Machines of Finite Depth: Towards a Formalization of Neural Networks

Pietro Vertechi*, **Mattia G. Bergomi***

pietro.vertechi@protonmail.com, mattiagbergomi@gmail.com

Abstract

We provide a unifying framework where artificial neural networks and their architectures can be formally described as particular cases of a general mathematical construction—*machines of finite depth*. Unlike neural networks, machines have a precise definition, from which several properties follow naturally. Machines of finite depth are modular (they can be combined), efficiently computable, and differentiable. The backward pass of a machine is again a machine and can be computed without overhead using the same procedure as the forward pass. We prove this statement theoretically and practically, via a unified implementation that generalizes several classical architectures—dense, convolutional, and recurrent neural networks with a rich shortcut structure—and their respective backpropagation rules.

1 Introduction

The notion of *artificial neural network* has become more and more ill-defined over time. Unlike the initial definitions (Rumelhart 1986), which could be easily formalized as directed graphs, modern neural networks can have the most diverse structures and do not obey a precise mathematical definition.

Defining a deep neural network is a practical question, which must be addressed by all deep learning software libraries. Broadly, two solutions have been proposed. The simplest approach defines a deep neural network as a stack of pre-built layers. The user can select among a large variety of pre-existing layers and utilize them as building blocks. This approach—*domain-specific language*—simplifies the end-user's mental load and leads to computationally efficient models. Unfortunately, the building-block-based approach can quickly become limiting and prevent users from exploring more innovative architectures (Barham and Isard 2019). At the opposite end of the spectrum, *differentiable programming* (Wang et al. 2018) posits that every code is a model, provided that it can be differentiated by an automatic differentiation engine (Frostig, Johnson, and Leary 2018; Innes et al. 2019; Moses and Churavy 2020; Paszke et al. 2017, 2021; Saeta and Shabalin 2021; van Merrienboer et al. 2018). This is certainly a promising direction, which has led to a number of advances, e.g., (Innes et al. 2019; Rackauckas et al. 2020; Zubov et al. 2021). Unfortunately, this approach has several drawbacks, some practical and some theoretical. On the practical side, it becomes difficult to optimize the runtime of the forward and backward pass of an automatically-differentiated, complex, unstructured code. From a theoretical perspective, the *space of models* becomes somewhat ill-defined, as it is now the space of *all differentiable codes*—not a structured mathematical space. A well-behaved smooth space of neural networks would be invaluable for automated differentiable *architecture search* (Liu, Simonyan, and Yang 2018). Furthermore, a well-defined notion of neural network would greatly simplify *sharing* models as precise mathematical quantities rather than differentiable code written in a particular framework.

Our ambition is to establish a unified framework for deep learning, in which deep feedforward and recurrent neural networks, with or without shortcut connections, are defined in terms of a *unique* layer, which we will refer to as a *parametric machine*. This approach allows for extremely simplified flows for designing neural architectures, where a small set of hyperparameters determines the whole architecture. By virtue of their precise mathematical definition, parametric machines will be language-agnostic and independent of automatic differentiation engines. Computational efficiency, in particular in terms of efficient *gradient* computation, is granted by the mathematical framework.

The theoretical framework of parametric machines (Vertechi and Bergomi 2020) unifies seemingly disparate architectures, designed for structured or unstructured data, with or without recurrent or shortcut connections. We provide theorems ensuring that 1. under the assumption of finite depth, the output of a machine can be computed efficiently; 2. complex architectures with shortcuts can be built by adding together machines of depth one, thus generalizing neural networks at any level of granularity (neuron, layer, or entire network); 3. backpropagating from output to input space is again a machine computation and has a computational cost comparable to the forward pass. In addition to the theoretical framework, we implement the input-output computations of parametric machines, as well as their derivatives, in the Julia programming language (Bezanson et al. 2017) (both on CPU and GPU).

*These authors contributed equally.

The implementation is open source and available at https://github.com/LimenResearch/ParametricMachinesDemos.jl. Each algorithm can be used as standalone or layer of a classical neural network architecture.

Section 2 introduces the abstract notion of *machine*, as well as its theoretical properties. In section 2.1, we define the *machine equation* and establish the link with deep neural networks and backpropagation, seen as machines on a global normed vector space. Section 2.2 discusses under what conditions the machine equation can be solved efficiently, whereas in section 2.3 we discuss how to combine machines. The theoretical framework is completed in section 2.4, where we introduce the notion of parametric machine and discuss explicitly how to differentiate its output with respect to the input and the parameters. Section 3 is devoted to practical applications: we discuss in detail an implementation of machines that extends classical dense, convolutional, and recurrent networks with a rich shortcut structure.

We refer the reader to supplementary sections S2 and S3 for complete proofs and detailed examples, respectively.

2 Machines

The key intuition to the proposed framework is that a neural network can be considered as an endofunction $f : X \to X$ on a space of global functions X (defined on all neurons on all layers). We will show that this viewpoint allows us to recover classical neural networks with arbitrarily complex shortcut connections. In particular, the forward pass of a neural network corresponds to computing the inverse of the mapping $\mathrm{id} - f$. We explore under what conditions on f the mapping $\mathrm{id} - f$ is invertible and provide practical strategies for computing the inverse and its derivative.

Our approach meshes well with the recent trend of *deep equilibrium models* (Bai, Kolter, and Koltun 2019; Winston and Kolter 2020; Gurumurthy et al. 2021). While the overarching formalism is similar, we choose a set of fixed-point problems based on an *algebraic* condition which generalizes classical neural architectures and does not compromise on computational efficiency.

