Euro. Jnl of Applied Mathematics (2021), *vol.* 32, *pp.* 888–936

doi:10.1017/S0956792521000139

Structure-preserving deep learning

E. CELLEDONI[1], M. J. EHRHARDT[2], C. ETMANN[3], R. I. MCLACHLAN[4], B. OWREN[1], C.-B. SCHONLIEB[3] and F. SHERRY[3]

[1]*Department of Mathematical Sciences, NTNU, N-7491 Trondheim, Norway*
emails: elena.celledoni@ntnu.no; brynjulf.owren@ntnu.no
[2]*Institute for Mathematical Innovation, University of Bath, Bath BA2 7JU, UK*
email: m.ehrhardt@bath.ac.uk
[3]*Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Wilberforce Road, Cambridge CB3 0WA, UK*
emails: cetmann@damtp.cam.ac.uk; cbs31@cam.ac.uk; fs436@cam.ac.uk
[4]*School of Fundamental Sciences, Massey University, Private Bag 11-222, Palmerston North, New Zealand*
email: r.mclachlan@massey.ac.nz

(*Received 5 June 2020*; *revised 25 February 2021*; *accepted 19 April 2021*; *first published online 27 May 2021*)

Over the past few years, deep learning has risen to the foreground as a topic of massive interest, mainly as a result of successes obtained in solving large-scale image processing tasks. There are multiple challenging mathematical problems involved in applying deep learning: most deep learning methods require the solution of hard optimisation problems, and a good understanding of the trade-off between computational effort, amount of data and model complexity is required to successfully design a deep learning approach for a given problem.. A large amount of progress made in deep learning has been based on heuristic explorations, but there is a growing effort to mathematically understand the structure in existing deep learning methods and to systematically design new deep learning methods to preserve certain types of structure in deep learning. In this article, we review a number of these directions: some deep neural networks can be understood as discretisations of dynamical systems, neural networks can be designed to have desirable properties such as invertibility or group equivariance and new algorithmic frameworks based on conformal Hamiltonian systems and Riemannian manifolds to solve the optimisation problems have been proposed. We conclude our review of each of these topics by discussing some open problems that we consider to be interesting directions for future research.

Key words: Deep learning, ordinary differential equations, optimal control, structure-preserving methods

2020 Mathematics Subject Classification: 68T07, 65L (Primary); 65K, 49M (Secondary)

1 Introduction

Structure-preserving numerical schemes have their roots in geometric integration [62], and numerical schemes that is build on characterisations of PDEs as metric gradient flows [5], just to name a few. The overarching aim of structure-preserving numerics is to preserve certain properties of the continuous model, e.g. mass or energy conservation, in its discretisation. But structure preservation is not just restricted to play a role in classical numerical analysis of ODEs and PDEs. Indeed, through the advent of continuum interpretations of neural networks [60, 46, 47, 116],

structure preservation is also entering the field of deep learning. Here, the main objectives are to use the continuum model and structure-preserving schemes to derive stable and converging neural networks and associated training procedures and algorithms.

1.1 Neural networks

Neural networks are a rich class of machine learning models that can be leveraged for many different tasks including regression, classification, natural language processing, reinforcement learning and image generation [85]. While it is difficult to provide an all-encompassing definition for neural networks, they can generally be characterised as a combination of simple, parametric functions between *feature spaces*. These functions act as individual building blocks (commonly called the *layers* of the network). The main mechanism for combining these layers, which will be adopted in this work, is by function composition.

