisymmetric functions g such as tanh, we can without any restriction consider the case where all the a_i's are nonnegative, since changing the sign of a_i is equivalent to changing the sign of V_i.

In ordinary neural networks, we have an integer index i. To obtain an uncountable number of hidden units, we introduce a continuous-valued (possibly vector-valued) index $u \in \mathbb{R}^m$. We can replace the usual sum over hidden units by an integral that goes through the different weight vectors that can be assigned to a hidden unit:

$$f(x) = \beta + \int_{E \subset \mathbb{R}^m} a(u) g[\tilde{x} \cdot V(u)]\, du \tag{2}$$

where $a : E \to \mathbb{R}$ is the hidden-to-output weight function, and $V : E \to \mathbb{R}^{d+1}$ is the input-to-hidden weight function.

How can we prevent overfitting in these settings? In this paper we discuss three types of solutions: (1) finite-dimensional representations of V, (2) $\mathcal{L}^1$ regularization of the output weights with a and V completely free, (3) $\mathcal{L}^2$ regularization of the output weights, with a and V completely free.

Solutions of type 1 are completely new and give new insights on neural networks. Many possible parametrizations are possible to map the "hidden unit index" (a finite-dimensional vector) to a weight vector. This parametrization allows us to construct a representation theorem similar to Kolmogorov's, i.e. we show that functions from K compact of $\mathbb{R}^d$ to $\mathbb{R}$ can be represented by $d+1$ functions from $[0,1]$ to $\mathbb{R}$. Here we study an affine-by-part parametrization, with interesting properties, such as faster convergence (as the number of hidden units grows). Solutions of type 2 ($\mathcal{L}^1$ regularization) were already presented in (Bengio et al., 2005) so we do not focus on them here. They yield a convex formulation but with an infinite number of variables, requiring approximate optimization when the number of inputs is not tiny. Solutions of type 3 ($\mathcal{L}^2$ regularization) give rise to kernel machines that are similar to Gaussian processes, albeit with a type of kernel not commonly used in the literature. We show that an analytic formulation of the kernel is possible when hidden units compute the sign of a dot product (i.e., with formal neurons). Interestingly, experiments suggest that this kernel is more resistant to overfitting than the Gaussian kernel, allowing to train working models with no hyper-parameters whatsoever.

2 Affine neural networks

2.1 Core idea

We study here a special case of the solutions of type (1), with a finite-dimensional parametrization of the continuous neural network, based on parametrizing the term $V(u)$ in eq. 2, where u is scalar.

$$f(x) = \beta + \int_{E\subset\mathbb{R}} a(u) g[\tilde{x}\cdot V(u)]\, du \tag{3}$$

where V is a function from the compact set E to $\mathbb{R}^{d+1}$ (d being the dimension of x).

If u is scalar, we can get rid of the function a and include it in V. Indeed, let us consider a primitive A of a. It is invertible because a is nonnegative and one can consider only the u such that $a(u) > 0$ (because the u such that $a(u) = 0$ do not modify the value of the integral). Making the change of variable $t = A(u)$ (choosing any primitive A), we have $dt = a(u)d(u)$. $V(u)$ will become $V(A^{-1}(t))$ which can be written $V_A(t)$ with $V_A = V \circ A^{-1}$. An equivalent parametrization is therefore

$$f(x) = \beta + \alpha \int_{A^{-1}(E)} g[\tilde{x}\cdot V_A(t)]\, dt.$$

This formulation shows that the only thing to optimize will be the function V, and the scalars α and β, *getting rid of the optimization of the hidden-to-output weights.* In the remainder of the paper, V_A will simply be denoted V. If we want the domain of integration to be of length 1, we have to make another change of variable $z = \frac{t-t_0}{\alpha}$ where α is the length of E and $t_0 = \inf(A^{-1}(E))$.

$$f(x) = \beta + \alpha \int_0^1 g[\tilde{x}\cdot V(z)]\, dz. \tag{4}$$

Lemma 1. *When V is a piecewise-constant function such that $V(z) = V_i$ when $p_{i-1} \le z < p_i$ with $a_0 = 0$, $\alpha = \sum_i a_i$ and $p_i = \frac{1}{\alpha}\sum_{j=0}^{i} a_j$, we have an ordinary neural network:*

$$\beta + \alpha \int_0^1 g[\tilde{x}\cdot V(z)]\, dz = \beta + \sum_i a_i g(\tilde{x}\cdot V_i) \tag{5}$$

Proof.