2.1 Resolvent

In the classical deep learning framework, function composition appears to be the natural operation to combine consecutive layers. For instance, let us consider a multilayer perceptron denoted by $X_0 \xrightarrow{l_1} X_1 \xrightarrow{l_2} \dots X_{d-1} \xrightarrow{l_d} X_d$, where $l_i : X_{i-1} \to X_i$ is the standard fully-connected layer followed by a pointwise nonlinearity. This sequence of layers can be composed into a map $l_d l_{d-1} \dots l_2 l_1 \colon X_0 \to X_d$. However, this intuition breaks down in the presence of shortcut connections or more complex, non-sequential architectures. From a mathematical perspective, a natural alternative is to consider a *global* space $X = \bigoplus_{i=0}^{d} X_i$ and the global endofunction

$$f = \sum_{i=1}^{d} l_i \in C^k(X, X), \text{ where } k \in \mathbb{N} \cup \{\infty\}. \quad (1)$$

Explaining the relationship between f and the layer composition $l_d l_{d-1} \dots l_2 l_1$ is not straightforward. We assume that the output of the network is the entire space X, and not only the output of the last layer, X_d. Let the input function be the continuously differentiable inclusion map $g \in C^k(X_0, X)$. The map g embeds the input data into a space which encompasses input, hidden layers, and output. The network transforms the input map g into an output map $h \in C^k(X_0, X)$. Practically, h computes the activation values of all layers and stores not only the final result but also all intermediate activations.

The observation on which our framework is based is that f (as in eq. (1)) and g (the input function) are sufficient to determine h (the output function). Indeed, h is the only map in $C^k(X_0, X)$ respecting:

$$h = g + fh. \quad (2)$$

The existence of a unique solution to eq. (2) for any choice of input function g is the minimum requirement to ensure a well-defined input-output mapping for a *machine*, our generalization of a deep neural network.

Definition 1. *Let X be a normed vector space. Let $k \in \mathbb{N} \cup \{\infty\}$. An endofunction $f \in C^k(X, X)$ is a* k-differentiable machine *if, for all normed vector space X_0 and for all map $g \in C^k(X_0, X)$, there exists a unique map $h \in C^k(X_0, X)$ such that eq.* (2) *holds. We refer to X_0 and X as* input space *and* machine space*, respectively. We refer to eq.* (2) *as the* machine equation*.*

In the remainder we assume and shall use $k \geq 1$ to allow for backpropagation. See supplementary section S1 for mathematical preliminaries on normed vector spaces and Fréchet derivatives.

Definition 1 and eq. (2) describe the link between the input function g and the output function h. By a simple algebraic manipulation, we can see that eq. (2) is equivalent to $(\mathrm{id} - f)h = g$, where id is the identity function. In other words, f is a machine if and only the composition with $\mathrm{id} - f$ induces a bijection $C^k(X_0, X) \xrightarrow{\sim} C^k(X_0, X)$ for all normed vector space X_0. In other words, $\mathrm{id} - f$ is an isomorphism.

Proposition 1. *Let X be a normed vector space. $f \in C^k(X, X)$ is a machine if and only if* $\mathrm{id} - f$ *is an isomorphism. Whenever that is the case, the* resolvent *of f is the mapping*

$$R_f = (\mathrm{id} - f)^{-1}. \quad (3)$$

Then, $h = g + fh$ if and only if $h = R_f g$.

Thanks to proposition 1, it follows that the derivative of a machine and its dual are machines.

Proposition 2. *Let $f \in C^k(X, X)$ be a machine Let $x_0 \in X$. Then, the derivative $Df(x_0)$—a bounded linear endofunction in $B(X, X)$—and its dual $(Df(x_0))^* \in B(X^*, X^*)$ are machines, with resolvents $R_{Df(x_0)} = DR_f(x_0)$ and $R_{(Df(x_0))^*} = (DR_f(x_0))^*$, respectively.*

Another consequence of proposition 1 is that sequential deep neural networks $f = \sum_{i=1}^{d} l_i$ and nilpotent operators—bounded operators $f : X \to X$ such that $f^n =$

0 for some $n \in \mathbb{N}$—are machines (see supplementary section S3 for details). Sequential neural networks and nilpotent operators have some overlap: whenever all layers l_i are linear, $f = \sum_{i=1}^{d} l_i$ is a linear nilpotent operator. However, in general they are distinct: neural networks can be nonlinear and nilpotent operators can have more complex structures (shortcut connections). The goal of the next section is to discuss a common generalization—machines of *finite depth*.

2.2 Depth

We noted in section 2.1 that nilpotent continuous linear operators and sequential neural networks are machines, whose resolvents can be constructed explicitly. The same holds true for a more general class of endofunctions, namely endofunctions of *finite depth*. We follow the convention that, given a normed vector space X, a cofiltration is a sequence of quotients $X/V_i \to X/V_j$ for $i \geq j$, where each V_k is a closed subspace of X for every $k \in \mathbb{N}$.

