Neural Networks - A Continuum of Potential

Eric Hu
McGill University
Montreal, QC
eric.hu@mail.mcgill.ca

Johnny Huang
McGill University
Montreal, QC
lihui.huang@mail.mcgill.ca

Viet Nguyen
McGill University
Montreal, QC
baviet.nguyen@mail.mcgill.ca

Abstract

Theories on neural networks with an infinite number of hidden units were explored since the late 1990's [3] [4] [13], deepening the understanding of these computational models in two principal aspects: 1. network behavior under some limiting process on its parameters and 2. neural networks from a functional perspective [10]. Continuous neural networks are of particular interest due to their computational feasibility from finite affine parametrizations and strong convergence properties under constraints. In this paper, we survey some of the theoretical groundings of continuous networks, notably their correspondence with Gaussian processes, computation and applications through the Neural Tangent Kernel (NTK) formulation, and apply the infinite dimensional extension to inputs and outputs all the while considering their universal approximation properties.

1 Affine neural networks

Ever since its popularization by authors of [12] [8], and the more recent works [7], the artificial neural network as a supervised learning model has proven to be a powerful tool due to its ability to learn complex correspondences between data and labels as well as strong convergence properties, bolstered by a variety of computational architectures for an even wider variety of applications. Continuous networks, a subclass of neural networks, are characterized by their continuum of hidden units instead of a finite number [11] [3]. Beyond the theoretical studies of continuous neural networks lie feasible and applicable computational models with stronger convergence guarantees on constrained approximation objectives than their finite counterparts.

1.1 Neural network representation

Given a finite dataset $\mathcal{D} = \{(x^i, y^i)\}_{i=1,2,...,n}$, $x^i = (x^i_1, ..., x^i_m)^T \in \mathcal{X}, y^i \in \mathcal{Y}$, neural networks leverage the descriptive potential of linear combinations of perceptrons arranged in a multi-layered structure to infer complex and general correspondences $f : \mathcal{X} \longrightarrow \mathcal{Y}$ from samples in $\mathcal{D}$. We interest ourselves in neural networks with an uncountable number of perceptrons approximating a function, focusing particularly on alternative finite neural networks that performs the same approximation. Given an infinite network $f : \mathcal{X} \longrightarrow \mathcal{Y}$, one seeks a family of finite neural networks $\mathcal{F} = \{f_\alpha\}_\mathcal{A}$ such that $\forall \alpha \in \mathcal{A}, ||f_\alpha - f||_\infty < \epsilon$ for any $\epsilon > 0$. We restrict ourselves to neural networks with a single continuous hidden layer and a single output perceptron, sufficient conditions to approximate any continuous real-valued function guaranteed by the universal approximation theorem [4]. Let $W \in \mathbf{R}^{d \times m}$ where d is the number of hidden perceptrons, $g : \mathbf{R} \longrightarrow \mathbf{R}$ be an activation function, $a \in \mathbf{R}^d$ the hidden-to-output weight vector, and $\beta \in \mathbf{R}$ a bias. A neural network with the above setting is a function $f : \mathbf{R}^m \longrightarrow \mathbf{R}$ such that on input x, returns:

$$f(x) = \beta + \sum_{i=1}^{d} a_i g(W_i \cdot x)$$

where W_i is the i^{th} column of W. Here, the number of hidden perceptrons are determined by the number of rows d of W. If $W = (w_i)$ where w_i is the i^{th} column of W, consider the function $W' : \{1, 2, ..., d\} \longrightarrow \mathbf{R}^m$ such that $W'(n) = w_n^T$, the n^{th} row of W. Thus, $f(x) = \beta + \sum_{i=1}^{d} a_i g(W'(i) \cdot x)$. In light of this alternative representation, we extend the domain of W' to a continuum, effectively making the number of hidden units uncountable.

Definition 1.1. (Continuous neural network) Given the above setting, let $E \subset \mathbf{R}$ compact with positive Lebesgue measure, $W : E \longrightarrow \mathbf{R}^m$ be a weight map, $a : E \longrightarrow \mathbf{R}$ be a hidden-to-output map, $K \subset \mathbf{R}^m$ compact, a continuous neural network is a continuous function $f : K \longrightarrow \mathbf{R}$ such that on input x, computes:

$$f(x) = \beta + \int_E a(u) \cdot g(W(u) \cdot x) du$$

For the sake of simplicity, we choose to ignore layer biases, which justifies the choice of the codomains of W. In practice, the bias is included by changing the codomain of the weight map to $W : E \longrightarrow R^{m+1}$, and append a 1 to every input vector x, thus $x \in R^{m+1}$. One can think of the above as indexing the hidden layer by a closed continuum. Analogous to the Kolmogorov-Arnold representation theorem, continuous neural networks can be represented by a finite number of continuous functions, which then in turn can be approximated to arbitrary precision by finite neural networks in the uniform topology.

1.2 Universal approximation theorem for continuous neural networks

On top of being evidence of expressibility of continuous neural networks, the continuous universal approximation theorem also demonstrates the feasibility of computations of these networks' outputs.

Finite-dimensional parametrization Given a continuous neural network f as defined previously, let $A \in \int a(u)du$ be a primitive of a, injective. In fact, we can consider integrating over the support of a since $\forall u \in E$ such that $a(u) = 0$, u does not contribute to the integral. Moreover, we can make the restriction $a(u) > 0$ since changing the sign of $a(u)$ is equivalent to changing the sign of $W(u)$. Thus, making the change of variables $t = A(u), dt = a(u)du$, we have that $W(u) = W(A^{-1}(t))$. Redefining $W(t) := (W \circ A^{-1})(t)$, we have:

$$f(x) = \beta + \int_{A^{-1}(E)} g(W(t) \cdot x) dt$$

Normalizing the domain of integration, we make the change of variables $z = \frac{t - t_0}{\alpha}, dz = \frac{1}{\alpha} dt$, where α is the length of the closed interval E, and $t_0 = \inf A^{-1}(E)$, yields:

$$f(x) = \beta + \alpha \int_0^1 g(W(z) \cdot x) dz$$

leaving α, β, and W the only parameters left to optimize.