$$\begin{aligned}\beta + \alpha \int_0^1 g[\tilde{x}\cdot V(z)]\, dz &= \beta + \alpha \sum_i \int_{p_{i-1}}^{p_i} g[\tilde{x}\cdot V(z)]\, dz \\ &= \beta + \alpha \sum_i (p_i - p_{i-1}) g(\tilde{x}\cdot V_i) \\ &= \beta + \sum_i a_i g(\tilde{x}\cdot V_i)\end{aligned}$$

□

At this point, we can make an important comment. If $x \in \mathbb{R}^d$ we can rewrite $\tilde{x}\cdot V(z) = V^{d+1}(z) + \sum_{i=1}^{d} x_i V^i(z)$ where V^i is a function from $[0,1]$ to $\mathbb{R}$ and x_i is the i-th coordinate of x. But f is a function from K compact of $\mathbb{R}^d$ to $\mathbb{R}$. Using the neural network function approximation theorem of (Hornik, Stinchcombe and White, 1989), the following corollary therefore follows:

Corollary 1. *For every Borel measurable function f from K compact of $\mathbb{R}^d$ to $\mathbb{R}$ and $\forall\varepsilon > 0$, there exist $d+1$ functions V^i from $[0,1]$ to $\mathbb{R}$ and two reals α and β such that $\hat{f}$ defined by*

$$\hat{f}(x) = \beta + \alpha \int_0^1 \tanh\left(\sum_{i=1}^{d} V^i(z)\cdot x_i + V^{n+1}(z)\right) dz$$

achieves $\int_K \|f - \hat{f}\|_\infty < \varepsilon$.

Proof. In this proof, we will define the function $\phi_{V,\alpha,\beta}$:

$$\phi_{V,\alpha,\beta} = x \mapsto \beta + \alpha \int_0^1 \tanh\left(V(z)\cdot x\right)\, dz \tag{6}$$

Let f be an arbitrary Borel Measurable function on a compact set K and $\varepsilon > 0$. By the universal approximation theorem (Hornik, Stinchcombe and White, 1989), we know that there are input weights $v_i, i = 1,\ldots,n$, output weights $a_i, i = 1, \ldots, n$ and an output bias b such that

$$\sup_{x \in K} \left| f(x) - b - \sum_{i=1}^{n} a_i \tanh(v_i \cdot x) \right| < \varepsilon$$

By optionnally replacing v_i by $-v_i$, we can restrict all the a_i to be positive. Defining $\alpha = \sum_i a_i$ and V such that $V(z) = v_i$ if $\frac{\sum_{k=1}^{i-1} a_i}{\alpha} \leq z < \frac{\sum_{k=1}^{i} a_i}{\alpha}$, we have

$$\sup_{x \in K} \left| f(x) - b - \int_{z=0}^{1} \tanh\left(V(z) \cdot x\right) \, dz \right| < \varepsilon$$

Therefore, for all $\varepsilon > 0$, there exists a function V from $[0,1]$ to $\mathbb{R}^{d+1}$ and two reals α and β such that $\sup_{x \in K} |f(x) - \phi(V, \alpha, \beta)| < \varepsilon$.

But as V can be decomposed in $d+1$ functions from $[0,1]$ to $\mathbb{R}$, any Borel measurable function f can, with an arbitrary precision, be defined by

- $d+1$ functions from $[0,1]$ to $\mathbb{R}$
- two scalars α and β.

$\square$

This result is reminiscent of Kolmogorov's superposition theorem (Kolmogorov, 1957), but here we show that the functions V^i can be directly optimized in order to obtain a good function approximation.

2.2 Approximating an Integral

In this work we consider a parametrization of V involving a finite number of parameters, and we optimize over these parameters. Since f is linked to V by an integral, it suggests to look at parametrizations yielding good approximation of an integral. Several such parametrizations already exist:

- piecewise constant functions, used in the rectangle method. This is the simplest approximation, corresponding to ordinary neural networks (eq. 1),
- piecewise affine functions, used in the trapezoid method. This approximation yields better results and will be the one studied here, which we coin "Affine Neural Network".
- polynomials, used in Simpson's method, which allow even faster convergence. However, we were not able to compute the integral of polynomials through the function tanh.

2.3 Piecewise Affine Parametrization

Using a piecewise affine parametrization, we consider V of the form:

$$V(z) = V_{i-1} + \frac{z - p_{i-1}}{p_i - p_{i-1}}(V_i - V_{i-1}) \text{ when } p_{i-1} \leq z < p_i,$$

that is to say V is linear between p_{i-1} and p_i, $V(p_{i-1}) = V_{i-1}$ and $V(p_i) = V_i$ for each i. This will ensure the continuity of V.