Definition 2. *Let X be a normed vector space and $f \in C^k(X, X)$. Let $d \in \mathbb{N}$. A sequence of closed vector subspaces $X \supseteq V_0 \supseteq V_1 \supseteq \cdots \supseteq V_d = 0$ with projections $\pi_i \colon X \to X/V_i$ is a* depth cofiltration *of length d for f if the following conditions are verified.*

- $\pi_0 f = 0$ *or, equivalently,* $\operatorname{Im} f \subseteq V_0$.
- *For all i in $\{1, \dots, d\}$, there exists $\tilde{f}_i \in C^k(X/V_{i-1}, X/V_i)$ such that $\pi_i f = \tilde{f}_i \pi_{i-1}$.*

The depth *of f is the length of its shortest depth cofiltration, if any exists, and ∞ otherwise.*

Proposition 3. *Let $f \in C^k(X, X)$ be a machine. A sequence of closed vector subspaces $X \supseteq V_0 \supseteq V_1 \supseteq \cdots \supseteq V_d = 0$ is a depth cofiltration for f if and only if it is a depth cofiltration for $Df(x_0)$ for all $x_0 \in X$. Whenever that is the case, $X^* \supseteq (X/V_{d-1})^* \supseteq \cdots \supseteq (X/V_0)^* \supseteq (X/X)^* = 0$ is a depth cofiltration for $(Df(x_0))^*$ for all $x_0 \in X$.*

In simple cases, depth cofiltrations can be computed directly. For instance, if f is linear and continuous, then $X \supseteq \ker f^d \supseteq \cdots \supseteq \ker f \supseteq \ker f^0 = 0$ is a depth cofiltration for f if $f^{d+1} = 0$. Conversely, if a continuous linear operator f admits a depth cofiltration of length d, then necessarily $f^{d+1} = 0$. Hence, a continuous linear operator has finite depth if and only if it is nilpotent.

Sequential neural networks are another example of endofunction of finite depth. Let us consider a sequential architecture $X_0 \xrightarrow{l_1} X_1 \xrightarrow{l_2} \dots X_{d-1} \xrightarrow{l_d} X_d$ and $l = \sum_{i=1}^{n} l_i \in C^k(X, X)$, where $X = X_0 \oplus \cdots \oplus X_d$. Naturally $V_i = X_{i+1} + \cdots + X_d$ defines a depth cofiltration for l.

Definition 3. *Let $f \in C^k(X, X)$ and $g \in C^k(X_0, X)$. Let $d \in \mathbb{N}$ and let $X \supseteq V_0 \supseteq V_1 \supseteq \cdots \supseteq V_d = 0$ be a depth cofiltration. Its associated* depth sequence *is defined as follows:*

$$\tilde{h}_0 = \pi_0 g \quad \text{and} \quad \tilde{h}_i = \pi_i g + \tilde{f}_i \tilde{h}_{i-1} \text{ for } i \in \{1, \dots, d\},$$

where for all $i \in \{1, \dots, d\}$, $\tilde{h}_i \in C^k(X_0, X/V_i)$. A sequence $\{h_0, \dots, h_d\} \subseteq C^k(X_0, X)$ is a lifted depth sequence *if $\pi_i h_i = \tilde{h}_i$ for all $i \in \{0, \dots, d\}$.*

In other words, a depth sequence is a sequence of functions that approximate more and more accurately a solution to the machine equation, as we will prove in the following theorem. In general, the depth sequence can be lifted in different ways, which correspond to algorithms to solve the machine equation of different computational efficiency, as shown in fig. 1.

Proposition 4. *Let $\phi \in C^k(X_0, V_0)$. The sequence $h_0^\phi = g + \phi$ and $h_i^\phi = g + f h_{i-1}^\phi$ for $i \in \{1, \dots, d\}$ is a lifted depth sequence.*

Theorem 1. *Let us assume that $f \in C^k(X, X)$ admits a depth cofiltration of length d. Then, f is a machine. Furthermore, let us consider $g \in C^k(X_0, X)$, and let $\tilde{h}$ be its depth sequence. Then, $\tilde{h}_d = g + f\tilde{h}_d$.*

2.3 Composability

We develop the notion of machine *independence*, which will be crucial for *composability*, as it will allow us to create complex machines as a sum of simpler ones. In particular, we will show that deep neural networks can be decomposed as a sum of layers and are therefore machines of finite depth.

Definition 4. *Let X be a normed vector space. Let $f_1, f_2 \in C^k(X, X)$. We say that f_1* does not depend *on f_2 if, for all $x_1, x_2 \in X$, and for all $\lambda \in \mathbb{R}$, the following holds:*

$$f_1(x_1 + \lambda f_2(x_2)) = f_1(x_1). \tag{4}$$

Otherwise, we say that f_1 depends on f_2.

Definition 4 is quite useful to compute resolvents. For instance, f does not depend on itself if and only if it has depth at most 1, in which case it is a machine, and its resolvent can be computed via $R_f = \mathrm{id} + f$. Furthermore, by combining machines of finite depth with appropriate independence conditions, we again obtain machines of finite depth.

If f_1 is linear, then f_1 does not depend on f_2 if and only if $f_1 f_2 = 0$, but in general the two notions are distinct. For instance, functions $f_1(x) = x - 3$ and $f_2(x) = 3$ respect $f_1 f_2 = 0$, but f_1 depends on f_2 as $x - 3\lambda \neq x$ for $\lambda \neq 0$.

Definition 4 has an alternative formulation: f_1 does not depend on f_2 if and only if

$$(Df_1(x_1))f_2(x_2) = 0 \text{ for all } x_1, x_2 \in X. \tag{5}$$

Given $f_1, f_2 \in C^k(X, X)$, the sets $\{f \in C^k(X, X) \mid D(f_1(x_1))f(x_2) = 0 \text{ for all } x_1, x_2 \in X\}$ and $\{f \in C^k(X, X) \mid D(f(x_1))f_2(x_2) = 0 \text{ for all } x_1, x_2 \in X\}$ are vector spaces, as they are the intersection of kernels of linear operators. In other words, if f_1 does not depend on f_2 and $\hat{f}_2$, then it also does not depend on $\lambda f_2 + \hat{\lambda}\hat{f}_2$, and if f_1 and $\hat{f}_1$ do not depend on f_2, then neither does $\lambda f_1 + \hat{\lambda}\hat{f}_1$.