Lemma 1.2. (Alternative finite neural network) When W is a piecewise-constant function such that $W(z) = W_i$ when $z \in [p_{i-1}, p_i), a_0 = 0, \alpha = \sum_i a_i$, and $p_i = \sum_{j=0}^{i} a_j$, there exists a finite neural network that computes like a continuous neural network:

$$\beta + \alpha \int_0^1 g(W(z) \cdot x) dz = \beta + \sum_i a_i g(W_i \cdot x)$$

Proof.

$$\begin{aligned}\beta + \alpha \int_0^1 g(W(z)\cdot x)dz &= \beta + \alpha \sum_i \int_{p_{i-1}}^{p_i} g(W(z))dz \\ &= \beta + \alpha \sum_i (p_i - p_{i-1}) g(W_i \cdot x) \\ &= \beta + \sum_i a_i g(W_i \cdot x) = f(x)\end{aligned}$$

∎

Remark 1.3. We notice that the $m-$dimensional product $W(z)\cdot x$ can be written as $\sum_{i=1}^m W^i(z)x_i$, where $W^i : [0,1] \longrightarrow \mathbf{R}$, where $W^i(z) = \langle W(z), \hat{e}_i\rangle, \hat{e}_i$ being the i^{th} standard orthonormal basis in $\mathbf{R}^m$. In other words, the vector-valued function W can be entirely described by m scalar-valued functions W^i.

Continuous universal approximation The following is largely due to Le Roux and Bengio [11].

Theorem 1.4. (Continuous universal approximation theorem) Let $K \subset \mathbf{R}^m$ a compact domain. For all $f : K \longrightarrow \mathbf{R}$ measurable, and for all $\epsilon > 0$, there exists m functions $\{W^i : [0,1] \longrightarrow \mathbf{R}\}_{i=1,2,...,m}$, and $\alpha, \beta \in \mathbf{R}$, such that $\hat{f}$ defined by

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

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

We note to the reader that we use the hyperbolic tangent as our default activation function. However, the original formulation [4] is true for all activation functions of the form $g : \mathbf{R} \longrightarrow [0,1]$, non-decreasing, such that $\lim_{\lambda\longrightarrow\infty} g(\lambda) = 1$, and $\lim_{\lambda\longrightarrow-\infty} g(\lambda) = 0$.

Proof. (Idea) The proof follows directly from the statement of the universal approximation theorem for finite networks [4], and by defining a piecewise-constant function W over the normalized domain $[0,1]$ accordingly, we can reformulate the approximation theorem for finite networks with our constructed infinite network copy. From 1.3, it follows that any measurable function f can be defined ϵ-precisely by m functions $\{W^i : [0,1] \longrightarrow \mathbf{R}\}_{i=1,2,...,m}$ and $\alpha, \beta \in \mathbf{R}$. ∎

Remark 1.5. A formal proof of this type is given in 4.6

1.3 Convergence guarantees of piecewise affine networks

Beyond the realm of piecewise constant functions to approximate continuous neural networks is the use of piecewise affine parametrizations, which grants continuity of W and as we shall see in this subsection, could converge faster to a target function than a piecewise constant neural network, for a certain class of target functions.

Definition 1.6. (Affine neural networks) A continuous neural network is affine if $W : [0,1] \longrightarrow \mathbf{R}^m$ is of the form:

$$W(z) = W_{i-1} + \frac{z - p_{i-1}}{p_i - p_{i-1}}(W_i - W_{i-1})$$

when $z \in [p_{i-1}, p_i), W(p_{i-1}) = W_{i-1}, W(p_i) = W_i$. This is to say that W is linear between p_{i-1} and p_i, which ensures its continuity over its domain. We call W affine.

Notation 1.7. We define the following function as a shorthand:

$$\phi_{W,\alpha,\beta} = \beta + \alpha \int_0^1 \tanh(W(z)\cdot x)dz$$

Now, consider a target function f^* to approximate, and W^* such that $f^* = \phi_{W^*,\alpha^*,\beta^*}$, i.e. approximating f^* is the same as finding the weight function W^*. Then, not only are our approximations W never far from W^*, but for a certain class of target functions f^*, affine networks are sure to converge faster than piecewise constant networks, stated by the following theorems primarily due to [11]:

Theorem 1.8. Let $\psi = \phi_{W,\alpha^*,\beta^*}, \psi^* = \phi_{W^*,\alpha^*,\beta^*}$. $\forall x \in K \subset \mathbf{R}^m$ compact, $\forall W^*, \forall\alpha^*, \forall\beta^*, \forall W$, we have:

$$|\psi(x) - \psi^*(x)| \leq 2\alpha^* \int_0^1 \tanh(|(W(z) - W^*(z)) \cdot x|)dz$$

This theorem justifies attempts to optimize W^* since given any initialization of W, we are given an upper bound on how pointwise close we are to the target function ψ^*.

Proof. (Huang) We complete the proof by using the properties of the $\tanh$ function.

$$\begin{aligned}
|\psi(x) - \psi^*(x)| &= |(\beta^* + \alpha^* \int_0^1 \tanh(W(z) \cdot x)dz) - (\beta^* + \alpha^* \int_0^1 \tanh(W^*(z) \cdot x)dz)| \\
&\leq \alpha^* \int_0^1 |\tanh(W(z) \cdot x) - \tanh(W^*(z) \cdot x)|dz \\
&\leq \alpha^* \int_0^1 |\tanh(W(z) \cdot x - W^*(z) \cdot x) \cdot (1 - \tanh(W(z) \cdot x) \tanh(W^*(z) \cdot x))|dz \\
&\leq 2\alpha^* \int_0^1 \tanh(|(W(z) - W^*(z)) \cdot x|)dz
\end{aligned}$$

The factor 2 is derived by using the bound of the $\tanh$ function. Since $\forall x \in \mathbf{R}, -1 < \tanh(x) < 1$, we have that $\forall a, b \in \mathbf{R}$, an upper bound of $(1 - \tanh(a)\tanh(b))$ is 2. ■

The following makes a statement on the optimization of affine neural networks when conditions are imposed on the target function.

Theorem 1.9. Let $f^* = \phi_{W^*,\alpha^*,\beta^*}$ such that W^* is a with a finite number of discontinuities, and continuously differentiable on them. Then, $\exists C \in \mathbf{R}$, W affine with h pieces, and $\alpha, \beta \in \mathbf{R}$ such that $\forall x \in K, |\phi_{W,\alpha,\beta} - f^*(x)| \leq \frac{C}{h^2}$.

That is to say that $\phi_{W,\alpha,\beta}$ converges pointwise faster than their piecewise constant alternatives, which achieves an upper bound of $\frac{C'}{h}$ for some $C' \in \mathbf{R}$. However, since the authors have not explicitly described the bounds on constants C, C', it is highly possible that the discrepancy between these constants nullifies the theoretical advantage of the inverse squared term in the error bounds in the theorem. Furthermore, finding this class of target functions satisfying the above theorem is non-trivial, which suggests novel research directions:

- Analytic description of the class of functions $\mathcal{F} = \{f^*_\alpha\}_\mathcal{A}$ such that $f^* \in \mathcal{F}$ satisfies 1.9.
- The extent to which affine parametrization is computationally feasible.
- A quantization scheme for the expected gain from using an affine parametrization for continuous networks.