In addition, we will set $V_{n+1} = V_1$ to avoid border effects and obtain an extra segment for the same number of parameters.

Rewriting $p_i - p_{i-1} = a_i$ and $V(z) = V_i(z)$ for $p_{i-1} \leq z < p_i$, the output $f(x)$ for an input example x can be written:

$$\begin{aligned} f(x) &= \sum_i \int_{z=p_{i-1}}^{p_i} \tanh\left[V_i(z) \cdot \tilde{x}\right] \, dz \\ f(x) &= \sum_i \frac{a_i}{(V_i - V_{i-1}) \cdot \tilde{x}} \ln\left(\frac{\cosh(V_i \cdot \tilde{x})}{\cosh(V_{i-1} \cdot \tilde{x})}\right) \end{aligned} \quad (7)$$

In the case where $V_i \cdot \tilde{x} = V_{i-1} \cdot \tilde{x}$, the affine piece is indeed constant and the term in the sum becomes $a_i \tanh(V_i \cdot \tilde{x})$, as in a usual neural network. To respect the continuity of function V, we should restrict the a_i to be positive, since p_i must be greater than p_{i-1}.

2.3.1 Is the continuity of V necessary?

As said before, we want to enforce the continuity of V. The first reason is that the trapezoid method uses continuous functions and the results concerning that method can therefore be used for the affine approximation. Besides, using a continuous V allows us to have the same number of parameters for the same number of hidden units. Indeed, using a piecewise affine discontinuous V would require twice as many parameters for the input weights for the same number of hidden units.

The reader might notice at that point that there is no bijection between V and f. Indeed, since V is only defined by its integral, we can switch two different pieces of V without modifying f.

2.4 Extension to multiple output neurons

The formula of the output is a linear combination of the a_i, as in the ordinary neural network. Thus, the extension to l output neurons is straightforward using the formula

$$f_j(x) = \sum_i \frac{a_{ij}}{(V_i - V_{i-1}) \cdot \tilde{x}} \ln\left(\frac{\cosh(V_i \cdot \tilde{x})}{\cosh(V_{i-1} \cdot \tilde{x})}\right) \quad (8)$$

for $j = 1, \ldots, l$.

2.5 Piecewise affine versus piecewise constant

Consider a target function f^* that we would like to approximate, and a target V^* that gives rise to it. Before going any further, we should ask two questions:

- is there a relation between the quality of the approximation of f^* and the quality of approximation of V^*?
- are piecewise affine functions (i.e. the affine neural networks) more appropriate to approximate an arbitrary function than the piecewise constant ones (i.e. ordinary neural networks)?

Using the function ϕ defined in equation 6, we have:

Theorem 1. *$\forall x, \forall V^*, \forall \alpha^*, \forall \beta^*, \forall V$, we have*

$$|\psi(x) - \psi^*(x)| \leq 2\alpha^* \int_0^1 \tanh\left(|(V(z) - V^*(z)) \cdot \tilde{x}|\right) dz \tag{9}$$

with $\psi = \phi_{V,\alpha^*,\beta^*}$ and $\psi^* = \phi_{V^*,\alpha^*,\beta^*}$.

The proof, which makes use of the bound on the function tanh, is omitted for the sake of simplicity. Thus, if V is never far from V^* and x is in a compact set K, we are sure that the approximated function will be close to the true function. This justifies the attempts to better approximate V^*.

We can then make an obvious remark: if we restrict the model to a finite number of hidden neurons, it will never be possible to have a piecewise constant function equal to a piecewise affine function (apart from the trivial case where all affine functions are in fact constant). On the other hand, any piecewise contant function composed of h pieces can be represented by a continuous piecewise affine function composed of at most $2h$ pieces (half of the pieces being constant and the other half being used to avoid discontinuities), given that we allow vertical pieces (which is true in the affine framework).