Theorem 2. *Let f_1, f_2 be machines, of depth d_1, d_2 respectively, such that f_1 does not depend on f_2. Then $f_1 + f_2$ is also a machine of depth $d \leq d_1 + d_2$ and $R_{f_1+f_2} = R_{f_2} R_{f_1}$. If furthermore f_2 does not depend on f_1, then $R_{f_1+f_2} = R_{f_1} + R_{f_2} - \mathrm{id}$ and $d \leq \max(d_1, d_2)$.*

A natural notion of *architecture with shortcuts* follows from theorem 2. Let $f_1, \dots, f_n$ be such that f_i does not

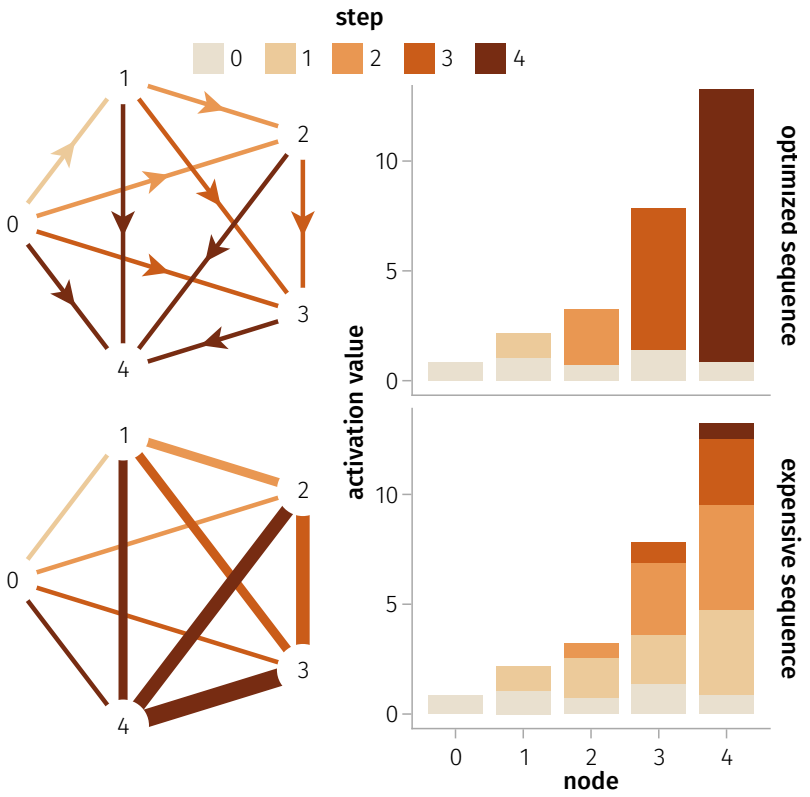

Figure 1: Different sequences to solve a linear machine with shortcuts on 5 nodes. The left column shows the connectivity graph. The right column describes the accumulated activation value on each node for a given input. For visual simplicity, we take positive connectivity matrix and input values, so that all updates are positive and can be represented a stacked bar plots. The top row represents an efficient strategy to solve the machine equation: at step i we update only the i-th node, as implied by color and orientation of the edges. In the bottom row, at the i-th step we evaluate $fh_i + g$. This is inefficient: connections need to be computed several times (line width of the edges of the bottom graph). The optimized sequence avoids this inefficiency by deferring the computation of each connection until the input value is fully determined.

depend on f_j if $i \leq j$. Then each f_i has depth at most 1, hence $f = \sum_{i=1}^{n} f_i$ has depth at most n by theorem 2. Indeed, $f_1 + \cdots + f_{i-1}$ does not depend on f_i, as can be verified for each addend individually thanks to eq. (5), hence by induction $f_1 + \cdots + f_i$ has depth at most i. Then, f is a machine of depth at most n, whose resolvent can be computed as $R_{f_n} \cdots R_{f_1} g = (\mathrm{id} + f_n) \cdots (\mathrm{id} + f_1) g$. In practice, this corresponds to the lifted depth sequence $\tilde{h}_0 = g$ and $\tilde{h}_{i+1} = \tilde{h}_i + f_i \tilde{h}_i$. This strategy can be applied to *acyclic* architectures with arbitrarily complex shortcuts. See supplementary fig. S1 for an intuition.

More generally, theorem 2 establishes a clear link between sums of independent machines and compositions of layers in classical neural networks. The independence condition determines the order in which machines should be concatenated, even in the presence of complex shortcut connections. Furthermore, if the initial building blocks all have finite depth, then so does the sum. Thus, we can compute the machine's resolvent efficiently. As a consequence, machines of finite depth are a *practically computable* generalization of deep neural networks and nilpotent operators.

2.4 Optimization

The ability to minimize an error function is crucial in machine learning applications. This section is devoted to translating classical backpropagation-based optimization to our framework. Given the input map $g\colon X_0 \to X$ and a loss function $\mathcal{L}\colon X \to \mathbb{R}$, we wish to find $f\colon X \to X$ such that the composition $\mathcal{L}h$ is minimized. To constrain the space of possible endofunctions (architectures and weights), we restrict the choice of f to a smoothly parameterized family of functions f_p, where p varies within a parameter space P.

Let P be a normed vector space of *parameters*. A *parametric machine* is a C^k family of machines $f(p, x)\colon P \times X \to X$ such that, given a C^k family of input functions $g(p, x_0)$, the family of resolvents $h(p, x_0)$ is also jointly C^k in both arguments. We call f a *parametric machine*, with *parameter space* P. Whenever f is a parametric machine, we denote by R_f its *parametric resolvent*, that is the only function in $C^k(P \times X, X)$ such that $R_f(p, x_0) = x_0 + f(p, R_f(p, x_0))$.