2 The Gaussian process correspondence

An important approach to avoid over-fitting for neural networks is to introduce priors for the output function and decrease the complexity of the model learned from training dataset. Bayesian learning methods make use of a prior probability that gives lower probability to more complex models. In this section we study the strong connection between Gaussian processes and infinitely wide continuous neural networks. Neal's thesis [3] introduced the idea that a single-layer fully-connected neural network with an i.i.d. prior over its parameters can be interpreted as a Gaussian process, as the number of hidden units increases to infinity in the limit. Later on, an equivalence between *deep* neural networks and Gaussian processes are developed [9].

2.1 Gaussian processes

One can understand Gaussian processes as an extension of a multivariate normal distribution over some continuous index. In probability theory and statistics, a Gaussian process is a stochastic process (a collection of random variables indexed by time or space), such that every finite collection of those random variables has a multivariate normal distribution, i.e. every finite linear combination of them is normally distributed. Gaussian processes are used to infer a distribution over functions, along with a mean function $\mu(x)$ and a covariance function $K(x, x')$, denoted as $f \sim GP(\mu, K)$. For each input x there is a corresponding random variable $f(x)$. The Gaussian process implies that for a given set of finite inputs $(x_1, x_2, ..., x_n)$, the random values $(f(x_1), f(x_2), ..., f(x_n))$ corresponds to a multivariate normal distribution.

2.2 Correspondence

We start by introducing the correspondence between neural networks and Gaussian processes for the case of one hidden layer as the number of hidden perceptrons approaches infinity. Here we use a new notation that focuses on the contribution of each hidden perceptron to the output of the neural network. Suppose that the d is the number of hidden perceptrons and ϕ is a pointwise nonlinear activation function. Let $x \in \mathbf{R}^m$ be the input, $z \in \mathbf{R}^n$ be the output, where m and n are the dimension of the input and output respectively. The weight for the input to hidden layer is W^0 and the weight for the hidden layer to output is W^1. For each component z_i of the output, let the bias be b_i. For the independence and randomnesses of the weights and bias, we can assume that the mean for each hidden-to-output weight and bias is zero, with variance $\sigma^2_{W^1}/d$ and σ^2_b.

Now we can compute the i^{th} component of the output with respect to the input x:

$$z_i(x) = b_i + \sum_{j=1}^{d} W^1_{ij} x^1_j(x)$$

where $x^1_j(x)$ represents the post-activation transformation of each perceptrons:

$$x^1_j(x) = \phi(\sum_{k=1}^{m} W^0_{jk} x_k)$$

Here we still ignore the layer biases for the sake of simplicity. Under the i.i.d assumption, all the components of the weights and inputs are independent, which implies that the post linear activation outputs $x^1_j(x)$ are independent, and they all have identical distribution. Thus $z_i(x)$ is the sum of i.i.d terms. As the width of the hidden layer d approaches infinity(i.e. $d \to \infty$), we can conclude from the Central Limit Theorem that $z_i(x)$ will converge to a Gaussian distribution. As we provide a finite collection of inputs $(x_{(1)}, x_{(2)}, ..., x_{(p)})$, the collection of outputs $(z_{(1)}, z_{(2)}, ..., z_{(p)})$ will converge to a multivariate normal distribution. Since we considered every input to be in $\mathbf{R}^m$, we obtain a Gaussian process, where the "function" we are trying to approximate is z_i.

Therefore we conclude that $z_i \sim GP(\mu(x), k(x, x'))$ The mean for z_i and the covariance function can be calculated as:

$$\mu(x) = E[z_i(x)] = E[b_i] + E[\sum_{j=1}^{d} W^1_{ij} x^1_j(x)] = E[b_i] + \sum_{j=1}^{d} E[W^1_{ij}] E[x^1_j(x)] = 0$$

$$K(x, x') = E[z_i(x) z_i(x')] = \sigma^2_b + \sum_{j=1}^{d} (\sigma^2_{W_1}/d) E[x^1_j(x) x^1_j(x')] = \sigma^2_b + \sigma^2_{W_1} C(x, x')$$

Here $C(x, x') = E[x^1_j(x) x^1_j(x')]$. According to Neal's paper[3], the $C(x, x')$ term is obtained by numerical integration over the input data.

The above arguments can be extended to the case of deep neural networks with two or more hidden layers by induction. Let z^{l-1}_j represent the j^{th} component of the output from layer $(l - 1)$ and we suppose that it is a Gaussian process for every j. we can then represent the output of layer l as:

$$z^l_i(x) = b^l_i + \sum_{j=1}^{d_l} W^l_{ij} x^l_j(x), \quad x^l_j(x) = \phi(z^{l-1}_j(x))$$

Similar to the previous statement, $z^l_i(x)$ is the sum of i.i.d terms and thus can be considered as a linear combination of Gaussian processes. Therefore when the width d_l grows to infinity, $z^l_i(x)$ converges to a Gaussian process: $z_i^l \sim GP(0, K_l)$. For the randomness of the weights and bias the mean is obviously zero. The covariance function is given by:

$$K_l(x, x') = E[z_i^l(x)z_i^l(x')] = \sigma_b^2 + \sigma_w^2 E[\phi(z_j^{l-1}(x))\phi(z_j^{l-1}(x'))]$$

We can introduce a recursive form of notation to indicate the covariance function:

$$K_l(x, x') = \sigma_b^2 + \sigma_w^2 F_\phi(K_{l-1}(x, x), K_{l-1}(x, x'), K_{l-1}(x', x'))$$

The recursive relation is determined by the function F_ϕ which depends on the the non-linearity of ϕ. F_ϕ takes input of $K_{l-1}(x, x)$, $K_{l-1}(x, x')$ and $K_{l-1}(x', x')$ since they are the only possible distinct entries for the covariance function $K_l(x, x')$ in the sense of recursion. By performing the recursive calculations, we can develop a sequence of Gaussian processes, one for each layer. In light of the previous construction, we present an interesting argument. When we consider a neural network of one hidden layer, we can use a Gaussian process to approximate it as the width goes to infinity. As we increase the depth, we can approximate the output of each hidden layer with a Gaussian process, while each Gaussian process obtained is recursively related. Therefore we can approximate a sequence of recursive functions with deep neural networks. Yet what happens if we consider this sequence of recursive functions as a deep neural network? We can thus approximate a deep neural network with a deep neural network. If we extend the depth to infinity, we can generate an infinite sequence of approximated Gaussian processes. We suppose that this construction can be used to approximate more complex continuous functions, or functions that contains a higher dimension of continuity.