Are affine neural networks providing a better parametrization than the ordinary neural networks? The following theorem suggests it:

Theorem 2. *Let $f^* = \phi_{V^*,\alpha^*,\beta^*}$ with V^* a function with a finite number of discontinuities and C^1 on each interval between two discontinuities. Then there exists a scalar C, a piecewise affine continuous function V with h pieces and two scalars α and β such that, for all x, $|\phi_{V,\alpha,\beta}(x) - f^*(x)| \leq Ch^{-2}$ (pointwise convergence).*

Proof. Let V^* be an arbitrary continuous function on $[p_{i-1}, p_i]$. Then, choosing the constant function $V : z \mapsto \dfrac{V^*(p_{i-1}) + V^*(p_i)}{2}$ yields for all z in $[p_{i-1}, p_i]$:

$$|V^*(z) - V(z)| \leq \frac{p_i - p_{i-1}}{2} M^1(V^*, [p_{i-1}, p_i]) \tag{10}$$

where $M^1(V, I) = \max_{z\in I} |V'(z)|$ ($M^1(V, I)$ is the maximum absolute value of the first derivative of V on the interval I).

Now let V^* be a function in C^1 (that is, a function whose derivative is continuous everywhere) and choose the affine function $V : z \mapsto V^*(p_{i-1}) + \dfrac{z - p_{i-1}}{p_i - p_{i-1}}[V^*(p_i) - V^*(p_{i-1})]$. The trapezoid method tells us that the following inequality is verified:

$$|V^*(z) - V(z)| \leq \frac{(z - p_{i-1})(p_i - z)}{2} M^2(V^*, [p_{i-1}, p_i])$$

where $M^2(V, I) = \max_{z\in I} |V''(z)|$ ($M^2(V, I)$ is the maximum absolute value of the second derivative of V on the interval I). Using the fact that, for all z in $[p_{i-1}, p_i]$, $(z - p_{i-1})(p_i - z) \leq \frac{(p_i - p_{i-1})^2}{4}$, we have

$$|V^*(z) - V(z)| \leq \frac{(p_i - p_{i-1})^2}{8} M^2(V^*, [p_{i-1}, p_i]) \tag{11}$$

Moreover, theorem 1 states that

$$|\psi(x) - \psi^*(x)| \leq 2|\alpha^*| \int_{z=0}^1 \tanh\left(|(V(z) - V^*(z)) \cdot \tilde{x}|\right) dz$$

with $\psi = \phi_{V,\alpha^*,\beta^*}$ and $\psi^* = \phi_{V^*,\alpha^*,\beta^*}$. Using

- $\int_0^1 \tanh(|q(z)|)\, dz \leq \sup_{[0,1]} \tanh(|q(z)|)$
- $\tanh(|q(z)|) \leq |q(z)|$

we have

$$|\psi(x) - \psi^*(x)| \leq 2|\alpha^*| \sup_{[0,1]} |(V(z) - V^*(z)) \cdot \tilde{x}| \tag{12}$$

$$\begin{aligned}
|\psi(x) - \psi^*(x)| &\leq 2|\alpha^*| \sup_{[0,1]} |(V(z) - V^*(z)) \cdot \tilde{x}| \\
&\leq 2|\alpha^*| \sup_{[0,1]} \left| \sum_i x_i (V_i(z) - V_i^*(z)) \right| \\
&\leq 2|\alpha^*| \sum_i |x_i| \sup_{[0,1]} |V_i(z) - V_i^*(z)|
\end{aligned}$$

In the case of a piecewise constant function, this inequality becomes:

$$\begin{aligned}
|\psi(x) - \psi^*(x) &\leq 2|\alpha^*| \sum_i |x_i| \sup_j \frac{p_j - p_{j-1}}{2} M_j^1 \\
|\psi(x) - \psi^*(x)| &\leq |\alpha^*| M^1 \sum_i |x_i| \sup_j (p_j - p_{j-1})
\end{aligned}$$

where M_j^1 is the maximum absolute value of the derivative of the function V_i^* on the interval $[p_{j-1}, p_j]$ and M^1 is the same for the whole interval $[0, 1]$.

In the case of a piecewise affine function, this inequality becomes:

$$
\begin{aligned}
|\psi(x) - \psi^*(x)| &\leq 2|\alpha^*| \sum_i |x_i| \sup_j \frac{(p_j - p_{j-1})^2}{8} M_j^2 \\
|\psi(x) - \psi^*(x)| &\leq \frac{|\alpha^*| M^2}{4} \sum_i |x_i| \sup_j (p_j - p_{j-1})^2
\end{aligned}
$$

where M^2 is the equivalent of M^1 for the second derivative.

If V^* has a k discontinuities and V has h pieces (corresponding to a neural network with h hidden neurons), we can place $p_{d,i}$ at the i-th discontinuity and split each interval between two discontinuities into $\frac{h}{k}$ pieces. The maximum value of $p_j - p_{j-1}$ is thus lower than $\frac{k}{h}$ (since the p_j are between 0 and 1).