In practical applications, we are interested in computing the partial derivatives of the parametric resolvent function R_f with respect to the parameters and the inputs. This can

be done using the derivatives of f and a resolvent computation. Therefore, the structure and cost of the backward pass (backpropagation) are comparable to those of the forward pass. We recall that the backward pass is the computation of the dual operator of the derivative of the forward pass.

Theorem 3. *Let $f(p, x)$ be a parametric machine. Let R_f denote the parametric resolvent mapping $x = R_f(p, x_0)$. Then, the following equations hold:*

$$\frac{\partial R_f}{\partial x_0} = R_{\frac{\partial f}{\partial x}} \quad \textit{and} \quad \frac{\partial R_f}{\partial p} = \frac{\partial R_f}{\partial x_0}\frac{\partial f}{\partial p}. \tag{6}$$

Analogously, by considering the dual of each operator,

$$\begin{aligned} \left(\frac{\partial R_f}{\partial x_0}\right)^* &= \left(R_{\frac{\partial f}{\partial x}}\right)^*, \\ \left(\frac{\partial R_f}{\partial p}\right)^* &= \left(\frac{\partial f}{\partial p}\right)^* \left(\frac{\partial R_f}{\partial x_0}\right)^*. \end{aligned} \tag{7}$$

In other words, 1. the partial derivative of R_f with respect to the inputs can be obtained via a resolvent computation, and 2. the partial derivative of R_f with respect to the parameters is the composition of the partial derivative of R_f with respect to the inputs and the partial derivative of f with respect to the parameters.

The relevance of theorem 3 is twofold. On the one hand, it determines a *practical approach* to backpropagation for general parametric machines. Initially the resolvent of $\left(\frac{\partial f}{\partial x}\right)^*$ is computed on the gradient of the loss function $\mathcal{L}$. Then, the result is backpropagated to the parameters. In symbols,

$$\frac{\partial \mathcal{L}(R_f(p, x_0))}{\partial p} = \left(\frac{\partial f}{\partial p}\right)^* \left(\frac{\partial R_f}{\partial x_0}\right)^* D\mathcal{L}(R_f(p, x_0)).$$

The gradient $D\mathcal{L}(x)$, where $x = R_f(p, x_0)$, linearly maps tangent vectors of X to scalars and is therefore a cotangent vector of X. Indeed, the dual machine $\left(\frac{\partial f}{\partial x}\right)^*$ is an endofunction of the cotangent space of X. On the other hand, theorem 3 guarantees that in a broad class of practical cases the computational complexity of the backward pass is comparable to the computational complexity of the forward pass. We will show this practically in the following section.

3 Implementation and Performance

In this section, we shall analyze several standard and non-standard architectures in the machine framework, provide a general implementation strategy, and discuss memory usage and performance for both forward and backward pass. We consider a broad class of examples where f has both a linear component w_p (parametrized by p) and a nonlinear component σ. Different choices of w will correspond to different architectures (multi-layer perceptron, convolutional neural network, recurrent neural network) with or without shortcuts.

We split the space X as a direct sum $X = Y \oplus Z$, i.e., $x = (y, z)$, where y and z correspond to values before and after the nonlinear activation function, respectively. Hence, we write $f_p = w_p + \sigma$, with $\sigma\colon Y \to Z$ and $w_p\colon Z \to Y$. The machine equation $x = f_p(x) + x_0$ can be written as a simple system of two equations: $y = w_p z + y_0$ and $z = \sigma(y) + z_0$. Given cotangent vectors $u_0 \in Z^*, v_0 \in Y^*$ (which are themselves computed by backpropagating the loss to the machine output) we can run the following *dual machine*: $u = w_p^* v + u_0$ and $v = (D\sigma(y))^* u + v_0$. Then, eq. (7) boils down to the following rule to backpropagate (v_0, u_0) both to the input and the parameter space.

$$\begin{aligned} \left(\frac{\partial x}{\partial x_0}\right)^* (v_0, u_0) &= (v, u), \\ \left(\frac{\partial x}{\partial p}\right)^* (v_0, u_0) &= \left(\frac{\partial w_p}{\partial p}\right)^* v. \end{aligned}$$

In practical cases, the computation of the dual machine has not only the same structure, but also the same computational complexity of the forward pass. In particular, in the cases we will analyze, the *global* linear operator $w_p \in B(Y, Z)$ will be either a fully-connected or a convolutional layer, hence the dual w_p^* would be a fully-connected or a transpose convolutional layer respectively, with comparable computational cost, as shown practically in fig. 2. In our applications, the nonlinearity σ will be pointwise, hence the derivative $D\sigma(x)$ can be computed pointwise, again with comparable computational cost to the computation of σ. Naturally, for σ to act pointwise, we require that $Y \simeq Z \simeq \mathbb{R}^I$ for some index set I.