2.3 Implementation for Bayesian Learning

Bayesian inference is a fundamental and useful method of updating the probabilistic distribution of a hypothesis. Since we can approximate a Gaussian process with neural networks, we can apply this technique by using a Gaussian prior over functions. Given the dataset $\mathcal{D}$ (from Section 1.1) which consists of input-target output pairs (x, y), we aim to predict the output of a new input x^* by using a distribution over functions $z(x)$. This distribution gives the values $(z_1, ..., z_n)$ to the inputs $(x_1, ..., x_n)$. From here, we can make predictions for the output by calculating the conditional probability:

$$P(z^*|\mathcal{D}, x^*) = \int P(z^*|z, x, x^*)P(z|\mathcal{D})dz = \frac{1}{P(y)}\int P(z^*, z|x^*, x)P(y|z)dz$$

where $y = (y_1, ..., y_n)^T$ are the outputs of the training set. The term $P(y|z)$ can be considered as observation noise, which can be described by a gaussian model with variance σ_ϵ^2 and mean z for a small $\epsilon > 0$.

As we apply the results from Section 2.2, we can consider that we obtained a Gaussian process from the neural network. Hence, the values $(z_1, ..., z_n, z^*)$ can be seen as $n + 1$ functions driven from a Gaussian process and $z^*, z|x^*, x \sim N(0, K)$ is a multivariate Gaussian whose covariance matrix has the form:

$$K = \begin{bmatrix} K_{\mathcal{D},\mathcal{D}} & K_{x^*,\mathcal{D}}^T \\ K_{x^*,\mathcal{D}} & K_{x^*,x^*} \end{bmatrix}$$

Each component in this covariance matrix corresponds to the division between training sets and the test point. To specify, $K_{\mathcal{D},\mathcal{D}}$ is a $n \times n$ matrix with the (x_i, x_j) entry as $K(x_i, x_j)$ where x_i, x_j are both inputs from the training set. Similarly $K_{x^*,\mathcal{D}}$ is a $1 \times n$ matrix with the i^{th} entry as $K(x^*, x^i)$. As is standard in Gaussian processes, the integral in the equation for conditional probability $P(z^*|D, x^*)$ can be evaluated, resulting in a normal distribution of $z^*|\mathcal{D}, x^* \sim N(\mu^*, K^*)$ with

$$\mu^* = K_{x^*,\mathcal{D}}(K_{\mathcal{D},\mathcal{D}} + \sigma_\epsilon^2 I_n)^{-1}y$$

$$K^* = K_{x^*,x^*} - K_{x^*,\mathcal{D}}(K_{\mathcal{D},\mathcal{D}} + \sigma_\epsilon^2 I_n)^{-1}K_{x^*,\mathcal{D}}^{-1}$$

where I_n is the $n \times n$ identity matrix. Therefore it can be seen that the predicted distribution for $z^*|\mathcal{D}, x^*$ is determined from direct matrix calculation, yet the premise to allow this process of calculation is to approximate a Gaussian process by deep continuous neural networks, and the covariance matrix is determined by the kernel of the approximated Gaussian process, which further depends on the depth, width, nonlinearity, and weights of our neural network.

3 Theoretical Extensions and Applications

While the idea of neural networks operating on the continuum is relatively unexplored, there are several recent works detailing neural networks operating in infinity and, with varying degrees of success, applying these networks to achieve meaningful results. In this section we explore a few situations where neural networks can be extended to the infinite case.

3.1 Convolutional Networks

In the case of convolutional neural networks, we can further the idea of replacing finite components of neural networks by their continuous analogues by considering the continuous convolution as a unit in such a network. In particular, for continuous $f : \mathbf{R}^D \to \mathbf{R}$, $g : \mathbf{R}^D \to \mathbf{R}$, and $x \in \mathbf{R}^D$ we can give the continuous convolution

$$(f * g)(\mathbf{x}) = \int_{-\infty}^{\infty} f(\mathbf{y})g(\mathbf{x}-\mathbf{y})\, d\mathbf{y}$$

Such an architecture is explored by Wang et al. [14], where a the authors specify a layer containing such units that takes as input a set of points in space and outputs another set of points. In the continuous convolution, the kernel function g can be restrained to local space (as is done in the discrete counterparts) by multiplying by a window function that is only nonzero for points within a certain area (such as the K-nearest neighbors or a fixed radius r). This new representation presents a couple of new implementation challenges, such as determining the kernel function and evaluating the convolution function in practice. For the latter, the authors use a numerical method reminiscent of Monte-Carlo integration, using in place of the continuum a set of N input of points $\mathbf{y}_i$ sampled from the domain and approximating the convolution function thus:

$$(f * g)(\mathbf{x}) = \int_{-\infty}^{\infty} f(\mathbf{y})g(\mathbf{x}-\mathbf{y})\, d\mathbf{y} \approx \sum_{i}^{N} \frac{1}{N} f(\mathbf{y}_i)g(\mathbf{x}-\mathbf{y}_i)$$

For the former, the authors use a multilayer perceptron in order to approximate the kernel function and justify this choice by citing Hornik's universal approximation theorem for neural networks for their expressiveness and consider other choices (such as polynomials) unfit due to limitations on expressiveness or potential numerical instability. Notwithstanding the complexity inherent to using neural network approximations for functions themselves units in neural networks, this method is still arguable for parametric estimation of a function with infinite support. Care should also be given to specify the properties of the output of a continuous convolution layer. For this, the authors prespecify an output domain $\mathcal{O} = \{\mathbf{x}_i\}$ so that given the input feature vector $\mathcal{F} = \{\mathbf{f}_{in}, j \in \mathbf{R}^F\}$ with dimensionality F and N quantity of points in the support domain $\mathcal{S} = \{\mathbf{y}_j\}$, we can use the kernel function $g_{d,k}$ specified for each dimension of the input feature $d < F$ and output feature $k < O$ thus:

$$h_{k,i} = \sum_{d}^{F} \sum_{j}^{N} g_{d,k}(\mathbf{y}_i - \mathbf{y}_j) f_{d,j}$$

This layer definition allows for deep architectures with several of these layers feeding into each other and where the final output can be interpreted as a point cloud in the continuous output space. These neural networks have potential for use in cases where data are provided not in pixel format but as points, usually in 2D or 3D space. For example, the aforementioned authors of [14] demonstrate high performance in semantic labelling of 3D point clouds and LIDAR motion estimation, both when alone and when combined with existing architectures.