For any $k \in \{0, \dots, K-1\}$, let $\mathcal{X}^k$ denote a vector space (our *feature space*). While in most applications, these are simply finite-dimensional Euclidean spaces, we will assume more general structures (such as Banach spaces) when appropriate. With this, we then define a generic layer

$$f^k : \mathcal{X}^k \times \Theta^k \to \mathcal{X}^{k+1},$$

where Θ^k is the set of possible parameter values of this layer. A neural network

$$\begin{aligned} \Psi : \mathcal{X} \times \Theta &\to \mathcal{Y} \\ (x, \theta) &\mapsto z^K \end{aligned} \tag{1.1}$$

can then be defined via the iteration

$$\begin{aligned} z^0 &= x \\ z^{k+1} &= f^k(z^k, \theta^k), \quad k = 0, \dots, K-1, \end{aligned} \tag{1.2}$$

such that $\mathcal{X}^0 = \mathcal{X}$ and $\mathcal{X}^K = \mathcal{Y}$, where $\theta := (\theta^0, \dots, \theta^{K-1}) \in \Theta^0 \times \cdots \times \Theta^{K-1} =: \Theta$ denotes the entirety of the network's parameters. The first layer is commonly referred to as the neural network's *input layer*, the final layer as the neural network's *output layer* and all of the remaining layers are called *hidden layers*.

While the above definitions are still quite general, in practice several standard layer types are employed. Most ubiquitous are those that can be written as a learnable affine combination of the input, followed by a simple, non-linear function: the layer's *activation function*. The quintessential example are *fully connected layers*

$$\begin{aligned} f : \mathbb{R}^M \times (\mathbb{R}^{M' \times M} \times \mathbb{R}^{M'}) &\to \mathbb{R}^{M'} \\ (z, (A, b)) &\mapsto \sigma(Az + b), \end{aligned} \tag{1.3}$$

whose parameters are the *weight matrix* $A \in \mathbb{R}^{M' \times M}$ and the *bias vector* $b \in \mathbb{R}^{M'}$. Its activation function $\sigma : \mathbb{R}^{M'} \to \mathbb{R}^{M'}$ is typically applied component-wise, e.g. the hyperbolic tangent $[\tanh(z)]_i := \tanh(z_i)$ or the *Rectified Linear Unit* (ReLU) $[\mathrm{relu}(z)]_i := \max(0, z_i)$. For classification networks, the most common choice for the output layer's activation function is the *softmax* activation function, which is given by

$$[\mathrm{softmax}(z)]_i = \frac{\exp(z_i)}{\sum_{j=1}^{M'} \exp(z_j)},$$

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

such that the neural network's output entries can be regarded as the individual class membership probabilities of the input. An important extension to the concept of fully connected layers lies in *convolutional layers* [86] where the matrix-vector product is replaced by the application of a (multichannel) convolution operator. These are the main building block of neural networks used in imaging applications.

Suppose we are given a set of paired training data $(x_n, y_n)_{n=1}^N \subset \mathcal{X} \times \mathcal{Y}$, which is the case for predictive tasks like regression or classification. Training the model then amounts to solving the optimisation problem

$$\min_{\theta \in \Theta} \left\{ E(\theta) = \frac{1}{N} \sum_{n=1}^{N} L_n(\Psi(x_n, \theta)) + R(\theta) \right\}. \tag{1.4}$$

Here, $L_n(y) := L(y, y_n) : \mathcal{Y} \to \mathbb{R}_\infty$ is the loss for a specific data point where $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_\infty := \mathbb{R} \cup \{\infty\}$ is a general loss function, which usually satisfies $L \geqslant 0$ and $L(y_1, y_2) = 0$ if and only if $y_1 = y_2$. The function L_n is usually smooth on its effective domain $\{y \mid L_n(y) < \infty\}$ and convex. The term $R : \Theta \to \mathbb{R}_\infty$ acts as a regulariser which penalises and constrains unwanted solutions. In this setting, solving (1.4) is a form of empirical risk minimisation [117]. Typically, variants of stochastic gradient descent are employed to solve this task. The calculation of the necessary gradients is performed using the famous *backpropagation* algorithm, which can be understood as an application of reverse-mode auto-differentiation [92].

1.2 Residual networks and differential equations

In the following, we will discuss artificial neural network architectures that arise from the numerical discretisation of time-and parameter-dependent differential equations. Differential equations have a long history in the mathematical treatment of neural networks. Initially, neural networks were motivated by biological neurons in the brain. Mathematical models for the interactions of biological neurons are based on non-linear, time-dependent differential equations. These have inspired some famous artificial neural networks such as the Hopfield networks [68].