The first obstacle in defining a machine of the type $w_p + \sigma$ is practical. How should one select a linear operator w_p and a pointwise nonlinearity σ, under the constraint that $w_p + \sigma$ is a machine of finite depth? We adopt a general strategy, starting from classical existing layers and partitions on index spaces. We take l_p to be a linear operator (in practice, a convolutional or fully connected layer). We consider a partition $I = \bigsqcup_{i=0}^n I_i$ of the underlying index set I. For $i \in \{0, \dots, n\}$, let π_i^Y, π_i^Z be the projection from Y or Z to the subspace corresponding to $I_0 \sqcup \dots \sqcup I_i$. We can define the linear component of the machine as $w_p = \sum_{i=1}^n \left(\pi_i^Y - \pi_{i-1}^Y\right) l_p \pi_{i-1}^Z$, that is to say, it is a modified version of l_p such that outputs in index subsets depend only on inputs in previous index subsets. It is straightforward to verify that

$$\begin{aligned} X = Y \oplus Z &\supseteq \ker \pi_0^Y + Z \\ &\supseteq \ker \pi_0^Y + \ker \pi_0^Z \supseteq \ker \pi_1^Y + \ker \pi_0^Z \\ &\supseteq \ker \pi_1^Y + \ker \pi_1^Z \dots \\ &\supseteq \ker \pi_n^Y + \ker \pi_n^Y = 0 \end{aligned}$$

is a depth cofiltration for $w_p + \sigma$, hence $w_p + \sigma$ is a machine of depth at most $2n + 1$.

Generalized multi-layer perceptron Let us consider a generalization of the multi-layer perceptron in our framework. Let $x[c]$ (a point in machine space) be a tensor with one index, where $c \in \{1, \dots, n_c\}$. Let $I_0, \dots, I_n$ be a partition of $\{1, \dots, n_c\}$. We adapt the notation of the previous section: whenever possible, capital letters denote tensors corresponding to linear operators in lower case. Let $L[c_2, c_1]$ be a tensor with two indices $c_1, c_2 \in \{1, \dots, n_c\}$,

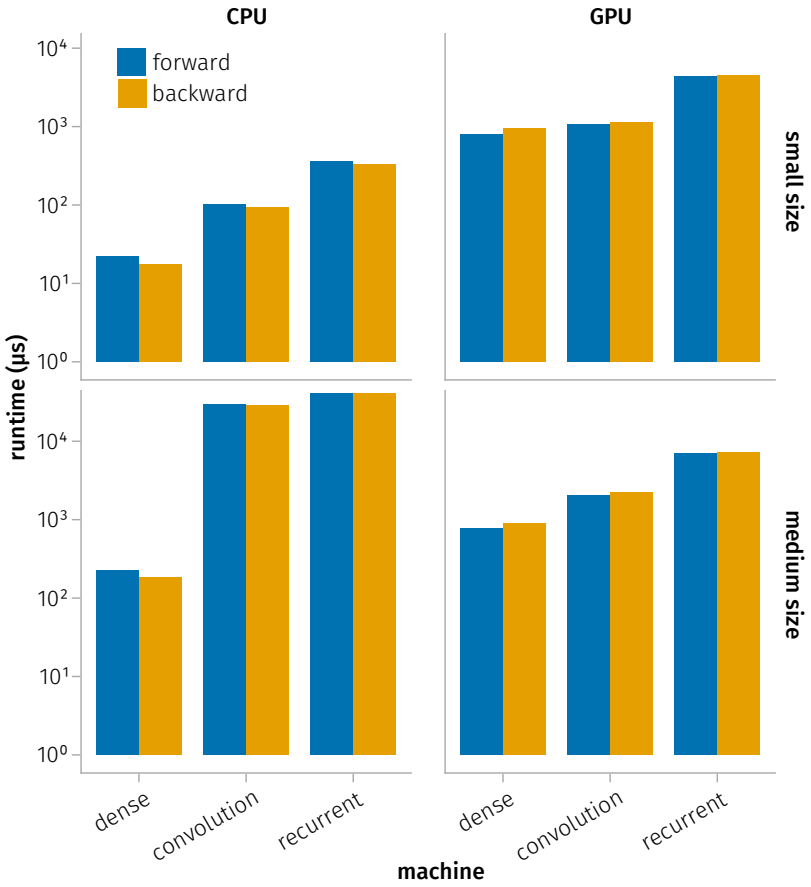

Figure 2: Ratio of runtime of backward pass over forward pass. The runtimes of backward and forward pass are comparable, across different models, problem sizes, and devices. The computation of the backward pass assumes that the forward pass has been computed already, and that its result is available. The backward pass denotes the backpropagation of cotangent vectors from machine space to input space. Backpropagating to parameter space requires an extra operation (see, e.g., eq. (12) for the dense case). See supplementary section S4 and table S1 for implementation details and timing values.

let $W = \sum_{i=1}^{n} \left(\pi_i^Y - \pi_{i-1}^Y\right) L \pi_{i-1}^Z$, and σ be a pointwise nonlinearity. We consider the machine equation

$$z = \sigma(y) + z_0, \tag{8}$$

$$y = Wz + y_0. \tag{9}$$

The backward pass can be computed via the dual machine computation

$$v = \sigma'(y) \odot u + v_0, \tag{10}$$

$$u = W^* v + u_0, \tag{11}$$

where σ' is the derivative of σ and $\odot$ is the Hadamard (elementwise) product, and the equations

$$Q = \sum_{i=1}^{n} \left(\pi_i^Y - \pi_{i-1}^Y\right) v z^* \pi_{i-1}^X, \tag{12}$$

where Q represents the cotangent vector (v_0, u_0) backpropagated to the parameters W. Equations (8) to (12) can be solved efficiently following the procedure described in algorithm 1. We describe the procedure exclusively for generalized multi-layer perceptrons, but the equivariant case (convolutional and recurrent neural networks) is entirely analogous.

Equivariant architectures We include under the broad term *equivariant* architectures (Bergomi et al. 2019) all machines whose underlying linear operator w_p is *translation-equivariant*—a shift in the input corresponds to a shift in the output. This includes convolutional layers for temporal or spatial data, as well as recurrent neural networks, if we consider the input as a time series that can be shifted forward or backward in time. The similarity between one-dimensional convolutional neural networks and recurrent neural networks will become clear in the machine framework. Both architectures can be implemented with the same linear operator l_p but different index space partitions.