3.2 Neural Tangent Kernels

It is also possible to show that the training process of a neural network can also be associated with a kernel. In particular, viewing a neural network architecture as a function from the space of parameters to the space of functions, one can view training as the process of optimizing the output function in the space of functions according to some functional cost. In work presented by Jacot et al. ([5]), the authors give the formulation for a new type of kernel and show that, if the width of each layer of a neural network approaches infinity, this kernel becomes deterministic at initialization and does not change during training. The authors arrive at the existence of the kernels thus:

Consider a neural network as a function from parameter space to function space $F : \mathbf{R}^P \longrightarrow \mathcal{F}$, taking parameters $\theta \in \mathbf{R}^P$ and outputting another function $f \in \mathcal{F} : \mathbf{R}^N \longrightarrow \mathbf{R}^O$, where N and O are the number of input and output nodes in the neural network. Then, tracking infinitesimal changes of the parameters, we construct a kernel for which the process of gradient descent is equivalent to following the (negative) kernel gradient. This kernel, formally given as

$$\Theta(\boldsymbol{\theta}) = \sum_{p=1}^{P} \frac{\partial}{\partial \theta_p} F(\boldsymbol{\theta}) \otimes \frac{\partial}{\partial \theta_p} F(\boldsymbol{\theta})$$

is the Neural Tangent Kernel. Given inputs $\mathbf{x}, \mathbf{x}'$, this kernel has output:

$$\Theta(\mathbf{x}, \mathbf{x}') = \mathrm{E}_{\boldsymbol{\theta} \sim \mathcal{W}} \left\langle \frac{\partial f(\boldsymbol{\theta}, \mathbf{x})}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}, \mathbf{x}')}{\partial \boldsymbol{\theta}} \right\rangle$$

We can arrive at a formulation for the NTK starting with formulations for the neural network itself. In particular, for a layer $\mathbf{h}$ we shall specify activation $\mathbf{g}^{(h)}$ and pre-activation $\mathbf{f}^{(h)}$. Starting with $\mathbf{g}^{(0)}(\mathbf{x}) = \mathbf{x}$, we inductively define each layer $h > 0$ with d_h neurons. Using the weight matrix $\mathbf{W}^{(h)} \in \mathbf{R}^{d_h \times d_{h-1}}$ for layer h, we give the components $\mathbf{f}^{(h)} : \mathbf{R}^{d_{h-1}} \longrightarrow \mathbf{R}^{d_h}$, $\mathbf{g}^{(h)} : \mathbf{R}^{d_{h-1}} \longrightarrow \mathbf{R}^{d_h}$ for the activation of layer h from layer $h-1$ thus:

$$\mathbf{f}^{(h)}(\mathbf{x}) = \mathbf{W}^{(h)} \mathbf{g}^{(h-1)}(\mathbf{x}), \mathbf{g}^{(h)}(\mathbf{x}) = \sqrt{\frac{c_\sigma}{d_h}} \sigma(\mathbf{f}^{(h)}(\mathbf{x}))$$

where $\sigma : \mathbf{R} \longrightarrow \mathbf{R}$ is an activation function, and $c_\sigma = \left(\mathrm{E}_{z \sim \mathcal{N}(0,1)}\left(\sigma(z)^2\right)\right)^{-1}$. In essence, this kernel is induced by a linear model on the feature mapping

$$\phi(x) = \nabla f(x, \boldsymbol{\theta})$$

where the input x is mapped to the gradient of its output over parameter space. One clear issue germane to the discussion of the fact that the parameter mapping is nonlinear, causing that the kernel to vary during training. As a result, it is important to know whether such a kernel can exist at all and the conditions necessary for kernel to become stable. In particular, in a given neural network architecture, as the width of each layer approaches infinity, the Neural Tangent Kernel associated to a randomly initialized network can be shown to become deterministic and furthermore does not change during training. Then, in the infinite width limit, the pre-activations $\mathbf{f}^{(h)}(\mathbf{x})$ of a layer h approaches a Gaussian process at all coordinates with covariance $\Sigma^{(h-1)} : \mathbf{R}^{d_0} \times \mathbf{R}^{d_0} \longrightarrow \mathbf{R}$ which is also be defined inductively:

$$\Sigma^{(0)}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\intercal \mathbf{x}$$

$$\mathbf{\Lambda}^{(h)} = \begin{pmatrix} \Sigma^{(h-1)}(\mathbf{x}, \mathbf{x}) & \Sigma^{(h-1)}(\mathbf{x}, \mathbf{x}') \\ \Sigma^{(h-1)}(\mathbf{x}', \mathbf{x}) & \Sigma^{(h-1)}(\mathbf{x}', \mathbf{x}') \end{pmatrix}$$

$$\Sigma^{(h)}(\mathbf{x}, \mathbf{x}') = c_\sigma \mathbf{E}_{(u,v) \sim \mathcal{N}(\mathbf{0}, \mathbf{\Lambda}^{(h)}(\mathbf{x}, \mathbf{x}'))}[\sigma(u)\sigma(v)]$$

Finally, we define derivative covariance:

$$\Sigma'^{(h)}(\mathbf{x}, \mathbf{x}') = c_\sigma \mathbf{E}_{(u,v) \sim \mathcal{N}(\mathbf{0}, \mathbf{\Lambda}^{(h)}(\mathbf{x}, \mathbf{x}'))}[\sigma'(u)\sigma'(v)]$$

Then, the NTK expression for a L-layer fully connected neural network as a neural network of infinite width is thus:

$$\Theta(\mathbf{x}, \mathbf{x}') = \sum_{h=1}^{L+1} \left(\Sigma^{(h-1)}(\mathbf{x}, \mathbf{x}') \cdot \prod_{h'=h}^{L+1} \Sigma'^{(h)}(\mathbf{x}, \mathbf{x}') \right)$$

Arora et al. also give a concrete theorem on the convergence of the NTK, improving on the initial paper's asymptotic proof of convergence:

Theorem 3.1. (Arora et. al.) Let $\epsilon > 0$ and $\delta \in (0, 1)$. Then, for a neural network with L layers and parameters $\boldsymbol{\theta}$, letting $\sigma(z) = \max(0, z)$ and $\min_{h \in [L]} d_h \geq \Omega(\frac{L^6}{\epsilon^4} \log(L/\delta))$, for any inputs $\mathbf{x}, \mathbf{x}' \in \mathbf{R}^{d_0}$ with $\|\mathbf{x}\| \leq 1, \|\mathbf{x}'\| \leq 1$, with probability at least $1 - \delta$:

$$\left| \left\langle \frac{\partial}{\partial \boldsymbol{\theta}} f(\boldsymbol{\theta}, \mathbf{x}), \frac{\partial}{\partial \boldsymbol{\theta}} f(\boldsymbol{\theta}, \mathbf{x}) \right\rangle - \Theta(\mathbf{x} - \mathbf{x}') \right| \leq (L+1)\epsilon$$

Proof. The extremely abridged idea for the proof of this theorem entails finding a lower bound for the given probability by finding lower bounds on the probabilities of certain constituent events and fiddling properly with epsilons such that one can combine these events through intersection or implication while still guaranteeing a certain lower bound.