On a time interval $[0, T]$, an approximation to the solution of the differential equation is obtained at discrete times $0 = t^0 < t^1 < \cdots < t^K = T$, e.g. $t^k = k\,h$ and $h = T/K$, corresponding to the different layers of the network architecture. For a fixed final time T, the existence of an underlying continuous model guarantees the existence of a continuous limit as the number of layers goes to infinity and h goes to zero.

In the spirit of geometric numerical integration [62], we discuss structural properties of the artificial neural network as arising from the structure-preserving discretisation of a differential equation with appropriate qualitative features such as having an underlying symmetry, an energy function or a Lyapunov function or preserving a volume form or a symplectic structure.

Contrary to the 'classical' design principle for layers of affine transformations followed by activation functions (cf. (1.3)), the so-called *residual layers* are a variation to this principle, which has risen to become a standard design concept of neural networks. Here, the output of one such layer is again added to its input, which again defines a network, the ResNet [65], $\Psi : \mathcal{X} \times \Theta \to \mathcal{X}$, $\Psi(x, \theta) = z^K$ that is now given by the iteration

$$\begin{aligned} z^0 &= x \\ z^{k+1} &= z^k + \sigma(A^k z^k + b^k), \quad k = 0, \ldots, K-1, \end{aligned} \tag{1.5}$$

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

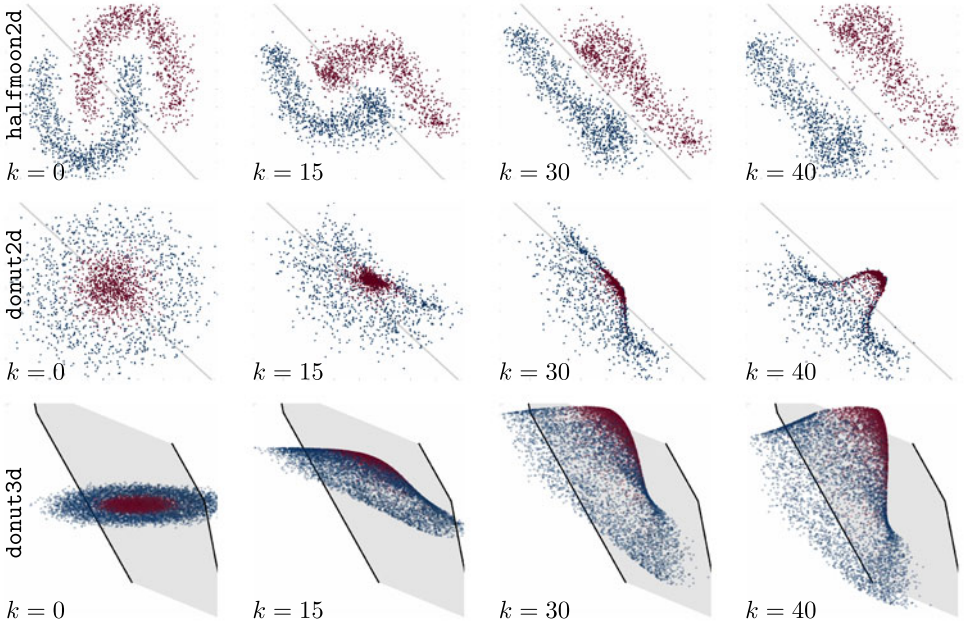

FIGURE 1. Evolutions of the ResNet model (1.6) for three different datasets. The upper two rows show evolutions in 2D and the lower row in 3D. The link function σ is the hyperbolic tangent and the data fit and regulariser are the squared 2-norm, $L_n(z) = \frac{1}{2}\|z - y_n\|_2^2$, $R(\theta) = \frac{\lambda}{2}\|\theta\|_2^2$.

if $\mathcal{X} = \mathcal{Y}$. If, on the other hand, the output space $\mathcal{Y}$ differs from $\mathcal{X}$, it is common to add another layer $\eta : \mathcal{X} \to \mathcal{Y}$ on top, which defines a network $\hat{\Psi} := \eta \circ \Psi : \mathcal{X} \to \mathcal{Y}$. This is, for example, a common scenario in classification, where the dimensionality of $\mathcal{Y}$ is determined by the number of classes.