We thus have the following bounds:

$$|\phi(V_C, \alpha^*, \beta^*)(x) - \phi(V^*, \alpha^*, \beta^*)(x)| \leq \frac{C_1}{h} \quad (13)$$

$$|\phi(V_A, \alpha^*, \beta^*)(x) - \phi(V^*, \alpha^*, \beta^*)(x)| \leq \frac{C_2}{h^2} \quad (14)$$

where V_C is a piecewise constant function and V_A is a piecewise affine function. C_1 and C_2 are two constants (the x_i are bounded since we are on a compact).

This concludes the proof. □

This theorem means that, if we try to approximate a function f^* verifying certain properties, as the number of hidden units of the network grows, an affine neural network will converge faster to f^* (for each point x) than an ordinary neural network. An interesting question would be to characterize the set of such functions f^*. It seems that the answer is far from trivial.

Besides, one must note that these are upper bounds. It therefore does not guarantee that the optimization of affine neural networks will always be better than the one of ordinary neural networks. Furthermore, one shall keep in mind that both methods are subject to local minima of the training criterion. However, we will see in the following section that the affine neural networks appear less likely to get stuck during the optimization than ordinary neural networks.

2.6 Implied prior distribution

We know that gradient descent or ordinary $\mathcal{L}^2$ regularization are more likely to yield small values of the parameters, the latter directly corresponding to a zero-mean Gaussian prior over the parameters. Hence to visualize and better understand this new parametrization, we define a zero-mean Gaussian prior distribution on the parameters β, a_i and V_i ($1 \leq i \leq h$), and sample from the corresponding functions.

We sampled from each of the two neural networks (ordinary discrete net and continuous affine net) with one input neuron, one output neuron, and different numbers of hidden units (n_h pieces) and zero-mean Gaussian priors with variance σ_u for input-to-hidden weights, variance σ_a for the input biases, variance σ_b for the output bias and variance $\frac{w_v}{\sqrt{h}}$ for hidden-to-output weights (scaled in terms of the number of hidden units h, to keep constant the variance of the network output as h changes). Randomly obtained samples are shown in figure 1. The x axis represents the value of the input of the network and the y axis is the associated output of the network. The different

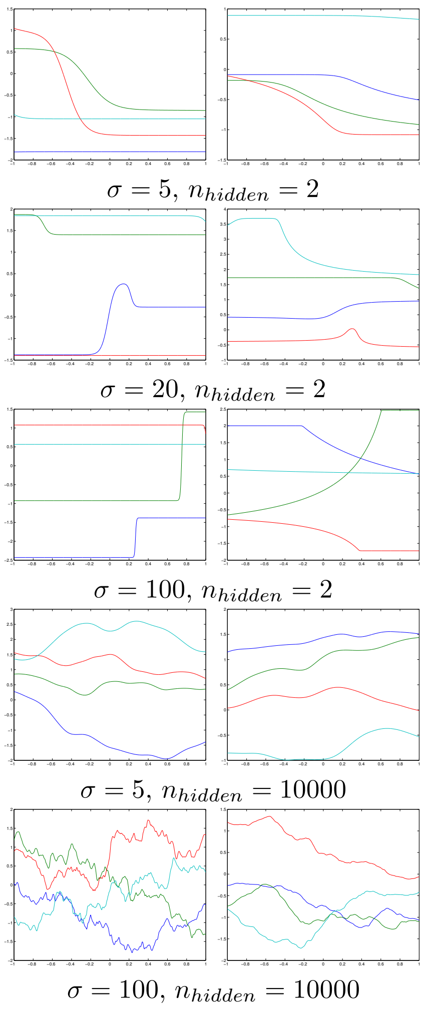

Figure 1: Functions generated by an ordinary neural network (left) and an affine neural network (right) for various Gaussian priors over the input weights and various number of hidden units.

priors over the functions show a very specific trend: when σ_u grows, the ordinary (tanh) neural network tends to saturate whereas the affine neural network does so much less often. This can easily be explained by the fact that, if $|V_i \cdot \tilde{x}|$ and $|V_{i+1} \cdot \tilde{x}|$ are both much greater than 1, hidden unit i stays in the saturation zone when V is piecewise constant (ordinary neural network). However, with a piecewise affine V, if $V_i \cdot \tilde{x}$ is positive and $V_{i+1} \cdot \tilde{x}$ is negative (or the opposite), we will go through the non-saturated zone, in which gradients on V_i and V_{i+1} flow well compared to ordinary neural networks. This might yield easier optimization of input-to-hidden weights, even though their value is large.