First, we focus on a single layer h, so that the full parameter $\boldsymbol{\theta}$ is replaced with a single weight matrix $\mathbf{W}^{(h)}$ so that the desired inequality is obtained by summing $L+1$ simpler inequalities

$$\left|\left\langle \frac{\partial}{\partial \mathbf{W}^{(h)}} f(\boldsymbol{\theta}, \mathbf{x}), \frac{\partial}{\partial \mathbf{W}^{(h)}} f(\boldsymbol{\theta}, \mathbf{x}') \right\rangle - \Sigma^{(h-1)}(\mathbf{x}, \mathbf{x}') \cdot \prod_{h'=h}^{L+1} \Sigma'(\mathbf{x}, \mathbf{x}') \right| \leq \epsilon$$

We then split this equation further, separating out the behaviors of the constituent parts multiplied in the NTK expression, namely $\Sigma^{(h-1)}$ and $\prod_{h'=h}^{L+1} \Sigma'^{(h')}$.

The corresponding constituents of the left-hand expression requires introducing a new vector function $\mathbf{b}^{(h)} : \mathbf{R}^{d_0} \longrightarrow \mathbf{R}^{d_h}$ where

$$\left\langle \frac{\partial}{\partial \mathbf{W}^{(h)}} f(\boldsymbol{\theta}, \mathbf{x}), \frac{\partial}{\partial \mathbf{W}^{(h)}} f(\boldsymbol{\theta}, \mathbf{x}') \right\rangle = \mathbf{b}^{(h)}(\mathbf{x}) \cdot \left(\mathbf{g}^{(h-1)}\right)^{\intercal}$$

noting that $\mathbf{b}^{(h)}$ can be defined inductively from $\mathbf{b}^{L+1} \equiv 1$. Here, our goal is to establish upper bounds on the following expressions for each of $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \in \{(\mathbf{x}, \mathbf{x}), (\mathbf{x}, \mathbf{x}')(\mathbf{x}', \mathbf{x}')\}$,

$$\left|\mathbf{g}^{(h)}(\mathbf{x}^{(2)})^{\intercal} \mathbf{g}^{(h)}(\mathbf{x}^{(1)}) - \Sigma^{(h)}(\mathbf{x}^{(2)}, \mathbf{x}^{(1)})\right|$$

and

$$\left|\left\langle \mathbf{b}^{(h)}(\mathbf{x}^{(1)}), \mathbf{b}^{(h)}(\mathbf{x}^{(2)}) \right\rangle - \prod_{h'=h}^{L+1} \Sigma'^{(h')}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})\right|$$

after which we use use the upper bounds on $\|\mathbf{x}\|$ and $\|\mathbf{x}'\|$ ($\Sigma^{(h)}(\mathbf{x}, \mathbf{x}) = \|\mathbf{x}\|^2$) and follow the chain of inequalities back to the original desired inequality. ■

This indicates that the Neural Tangent Kernel depends only on the method of initialization, layer architecture, and nonlinearity unit used in the network. As a result, such kernels could potentially be used to study neural architecture as its width approaches infinity.

3.3 Convolutional Neural Tangent Kernels

Applying the same ideas to Convolutional Neural Networks gives an analogous kernel. ([1]) The authors also give a more efficient method of implementation for these kernels, meaning that it was possible to measure the performance of CNTKs. In particular, the authors are able to achieve 77% classification accuracy on the CIFAR-10 dataset using CNTKs. Furthermore, testing multiple CNTKs over different architectures, the authors found a strong correlation between the neural network's performance and the corresponding CNTK performance, potentially indicating use in neural network architecture. More interestingly, however, this performance is still 5-6% lower than that of the finite networks from which the kernels were derived, a gap which is not currently understood according to current theory on over-paramaterization. It remains to be seen whether this gap is simply a result of the inherent performance gap between the tentative first implementation of a new algorithm and that of a relatively known one or whether this gap has a theoretical basis. It would also be of at least token interest to observe the behavior of the implemented CNTK on the training set, since infinite width neural networks asymptotically approach zero training error.

3.4 Research gaps

Curiously, despite the very recent growth of research in fields regarding regular and convolutional neural networks in the infinite width case, there is little exploration on wide recurrent neural networks or variants such as generative networks. This might be due to the computational difficulties associated with these particular architectures or the additional theoretical challenges introduced by inherent features such as internal states. We should note here that the topic of recurrent neural networks over the continuous time domain has been studied before, but works in this area are generally concerned with the stability of such networks, which, while of valuable theoretical interest, are outside the scope of our interests. Furthermore, while Neural Tangent Kernels are interesting and potentially highly useful tools for analysis of over-paramaterized neural networks, currently they are being re-derived to fit each type of network architecture, which might result in a significant time lag between the discovery of novel architectures and any useful analysis using such kernels.

4 Function machines

One might now want to consider the situation where the input and/or output of a neural network is infinite dimensional, i.e. the neural network takes in a continuum (and potentially outputs another continuum). A case where this problem setup applies is the Fourier transform: given a time continuum of signal, return a continuum of frequencies that represent the power spectral density of the input signal. Since a continuum C can be represented by some function from an indexing set (say, a compact $S \subset \mathbf{R}$) onto C, we will consider the function space, and study operators and functionals. To answer the question of to what extent can a neural network approximate these mathematical objects, we must consider a generalization of the neural network approximation problem setup.

4.1 A functional perspective

Answering the question of when a neural network can approximate certain functions is analogous to studying when the class of functions expressed by neural networks is dense in the family of functions we're interested in. We would like to remark to the reader that the bias term will be given consideration from here onwards. However, it will be added accordingly to computational steps instead of a 1 appended to the input vector.

Consider $\mathcal{F} = \{f : \mathcal{X} \longrightarrow \mathcal{Y}\}$ a family of morphisms, where $\mathcal{X}$ and $\mathcal{Y}$ are Banach spaces. Let $\mathcal{G}_L = \{T_L \circ g \circ T_{L-1} \circ ... \circ g \circ T_1 : \mathcal{X} \longrightarrow \mathcal{Y}\} \subset \mathcal{F}$ be the family of functions that can be represented by a neural network of L layers with activation g. Now, given a topology τ on $\mathcal{F}$ and $S \subset \mathcal{X}$ compact, we want to know when $\mathrm{cl}(\mathcal{G}_L|_S) = \mathcal{F}|_S$, where $\mathrm{cl}(\cdot)$ denotes the closure. For example, the universal approximation theorem [4] states that $\mathrm{cl}(\mathcal{G}_2|_S) = \mathcal{F}|_S$ when $\mathcal{X}, \mathcal{Y}$ are finite dimensional and $\mathcal{F}$ is the class of continuous functions from $\mathcal{X}$ to $\mathcal{Y}$. The following are notations that will be used in this section.