It is easy to see that (1.5) corresponds to a particular discretisation of an ODE. To make the connection more precise, denote by $z^k := z(t^k)$, $A^k := A(t^k)$, and $b^k := b(t^k)$ samples of three functions $z : [0, T] \to \mathbb{R}^M$, $A : [0, T] \to \mathbb{R}^{M\times M}$ and $b : [0, T] \to \mathbb{R}^M$. With this notation we can write the ResNet (1.5) as $\Psi(x, \theta) = z(T)$ with

$$z(t^{k+1}) = z(t^k) + h\sigma(A(t^k)z(t^k) + b(t^k)), \quad k = 0, \ldots, K-1, \quad z(0) = x \tag{1.6}$$

if $T = K$ and thus $h = 1$. For general T, it can be readily seen that (1.6) corresponds to the forward Euler discretisation of

$$\dot{z}(t) = f(z(t), \theta(t)), \quad t \in [0, T], \quad z(0) = x \tag{1.7}$$

with $\theta := (A, b) : [0, T] \to \mathbb{R}^{M\times M} \times \mathbb{R}^M \cong \mathbb{R}^{M^2+M}$, and

$$f(z(t), \theta(t)) = \sigma(A(t)z(t) + b(t)). \tag{1.8}$$

Thus, there is a natural connection between the discrete deep learning problem (1.4), (1.5) and the optimal control formulation (1.4), (1.7). The dynamics of the ResNet (1.5) as a discretisation of (1.7) is depicted in Figure 1.

From here on we will suppress the dependence on t whenever it is clear from the context.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

It is well-known that the optimal control formulation can be phrased as a closed dynamical system by using Pontryagin's principle and that this results in a constrained Hamiltonian boundary value problem [12]. The Hamiltonian of this system is

$$H(z,p,\theta) = \langle p, f(z,\theta)\rangle, \tag{1.9}$$

where $p \in T_z^*\mathcal{X} \equiv \mathbb{R}^M$ is a vector of Lagrange multipliers. The dynamical system is then given by

$$\dot{z} = \partial_p H, \quad \dot{p} = -\partial_z H, \tag{1.10}$$

subject to the constraint

$$0 = \partial_\theta H. \tag{1.11}$$

The adjoint equation for p can be expressed as

$$\dot{p} = \partial_z f^T p, \quad p(T) = \frac{1}{N}\sum_{n=1}^{N} L_n'(z(T)). \tag{1.12}$$

In what follows we will review some of the guiding principles of structure-preserving deep learning, emphasising recent contributions on new neural networks architectures as discretisations of ODEs and PDEs in Section 2, invertible neural networks in Section 3, the interpretation of the training of neural networks as an optimal control problem in Section 4, equivariant neural networks in Section 5 and structure-preserving numerical schemes for the training of neural networks in Section 6.

2 Neural networks inspired by differential equations

2.1 Structure-preserving ODE formulations

In this section, we look at how the ODE formulation (1.7) can be restricted or extended in order to ensure that its flow has favourable properties. The relationship of these flow properties to the performance of the network and its optimisation is an emerging topic of investigation, presently being explored theoretically and experimentally.

We are zooming in on the forward problem itself, assuming that the parameters θ are fixed. By abuse of notation, we will in this section write $f(t,z)$ for $f(z,\theta(t))$ in that we consider $\theta(t)$ a known function of t. It is perhaps not so obvious what the desirable properties of the flow should be, and to some extent we lean here on prior work by Haber and Ruthotto [60] as well as Chang et al. [23].