3 Non-Parametric Continous Neural Networks

This section returns on the strong link between kernel methods and continuous neural networks, first presented in (Neal, 1994). It also exhibits a clear connection with Gaussian processes, with a newly motivated kernel formula. Here, we start from eq. 2 but use as an index u the elements of $\mathbb{R}^{d+1}$ themselves, i.e. V is completely free and fully non-parametric: we integrate over all possible weight vectors.

To make sure that the integral exists, we select a set E over which to integrate, so that the formulation becomes

$$f(x) = \int_E a(u)g(x \cdot u)\, du \qquad (15)$$

$$= < a, g_x > \qquad (16)$$

with $<,>$ the usual dot product of $\mathcal{L}^2(E)$ and g_x the function such that $g_x(u) = g(x \cdot u)$.

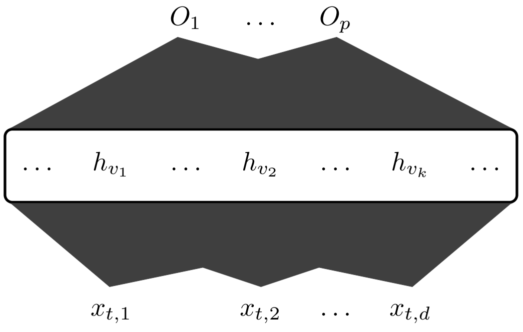

Figure 2: Architecture of a continuous neural network

3.1 $\mathcal{L}^1$-norm Output Weights Regularization

Although the optimization problem becomes convex when the $\mathcal{L}^1$-norm of a is penalized, it involves an infinite number of variables. However, we are guaranteed to obtain a finite number of hidden units with non-zero output weight, and both exact and approximate optimization algorithms have been proposed for this case in (Bengio et al., 2005). Since this case has already been well treated in that paper, we focus here on the $\mathcal{L}^2$ regularization case.

3.2 $\mathcal{L}^2$-norm Output Weights Regularization

In some cases, we know that the optimal function a can be written as a linear combination of the g_{x_i} with the x_i's the training examples. For example, when the cost function is of the form $c\left((x_1, y_1, f(x_1)), ..., (x_m, y_m, f(x_m))\right) + \Omega(\|f\|_{\mathcal{H}})$ with $\|f\|_{\mathcal{H}}$ is the norm induced by the kernel k defined by $k(x_i, x_j) =< g_{x_i}, g_{x_j} >$, we can apply the representer theorem (Kimeldorf and Wahba, 1971).

It has been known for a decade that, with Gaussian priors over the parameters, a neural network with a number of hidden units growing to infinity converges to a Gaussian process (chapter 2 of (Neal, 1994)). However, Neal did not compute explicitly the covariance matrices associated with specific neural network architectures. Such covariance functions have already been analytically computed (Williams, 1997), for the cases of sigmoid and Gaussian transfer functions. However, this has been done using a Gaussian prior on the input-to-hidden weights. The formulation presented here corresponds to a uniform prior (i.e. with no arbitrary preference for particular values of the parameters) when the transfer function is sign. The sign function has been used in (Neal, 1994) with a Gaussian prior on the input-to-hidden weights, but the explicit covariance function could not be computed. Instead, approximating locally the Gaussian prior with a uniform prior, Neal ended up with a covariance function of the form $k(x, y) \approx A - B||x - y||$. We will see that, using a uniform prior, this is exactly the form of kernel one obtains.

3.3 Kernel when g is the sign Function

Theorem 3. *A neural network with an uncountable number of hidden units, a uniform prior over the input weights, a Gaussian prior over the output weights and a* sign *transfer function is a Gaussian process whose kernel is of the form*

$$k(x_i, x_j) = 1 - \theta\|x_i - x_j\| \qquad (17)$$

Such a kernel can be made hyperparameter-free for kernel regression, kernel logistic regression or SVMs.

Proof. For the sake of shorter notation, we will denote the sign function by s and warn the reader not to get confused with the sigmoid function.