Notation 4.1. We denote the family of continuous functions from $\mathcal{X}$ to $\mathcal{Y}$ by $C(\mathcal{X}, \mathcal{Y})$. Furthermore, $C(\mathcal{X}) := C(\mathcal{X}, \mathbf{R})$. We denote $\|\cdot\|_{\mathcal{X}}$ the norm associated to $\mathcal{X}$, inducing its topology. We will assume $C(\cdot, \cdot)$ is with the uniform topology induced by the uniform norm, and $\mathcal{L}^p(\cdot, \cdot)$ is with its p-norm.

Abstracting infinite dimensional layers is necessary to formulate exactly what class of neural networks we are considering. The following excellent definitions are due to [2].

Definition 4.2. Let $K \subset \mathbf{R}^d, K' \subset \mathbf{R}^{d'}$ both compact, T is some affine map.

1. (Operator layer) When $T : \mathcal{L}^1(K, \mu) \longrightarrow \mathcal{L}^1(K', \mu)$, $T = T^{\mathfrak{o}}$ is said to be an **operator layer** if there is some continuous family of measures $W_t \ll \mu$ over $t \in K'$ and a function $b \in \mathcal{L}^1(K, \mu)$ such that:

$$T^{\mathfrak{o}} : f \mapsto \left(t \mapsto \int_K f dW_t + b(t)\right)$$

presented in [10] in less generality.

2. (Functional layer) When $T : \mathcal{L}^1(K, \mu) \longrightarrow \mathbf{R}^{d'}$, $T = T^{\mathfrak{f}}$ is said to be a **functional layer** if there is some measure $W \ll \mu$ and a vector $b \in \mathbf{R}^d$ such that:

$$T^{\mathfrak{f}} : f \mapsto \int f dW + b$$

first introduced by [13].

3. (Basis layer) When $T : \mathbf{R}^d \longrightarrow \mathcal{L}^1(K', \mu)$, $T = T^{\mathfrak{b}}$ is said to be a **basis layer** if there is some function $w : K' \longrightarrow \mathbf{R}^d$ and $b \in \mathcal{L}^1(K', \mu)$ such that:

$$T^{\mathfrak{b}} : y \mapsto \left(t \mapsto \sum_{i=1}^{d} y_i [w(t)]_i + b(t) \right)$$

Thus, the statement that continuous neural networks universally approximate continuous functions over a compact domain S is formulated as $\mathrm{cl}(\mathcal{G}_2|_S = \{T^{\mathfrak{f}} \circ g \circ T^{\mathfrak{b}} : \mathbf{R}^d \longrightarrow \mathbf{R}^{d'}\}) = C(\mathbf{R}^d, \mathbf{R}^{d'})|_S$.

4.2 Functional approximation

We give a formulation for functional approximation using continuous neural networks and show that they are a class of universal approximators by heavily relying on results from [10], [11], and [2]. Let μ be a $\sigma-$finite measure on (X, Σ) measurable. Consider $\mathcal{X} = \mathcal{L}^p(X, \mu)$. We define a hidden unit's output post-activation to be:

$$N_i(f) = g\left(b + \int f w_i d\mu\right)$$

where b is a real-valued bias, $w_i \in \mathcal{L}^q(X, \mu)$ is the weight function for unit N_i for i indexed by E compact, and where p and q are conjugate exponents. Let φ be the canonical isometric isomorphism $\varphi : \mathcal{L}^q(X, \mu) \longrightarrow \mathcal{L}^p(X, \mu)^*$ such that:

$$\varphi[w_i](f) = \mathcal{W}_i(f) = \int w_i f d\mu$$

where $\mathcal{W}_i : \mathcal{L}^p(X, \mu) \longrightarrow \mathbf{R}$ is a bounded linear functional.

Indeed, defining $W_i(E) = \mathcal{W}_i(\chi_E) = \int_E w_i d\mu$, and realizing that $W_i \ll \mu$, w_i is exactly the Radon-Nikodym derivative $\frac{dW_i}{d\mu}$, we rewrite N_i as:

$$N_i(f) = g\left(b + \int f dW_i\right)$$

The formulation of continuous networks defined in 1.1 motivates the following definition:

Definition 4.3. (Functional contNN) Let (X, μ) be a measure space, and $\mathcal{L}^p(X, \mu)$ the space of measurable functions such that the p^{th} power of their absolute value is Lebesgue integrable, $K \subset \mathcal{L}^p(X, \mu)$ compact. A functional continuous neural network, or functional contNN, is a functional $F : C(K) \longrightarrow \mathbf{R}$ that computes:

$$F(f) = b' + \int_E a(t) g\left(b + \int_K f dW_t\right) dt$$

where b', b are real-valued biases, g an activation function, and $E \subset \mathbf{R}$ compact. It is not hard to see that the above typechecks.

Remark 4.4. It is true that we can make the indexing set E a subset of $\mathbf{R}^d$ for any $d \in \mathbf{N}$, but not necessary.

Remark 4.5. When a is measurable, we can generalize the formulation of a functional contNN. Let $(W_v^1 \ll \mu)_{v \in E}$ be a family of Lebesgue absolutely continuous measures, $W^2 \ll \mu$ Lebesgue absolutely continuous, such that a is the Radon-Nikodym derivative $\frac{dW^2}{d\mu}$. We thus have:

$$F(f) = b' + \int_E g\left(b + \int_K f dW_v^1\right) dW^2$$

We encourage the reader to check that this formulation is equivalent to 4.3. This generalization makes it visible that the family of functional contNNs are described by an operator layer composed with a functional layer: $\{T^{\mathfrak{f}} \circ g \circ T^{\mathfrak{o}} : C(K) \longrightarrow \mathbf{R}^{d'}\}$. It is important to note here that although our generalized formula gives an output in $\mathbf{R}$, it readily extends to giving outputs in $\mathbf{R}^{d'}$ by indexing each dimension with $F_i(f)$.

Theorem 4.6. (Viet's theorem) cl$(\mathcal{G}_2|_S = \{T^{\mathfrak{f}} \circ g \circ T^{\mathfrak{o}} : C(K) \to \mathbf{R}^{d'}\}) = C(C(K), \mathbf{R}^{d'})|_S$.