A central consideration in both ODEs and neural networks is that of *stability*, so the well-developed stability theory of ODEs and their discretisation offers valuable insights into the stability of neural networks. However, it is important to qualify in which sense one is talking about stability, and how the different notions of stability of ODEs carry over to neural networks. A classical notion of stability of ODEs considers the stability of equilibrium points, i.e. points $z^0 = z(0)$ with $f(t,z^0) = 0$ for $t \in [0,\infty)$. This type of stability can be studied in terms of Lyapunov functions, that is, functions $V(z)$ that are non-increasing along solution trajectories. Functions that are constant along solutions are called first integrals, and a particular instance is the energy function of autonomous Hamiltonian systems.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

In this view, trajectories for perturbations of the initial value are studied for all $t \in [0, \infty)$. However, this notion of stability has limited applicability to neural networks, as one typically only considers finite time horizons $t \in [0, T]$ (cf. Section 1.2). Furthermore, equilibria hold no particular significance in neural networks.

Therefore, a more directly applicable view of stability is whether $z(T)$ does not change 'too much' when the initial value z^0 is perturbed. That is, small perturbations in the data should not lead to large deviations in the end result. Such considerations lead to asking whether $z(T)$ is, e.g. Lipschitz continuous (or more generally, uniformly continuous) in z^0. This means that for any two solutions $z_1(t)$ and $z_2(t)$ to (1.7), we have

$$\|z_2(T) - z_1(T)\| \leqslant C\|z_2(0) - z_1(0)\| \tag{2.1}$$

for some $C \geqslant 0$. It is well-known that this type of estimate can be obtained in several different ways, depending on the properties of the underlying vector field in (1.7). In particular, the Lipschitz constant C now depends on f as well as θ. Thus, in order to guarantee that C is not too large, one may want to impose certain restrictions on f or θ.

If $f(t, z)$ is Lipschitz in its second argument with constant $\mathrm{Lip}(f)$, then (2.1) holds with $C = e^{T\,\mathrm{Lip}(f)}$. Looking at (1.8), one can use $\mathrm{Lip}(f) = \mathrm{Lip}(\sigma) \max_t \|A(t)\|$ where $\mathrm{Lip}(\sigma)$ is a Lipschitz constant for the activation function σ. Note that for a discretisation of this ODE, the smallest Lipschitz constant not only depends on f and θ, but also on number of discretisation steps N, as well as the chosen numerical discretisation scheme. If one is to design a neural network as a discretised ODE which is supposed to be stable in the sense stated above, one has to take all of these into account.

For stability of non-linear ODEs one may, for instance, consider growth and contractivity in the L^2-norm using a one-sided Lipschitz condition. This is similar to the analysis proposed in [139]. Suppose there exists a constant $\nu \in \mathbb{R}$ such that for all admissible z_1, z_2, and $t \in [0, T]$ we have

$$\langle f(t, z_2) - f(t, z_1), z_2 - z_1 \rangle \leqslant \nu \|z_2 - z_1\|^2. \tag{2.2}$$

In this case it is easily seen that for any two solutions $z_1(t)$, $z_2(t)$ to (1.7) one has

$$\|z_2(t) - z_1(t)\| \leqslant \|z_2(0) - z_1(0)\|\, e^{\nu t},$$

so that the problem is non-expansive if $\nu \leqslant 0$ [64]. For instance, the vector field (1.8) satisfies (2.2) for σ absolutely continuous if

$$\nu \leqslant \sup_{t,D} \lambda_{\max}\left(\frac{DA(t) + (DA(t))^T}{2}\right),$$

where the supremum is taken over all diagonal matrices D with diagonal entries in $\sigma'(\mathbb{R})$. Note that even if $A(t)$ is skew-symmetric, the vector field (1.8) may still have a positive one-sided Lipschitz constant ν, but if we assume for instance that $\sigma'(s) \in [0, 1]$, it holds that $\nu \leqslant \frac{1}{2} \max_t \|A(t)\|_\infty$. Haber and Ruthotto [60] suggest using the eigenvalues of the linearised ODE vector field to analyse the stability behaviour of the forward problem. If the eigenvalues of the resulting Jacobian matrix have only non-positive real parts, it may be an indication of stability, yet for non-autonomous systems such an analysis may lead to wrong conclusions. It is indicated in [60] that by adding a regularisation term to the loss function in the training phase,