The equivariant case is entirely analogous to the non-equivariant one. We consider the simplest scenario: one-dimensional convolutions of stride one for, e.g., time series data. We consider a discrete grid with two indices $t \in \{1, \ldots, n_t\}$ and $c \in \{1, \ldots, n_c\}$, referring to time and channel, respectively. Thus, the input data will be a tensor of two indices, $y[t, c]$. The convolutional kernel will be a tensor of three indices, $L[\tau, c_1, c_2]$, representing time lag (kernel size), input channel, and output channel, respectively. Let

Algorithm 1: Computation of non-equivariant machine.

Forward pass:

1: Initialize arrays y, z of size n_c and value $y = y_0$, $z = z_0$
2: **for** $i = 0$ to n **do**
3: $\quad y[I_i]$ += $W[I_i, :]z$, eq. (9)
4: $\quad z[I_i]$ += $\sigma\left(y[I_i]\right)$, eq. (8)
5: **end for**

Backward pass:

1: Initialize arrays u, v of size n_c and value $u = u_0$, $v = v_0$
2: **for** $i = n$ to 0 **do**
3: $\quad u[I_i]$ += $\left(L[:, I_i]\right)^* v$, eq. (11)
4: $\quad v[I_i]$ += $\sigma'\left(y[I_i]\right) \odot u[I_i]$, eq. (10)
5: **end for**
6: Initialize $Q = vz^*$, eq. (12)
7: Set $Q[I_j, I_i] = 0$, for all $j \leq i$, eq. (12)

$I_0, \ldots, I_n$ be a partition of $\{1, \ldots, n_t\} \times \{1, \ldots, n_c\}$.

We again denote $W = \sum_{i=1}^{n} \left(\pi_i^Y - \pi_{i-1}^Y\right) L \pi_{i-1}^Z$ and consider the machine equation $z = \sigma(y) + z_0$ and $y = W * z + y_0$. where $*$ denotes convolution. The backward pass can be computed via the dual machine computation $v = \sigma'(y) \odot u$ and $u = W *^t v + u_0$, where $*^t$ denotes transposed convolution, and the equations $\hat{Q}[\tau, c_1, c_2] = \sum_{t=\tau+1}^{n_t} z[t - \tau, c_1] v[t, c_2]$ and $Q = \sum_{i=1}^{n} \left(\pi_i^Y - \pi_{i-1}^Y\right) \hat{Q} \pi_{i-1}^X$, where Q represents the cotangent vector u_0 backpropagated to the parameters.

A common generalization of convolutional and recurrent neural networks Specific choices of the partition $I_1, \ldots, I_n$ will give rise to radically different architectures. Setting $I_i = \{1, \ldots, n_t\} \times J_i$ for some partition $J_0 \sqcup \cdots \sqcup J_n = \{1, \ldots, n_c\}$ gives a deep convolutional network with all shortcuts. On the other hand, setting $I_{t,i} = \{t\} \times J_i$ (with $I_{t,i}$ sorted by lexicographic order of (t, i)) yields a recurrent neural network with shortcuts in depth and time. The *dual machine* procedure is equivalent to a generalization of *backpropagation through time* in the presence of shortcuts.

Memory usage Machines' forward and backward pass computations are implemented differently from classical feedforward or recurrent neural networks: we store in memory a *global* tensor of all units at all depths, and we update it in place in a blockwise fashion. However, machines and neural networks have comparable memory usage during training. For instance, when computing the forward pass of a feedforward neural network without shortcuts, the outputs of all layers are needed stored in memory by the automatic differentiation engine to compute backpropagation.

4 Conclusions

We provide functional foundations for the study of deep neural networks. We define the abstract notion of *machine*, whose *resolvent* generalizes the computation of a deep neural network. It is a unified concept that encompasses several flavors of manually designed equivariant (convolutional (LeCun, Bengio et al. 1995) and recurrent (Werbos 1988) neural networks) and non-equivariant (multilayer perceptron, see (Rumelhart 1986)) neural network architectures.

Our approach attempts to answer seemingly simple questions: what are the *defining features* of deep neural networks? How can a deep neural network be specified? On these questions, current deep learning frameworks are broadly divided in two camps: *domain-specific languages* and *differentiable programming*. We aim to strike a balance between these opposite ends of the configurability spectrum. This is done via a principled, mathematical notion of machine: an endofunction of a normed vector space respecting a simple property. A subset of machines, machines of finite depth, are a *computable* generalization of deep neural networks inspired by *nilpotent* linear operators. The output of such a machine can be computed by iterating a simple sequence, whose behavior is remindful of *non-normal networks* (Hennequin, Vogels, and Gerstner 2012).

We use a general procedure to define several classes of machines of finite depth. As a starting point, we juxtapose linear and nonlinear continuous endofunctions of a normed vector space. Although this linear/nonlinear juxtaposition is shared with deep neural networks, the notion of composition of layers in neural networks is unfortunately ill-defined, especially in the presence of shortcut connections. In the proposed *machine framework*, the composition is replaced by the sum, and sequentiality by the weaker notion of independence, ensuring that the sum of machines of finite depth is again a machine of finite depth, whose resolvent (forward pass) can be computed explicitly and efficiently.