Proof. The proof employs the original formulation in 4.3. The Riemann integral inspires us to proceed in a similar fashion as 1.4. Recall that we can parametrize and normalize a continuous neural network similar to 1.2. We thus have:

$$\begin{aligned}\hat{F}(f) &= b' + \alpha \int_0^1 g\left(b + \int_K f dW_z\right) dz \\ &= b' + \alpha \int_0^1 g\left(b + \int_K f w_z d\mu\right) dz\end{aligned}$$

where $w_z = \frac{dW_z}{d\mu}$. Recall that by the universal approximation theorem for nonlinear functionals in [13], given $\epsilon > 0$, if F is a continuous functional over S, there exists functions $w_1, \ldots, w_{d'}$, output weights $a_i, i = 1, \ldots, d'$, and biases b, b' such that:

$$\sup_{f \in S} \left| F(f) - b' - \sum_{i=1}^{d'} a_i g\left(b + \int_K w_i f d\mu\right) \right| < \epsilon$$

By optimally replacing w_z with $-w_z$ and b with $-b$, we can restrict all a_i's to be positive. Let $\alpha = \sum_i a_i$ and defining $w'_z = w_i$ if $\frac{\sum_{k=1}^{i=1} a_k}{\alpha} \le z \le \frac{\sum_{k=1}^{i} a_k}{\alpha}$, we have:

$$\sup_{f \in S} \left| F(f) - b' - \alpha \int_0^1 g\left(b + \int_K f w_z d\mu\right) dz \right| < \epsilon$$

$$\sup_{f \in S} \left| F(f) - \hat{F}(f) \right| < \epsilon$$

Therefore, functional contNNs are universal approximators. ∎

In fact, they are equivalent to functional networks introduced by [10], which were already proven to be universal approximators.

4.3 Operator approximation

Authors of [2] claim that when $\mathcal{X}, \mathcal{Y}$ are topological vector spaces and $f : \mathcal{X} \to \mathcal{Y}$, continuity of f and compactness of its domain is enough to be uniformly approximated by continuous neural networks. In particular, one can take $\mathcal{X}$ and $\mathcal{Y}$ to be infinite dimensional function spaces. The following theorem gives the universal approximation property for continuous operators on a compact domain:

Theorem 4.7. (Guss et. al.) Let $F : C(K) \to C(K')$ be a continuous operator. For every $\epsilon > 0$ and any $d'' > 0$, there exists a compact indexing set $K'' \subset \mathbf{R}^{d''}$, two continuous families of Lebesgue absolutely continuous measures $(W_v^1 \ll \mu)_{v \in K''}, (W_y^2 \ll \mu)_{y \in K'}$, and biases $b \in \mathcal{L}^1(K, \mu), b' \in \mathcal{L}^1(K'', \mu)$ such that:

$$\sup_{f \in S, y \in K'} \left\| \int_{K''} g\left(b(v) + \int_K f dW_v^1\right) dW_y^2 + b'(y) - F[f](y) \right\| < \epsilon$$

which essentially says that under the supremum norm, the class of compositions of operator layers is dense in the collection of continuous operators over a compact domain S. Hence, cl$(\mathcal{G}_2|_S = \{T^{\mathfrak{o}} \circ g \circ T^{\mathfrak{o}} : C(K) \to C(K')\}) = C(C(K), C(K'))|_S$.

In fact, this approximation is still true even if the hidden layer was not infinite dimensional, i.e. cl$(\mathcal{G}_2|_S = \{T^{\mathfrak{b}} \circ g \circ T^{\mathfrak{f}} : C(K) \to C(K')\}) = C(C(K), C(K'))|_S$. A categoric proof is given to the above two theorems in [2], with the crucial steps being:

1. Studying the decomposition of maps of the form $F : \mathcal{X} \to \mathcal{Y}$ into finite dimensional maps $\tilde{F} : V \subset \mathbf{R}^N \to \mathbf{R}^M$ through *sample factorizations*, objects that behave similarly to functors.

2. Constructing a neural network N which approximates $\tilde{F}$.
3. Showing that different compositions of sample factorizations with N layers can be approximated using layers in 4.2.

In theory, it should be possible for neural networks to approximate operators such as the Fourier transform, the Laplace transform, and even derivation in differential algebra. Given a way to compute the outputs of these function machines, one can, for instance, instead of performing Fast Fourier Transform on a digital signal, one could consider the approximation of the Fourier transform on hypothetical continuous time data, which arises frequently in quantum mechanics, and parametrize the function machine instead. This enables the formulation of a wide class of operations as closed form neural networks, opening new doors to alternative approaches through universal approximations to multiple scientific domains, endowing the theoretical study of operators and functionals with a new tool.

5 Conclusion

This paper explores the mathematical formulations underlying the novel and relatively unexplored form of neural networks with continuous hidden layers. Showing their universality property, we demonstrated that affine parametrizations could in theory yield faster convergence guarantees than their constant counterparts. With i.i.d. weights, we show the correspondence between neural networks and Gaussian processes in the infinite width limit. In addition, we explore extensions of the framework of infinite neural networks such as continuous convolutions and the Neural Tangent Kernel. These constructions have the potential to be useful tools for exploring new neural network architectures, all the while providing a method for the computation of infinite networks. Finally, we consider neural networks as function transformers where inputs and outputs may be infinite dimensional, and demonstrate that such constructs maintain their universality in approximation.

Although significant progress has been made in understanding this new scheme for neural networks, this area remains largely unexplored. Currently, we identify further study of affine neural networks and continuous convolutions, uses and computation of Neural Tangent Kernels, and finding computationally feasible implementations of neural networks as functional machines. On top of that, one can explore PDEs and functional analysis using an approximation scheme with functional networks from the study of their closed forms. All in all, a keen eye should still be kept on computational practicality and applicability as theoretical results are sometimes infeasible in practice.

References

[1] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8139–8148. Curran Associates, Inc., 2019.

[2] William H Guss and Ruslan Salakhutdinov. On universal approximation by neural networks with uniform guarantees on approximation of infinite dimensional maps. *arXiv preprint arXiv:1910.01545*, 2019.

[3] Geoffrey E. Hinton and Radford M. Neal. Bayesian learning for neural networks. 1995.

[4] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[5] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8571–8580. Curran Associates, Inc., 2018.

[6] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[8] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[9] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[10] Fabrice Rossi and Brieuc Conan-Guez. Modélisation supervisée de données fontionnelles par perception multi-couches. In *Actes des neuvièmes journées de la SFC conférence invitée, Toulouse, France*, pages 93–100, 2002.

[11] Nicolas Le Roux and Yoshua Bengio. Continuous neural networks, 2007.

[12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[13] Maxwell B Stinchcombe. Neural network approximation of continuous functionals and continuous functions on compactifications. *Neural Networks*, 12(3):467–477, 1999.

[14] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[15] Christopher KI Williams. Computing with infinite networks. In *Advances in neural information processing systems*, pages 295–301, 1997.