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

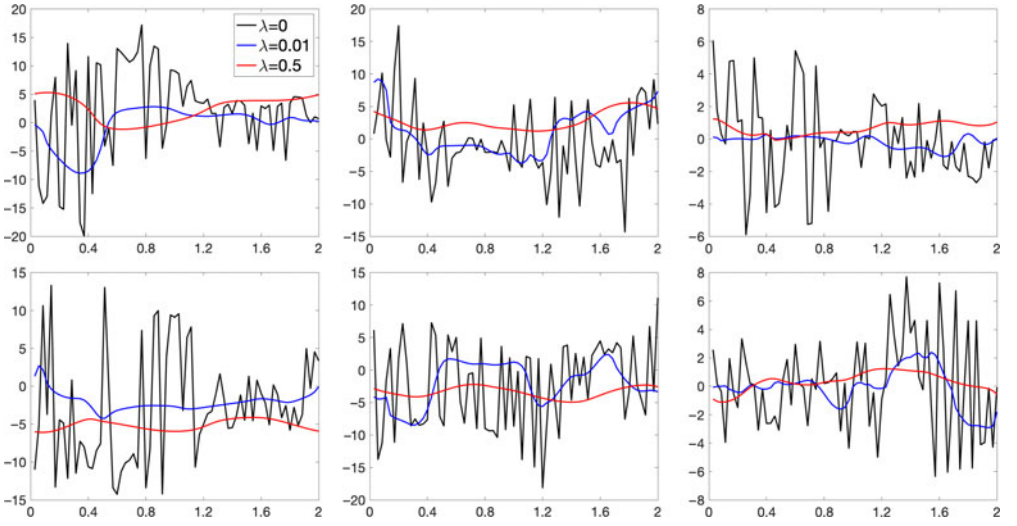

FIGURE 2. The variation of weights and biases over time for three different values of the regularisation parameter λ. The four leftmost plots show the elements of the weight matrix and the two rightmost ones show the biases.

the weights and biases may change slowly enough for the eigenvalue analysis to be valid when considering the stability of the forward problem. We illustrate this issue through an example. We use the standard vector field (1.8) and approximate its solution by the Euler method to obtain the standard ResNet iteration (1.6). Suppose that $R(\theta)$ in (1.4) is chosen to be a discretisation of

$$R_c(\theta(t)) = \tfrac{1}{2}\lambda \int_0^T \left\| \frac{d}{dt}\theta(t) \right\|^2 dt$$

in order to penalise rapid variations in the parameters $\theta(t)$ through the layers. A discretised version could be

$$R(\theta) = \frac{\lambda}{2h} \sum_{j=1}^{K} \|\theta_j - \theta_{j-1}\|_F^2,$$

where h is the time step $t^j - t^{j-1}$. We first train the model on the well-known spiral problem in two dimensions, using $K = 70$ two-dimensional layers and $N = 8000$ data points. We apply three different values for the regularisation parameter, $\lambda \in \{0, 0.01, 0.5\}$. In Figure 2 we have plotted how the weights and biases change over time.

The three trained models were run for a test dataset of 4000 points and the classification success rate was 99.2%, 94.9% and 94.3%, respectively, for $\lambda = 0, 0.01, 0.5$. We wanted to compare the eigenvalue analysis to the contractivity principle for this experiment. This was done locally in time by first computing the Jacobian matrix $J(t) := \frac{\partial f}{\partial z}(t, z(t))$ of the vector field $f(t, z(t)) = \sigma(A(t)z(t) + b)$ for every time step along a trajectory of the trained model. We compared the smallest real part of the eigenvalues of J with the logarithmic norm of J, which is a local version of the one-sided Lipschitz constant discussed above. The result is seen in Figure 3. Notice that the discrepancy between these two measures of stability does not seem to decrease as more regularisaton is added as one might expect.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