We prove that the backpropagation in the machine framework is entirely analogous to the forward pass and can be framed as a resolvent computation. Thus, we implement a backward pass whose computational cost is comparable to that of the forward pass, without resorting to automatic differentiation engines, provided that we can compute the derivative of the pointwise nonlinearity. In practice, the runtime of the backward and forward pass are comparable not incurring in automatic differentiation overhead (Srajer, Kukelova, and Fitzgibbon 2018). We believe that encompassing both forward and backward pass within a unified computational framework can be relevant in models where gradients are actively used in the forward pass (Drucker and Le Cun 1991; Varga, Csiszárik, and Zombori 2017; Zubov et al. 2021).

The proposed strategy to define machines of finite depth often generates architectures with a large number of shortcut connections. However, classical, sequential architectures can be recovered by forcing a subset of parameters to equal zero, thus cancelling the shortcut connections. However, this is only one of many possible ways of regularizing a machine. Several other approaches exist: setting to zero a different subset of parameters, as in the *lottery ticket hypothesis* (Frankle and Carbin 2018), penalizing large differences between adjacent parameters, or, more generally, choosing a representation of the parameter space with an associated notion of smoothness, as in kernel methods (Schölkopf, Smola, and Bach 2002). We intend to investigate the relative merits of these approaches in a future work.

References

Bai, S.; Kolter, J. Z.; and Koltun, V. 2019. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32.

Barham, P.; and Isard, M. 2019. Machine learning systems are stuck in a rut. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, 177–183.

Bergomi, M. G.; Frosini, P.; Giorgi, D.; and Quercioli, N. 2019. Towards a Topological–Geometrical Theory of Group Equivariant Non-Expansive Operators for Data Analysis and Machine Learning. *Nature Machine Intelligence*, 1–11.

Bezanson, J.; Edelman, A.; Karpinski, S.; and Shah, V. B. 2017. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1): 65–98.

Drucker, H.; and Le Cun, Y. 1991. Double backpropagation increasing generalization performance. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, 145–150 vol.2.

Frankle, J.; and Carbin, M. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Frostig, R.; Johnson, M. J.; and Leary, C. 2018. Compiling machine learning programs via high-level tracing. *Systems for Machine Learning*.

Gurumurthy, S.; Bai, S.; Manchester, Z.; and Kolter, J. Z. 2021. Joint inference and input optimization in equilibrium networks. *Advances in Neural Information Processing Systems*, 34.

Hennequin, G.; Vogels, T. P.; and Gerstner, W. 2012. Non-normal amplification in random balanced neuronal networks. *Physical Review E*, 86(1): 011909.

Innes, M.; Edelman, A.; Fischer, K.; Rackauckas, C.; Saba, E.; Shah, V. B.; and Tebbutt, W. 2019. A differentiable programming system to bridge machine learning and scientific computing. *arXiv preprint arXiv:1907.07587*.

LeCun, Y.; Bengio, Y.; et al. 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10): 1995.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*.

Moses, W.; and Churavy, V. 2020. Instead of Rewriting Foreign Code for Machine Learning, Automatically Synthesize Fast Gradients. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 12472–12485. Curran Associates, Inc.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic Differentiation in PyTorch. In *NIPS 2017 Workshop on Autodiff*.

Paszke, A.; Johnson, D.; Duvenaud, D.; Vytiniotis, D.; Radul, A.; Johnson, M.; Ragan-Kelley, J.; and Maclaurin, D. 2021. Getting to the Point. Index Sets and Parallelism-Preserving Autodiff for Pointful Array Programming. *arXiv preprint arXiv:2104.05372*.

Rackauckas, C.; Ma, Y.; Martensen, J.; Warner, C.; Zubov, K.; Supekar, R.; Skinner, D.; Ramadhan, A.; and Edelman, A. 2020. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*.

Rumelhart, D. 1986. Learning internal representation by back propagation. *Parallel distributed processing: exploration in the microstructure of cognition*, 1.

Saeta, B.; and Shabalin, D. 2021. Swift for TensorFlow: A portable, flexible platform for deep learning. *Proceedings of Machine Learning and Systems*, 3.

Schölkopf, B.; Smola, A. J.; and Bach, F. 2002. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press. ISBN 978-0-262-19475-4.

Srajer, F.; Kukelova, Z.; and Fitzgibbon, A. 2018. A benchmark of selected algorithmic differentiation tools on some problems in computer vision and machine learning. *Optimization Methods and Software*, 33(4-6): 889–906.

van Merrienboer, B.; Breuleux, O.; Bergeron, A.; and Lamblin, P. 2018. Automatic differentiation in ML: Where we are and where we should be going. *Advances in Neural Information Processing Systems*, 31: 8757–8767.

Varga, D.; Csiszárik, A.; and Zombori, Z. 2017. Gradient regularization improves accuracy of discriminative models. *arXiv preprint arXiv:1712.09936*.

Vertechi, P.; and Bergomi, M. G. 2020. Parametric machines: a fresh approach to architecture search. *arXiv preprint arXiv:2007.02777*.

Wang, F.; Decker, J.; Wu, X.; Essertel, G.; and Rompf, T. 2018. Backpropagation with callbacks: Foundations for efficient and expressive differentiable programming. *Advances in Neural Information Processing Systems*, 31: 10180–10191.

Werbos, P. J. 1988. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4): 339–356.

Winston, E.; and Kolter, J. Z. 2020. Monotone operator equilibrium networks. *Advances in neural information processing systems*, 33: 10718–10728.

Zubov, K.; McCarthy, Z.; Ma, Y.; Calisto, F.; Pagliarino, V.; Azeglio, S.; Bottero, L.; Luján, E.; Sulzer, V.; Bharambe, A.; et al. 2021. NeuralPDE: Automating Physics-Informed Neural Networks (PINNs) with Error Approximations. *arXiv preprint arXiv:2107.09443*.