We wish to compute

$$k(x,y) =< g_x, g_y >= E_{v,b}\left[s(v\cdot x+b)s(v\cdot y+b)\right].$$

Since we wish to define a uniform prior over v and b, we cannot let them span the whole space ($\mathbb{R}^n$ in the case of v and $\mathbb{R}$ in the case of b). However, the value of the function sign does not depend on the norm of its argument, so we can restrict ourselves to the case where $\|v\| = 1$. Furthermore, for values of b greater than δ, where δ is the maximum norm among the samples, the value of the sign will be constant to 1 (and -1 for opposite values of b). Therefore, we only need to integrate b on the range $[-\delta, \delta]$.

Defining a uniform prior on an interval depending on the training examples seems contradictory. We will see later that, as long as the interval is big enough, its exact value does not matter.

Let us first consider v fixed and compute the expectation over b. The product of two sign functions is equal to 1 except when the argument of one sign is positive and the other negative. In our case, this becomes:

$$\left\{\begin{array}{lcl} v\cdot x+b & < & 0\\ v\cdot y+b & > & 0\end{array}\right. \text{ or } \left\{\begin{array}{lcl} v\cdot x+b & > & 0\\ v\cdot y+b & < & 0\end{array}\right.$$

which is only true for b between $-\min(v\cdot x, v\cdot y)$ and $-\max(v\cdot x, v\cdot y)$, which is an interval of size $|v\cdot x - v\cdot y| = |v\cdot(x-y)|$.

Therefore, for each v, we have

$$\begin{aligned} E_b\left[s(v\cdot x+b)s(v\cdot y+b)\right] &= \frac{(2\delta - 2\,|v\cdot(x-y)|)}{2\delta}\\ &= 1 - \frac{|v\cdot(x-y)|}{\delta}.\end{aligned}$$

We must now compute

$$k(x,y) = 1 - E_v\left[\frac{|v\cdot(x-y)|}{\delta}\right] \tag{18}$$

It is quite obvious that the value of the second term only depends on the norm of $(x-y)$ due to the symmetry of the problem. The value of the kernel can thus be written

$$k(x,y) = 1 - \theta\|x-y\| \tag{19}$$

Using the surface of the hypersphere S_d to compute θ, we find

$$k(x,y) = 1 - \frac{\sqrt{2}}{\delta\sqrt{(d-1)\pi}}\|x-y\| \tag{20}$$

with d the dimensionality of the data and δ the maximum $\mathcal{L}^2$-norm among the samples. The coefficient in front of the term $\|x-y\|$ has a slightly different form when $d=1$ or $d=2$.

Let us now denote by K the matrix whose element (i,j) is $K(x_i, x_j)$. The solution in kernel logistic regression, kernel linear regression and SVM is of the form $K\alpha$ where α is the weight vector. It appears that the weight vector is orthogonal to $e = [11\ldots1]'$.

Thus, adding a constant value c to every element of the covariance matrix changes the solution from $K\alpha$ to $(K+cee')\alpha = K\alpha + ee'\alpha = K\alpha$.

Therefore, the covariance matrix is defined to an additive constant.

Besides, in kernel logistic regression, kernel linear regression and SVM, the penalized cost is of the form

$$C(K,\lambda,\alpha) = \mathcal{L}(K\alpha, Y) + \lambda\alpha' K\alpha$$

We can see that $C(K,\lambda,\alpha) = C(cK, c\lambda, \frac{\alpha}{c})$. Thus, multiplying the covariance matrix by a constant c and the weight decay by the same constant yields an optimal solution α^* divided by c. However, the product $K\alpha$ remains the same.

In our experiments, the value of the weight decay had very little influence. Furthermore, the best results have always been obtained for a weight decay equal to 0. This means that no matter the multiplicative factor by which K is multiplied, the result will be the same.

This concludes the proof that this kernel can be made hyperparameter-free. $\square$

3.3.1 Function sampling

As presented in (Neal, 1994), the functions generated by this Gaussian process are Brownian (see figure 3).

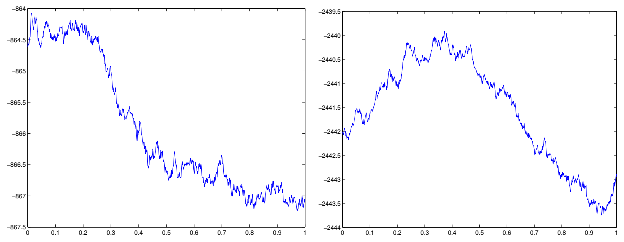

Figure 3: Two functions drawn from the Gaussian process associated to the above kernel function.

3.3.2 Experiments on the USPS dataset

We tried this new hyper-parameter free kernel machine on the USPS dataset, with quadratic cost to evaluate its stability. We optimized the hyperparameters of the

Gaussian kernel on the test set (optimization on the validation set yields 4.0% test error). As there are no hyperparameters for the sign kernel, this clearly is in favor of the Gaussian kernel. We can see in table 1 that the Gaussian kernel is much more sensitive to hyperparameters, whereas the performance of the sign kernel is the same for all values of λ. We show the mean and the deviation of the error over 10 runs.

Algorithm	$\lambda = 10^{-3}$	$\lambda = 10^{-12}$	Test
K_{sign}	2.27±0.13	**1.80±0.08**	4.07
G. $\sigma = 1$	**58.27±0.50**	58.54±0.27	58.29
G. $\sigma = 2$	**7.71±0.10**	7.78±0.21	12.31
G. $\sigma = 4$	**1.72±0.11**	2.10±0.09	4.07
G. $\sigma = 6$	**1.67*±0.10**	3.33±0.35	3.58*
G. $\sigma = 7$	**1.72±0.10**	4.39±0.49	3.77

Table 1: sign kernel *vs* Gaussian kernel on USPS dataset with 7291 training samples, with different Gaussian widths σ and weight decays λ. For each kernel, the best value is in bold. The star indicates the best overall value. The first two columns are validation error ± dev. The last one is the test error.

3.3.3 LETTERS dataset

Similar experiments have been performed on the LETTERS dataset. Again, whereas the sign kernel does not get the best overall result, it performs comparably to the best Gaussian kernel (see table 2).

Algorithm	$\lambda = 10^{-3}$	$\lambda = 10^{-9}$	Test
K_{sign}	5.36 ± 0.10	**5.22 ± 0.09**	5.5
G. $\sigma = 2$	**5.47 ± 0.14**	5.92 ± 0.14	5.8
G. $\sigma = 4$	**4.97* ± 0.10**	12.50 ± 0.35	5.3*
G. $\sigma = 6$	**6.27 ± 0.17**	17.61 ± 0.40	6.63
G. $\sigma = 8$	**8.45 ± 0.19**	18.69 ± 0.34	9.25

Table 2: sign kernel *vs* Gaussian kernel on LETTERS dataset with 6000 training samples, with different Gaussian widths σ and weight decays λ. For each kernel, the best value is in bold. The best overall value is denoted by a star. The first two columns are validation error ± standard deviation. The last one is test error for λ that minimizes validation error.

4 Conclusions, Discussion, Future Work

We have studied in detail two formulations of uncountable neural networks, one based on a finite parametrization of the input-to-hidden weights, and one that is fully non-parametric. The first approach delivered a number of interesting results: a new function approximation theorem, an affine parametrization in which the integrals can be computed analytically, and an error bound theorem that suggests better approximation properties than ordinary neural networks.

As shown in theorem 1, function V can be represented as $d+1$ functions from $\mathbb{R}$ to $\mathbb{R}$, easier to learn than one function from $\mathbb{R}^{d+1}$ to $\mathbb{R}$. We did not find parametrizations of those functions other than the continuous piecewise affine one with the same feature of analytic integration. To obtain smooth functions V with restricted complexity, one could set the functions V to be outputs of another neural network taking a discrete index in argument. However, this has not yet been exploited and will be explored in future work.

The second, non-parametric, approach delivered another set of interesting results: with sign activation functions, the integrals can be computed analytically, and correspond to a hyperparameter-free kernel machine that yields performances comparable to the Gaussian kernel. These results raise a fascinating question: why are results with the sign kernel that good with no hyper-parameter and no regularization? To answer this, we should look at the shape of the covariance function $k(x, y) = 1 - \theta\|x - y\|$, which suggests the following conjecture: it can discriminate between neighbors of a training example while being influenced by remote examples, whereas the Gaussian kernel does either one or the other, depending on the choice of σ.

References

Bengio, Y., Le Roux, N., Vincent, P., Delalleau, O., and Marcotte, P. (2005). Convex neural networks. In *Advances in Neural Information Processing Systems.*

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

Kimeldorf, G. and Wahba, G. (1971). Some results on tchebychean spline functions. *Journal of Mathematics Analysis and Applications*, 33:82–95.

Kolmogorov, A. (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Kokl. Akad. Nauk USSR*, 114:953–956.

Neal, R. (1994). *Bayesian Learning for Neural Networks.* PhD thesis, Dept. of Computer Science, University of Toronto.

Williams, C. (1997). Computing with infinite networks. In Mozer, M., Jordan, M., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9.* MIT Press.