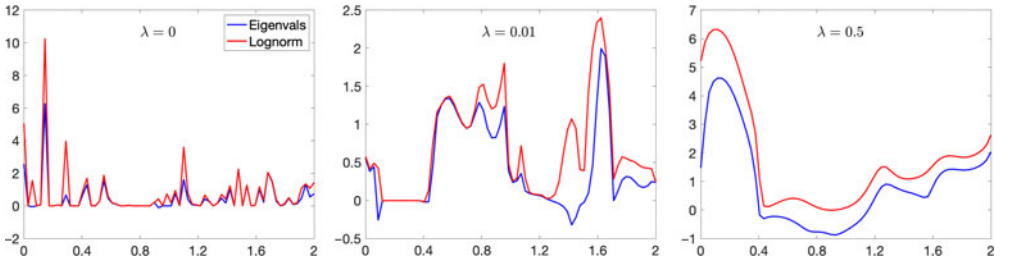

FIGURE 3. Eigenvalues and logarithmic norm for three values of the regularisation parameter λ as a function of time.

Clearly, the set of all vector fields of the form (1.8) include both stable and unstable cases. There are different ways of ensuring that a vector field has stable or contractive trajectories. One is to put restrictions on the family of matrices $A(t)$; another option is to alter the form of (1.8) either by adding symmetry to it or by embedding it in a larger space where it can be given desirable geometric properties. For example, by doubling the dimension, it is possible in several different ways to obtain a non-autonomous Hamiltonian vector field. In [23] and [60] several models are suggested. We mention a few of them in Section 2.1.2.

2.1.1 *Dissipative models and gradient flows*

For different reasons, it is desirable to use vector fields with good stability properties [135]. This will ensure that data which are close to each other in a chosen norm initially remain close as the features propagate through the neural network. A model suggested in [60] is to consider weight matrices $A(t)$ of the form

$$A(t) = S(t) - S(t)^T - \gamma I,$$

where $S(t)$ is arbitrary and γ is a small dissipation parameter. Flows of vector fields of the form $\dot{z} = A(t)z + b(t)$ exactly preserve the L^2-norm of the flow when $A(t)$ is skew-symmetric. The non-linearity will generally alter this behaviour, but adding a small amount of dissipation can improve the stability. It is, however, not guaranteed to reduce the one-sided Lipschitz condition.

Zhang and Schaeffer [139] analyse the stability of a model where the activation function σ is always chosen to be the *Rectified Linear Unit* (ReLU) function. The form of the discretised model is such that in the limit when h tends to zero, it must be replaced by a differential inclusion rather than an ODE of the form discussed above, meaning that $\dot{z} - f(t, z)$ belongs to some specified subdomain of $\mathbb{R}^M$. Their model vector field is

$$f(t, z) = -A_2(t)\sigma(A_1(t)z(t) + b_1(t)) + b_2(t). \tag{2.3}$$

Growth and stability estimates are derived for this class of vector fields, as well as for cases where restrictions are imposed on the parameters, such as $A_2(t)$ having positive elements or the case $A_1(t) = A_2(t)^T =: A(t)$ and $b_2(t) = 0$. For this last case, we consider for simplicity the ODE

$$\dot{z} = -A(t)^T\sigma(A(t)z + b(t)) = f(t, z), \tag{2.4}$$

which is a gradient system in the sense that $\dot{z} = -\nabla_z V$ with $V = \langle \gamma(A(t)z + b(t)), \mathbf{1}\rangle$ where $\gamma' = \sigma$ and $\mathbf{1}$ is the vector of ones.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

Theorem 2.1

(1) *Let $V(t,z)$ be twice differentiable and convex in the second argument. Then the gradient vector field $f(t,z) = -\nabla_z V(t,z)$ satisfies a one-sided Lipschitz condition (2.2) with $\nu \leqslant 0$.*

(2) *Suppose that $\sigma(s)$ is absolutely continuous and $0 \leqslant \sigma'(s) \leqslant 1$ a.e. in $\mathbb{R}$. Then (2.4) satisfies the one-sided Lipschitz condition (2.2) for any choice of parameters $A(t)$ and $b(t)$ with*

$$-\mu_*^2 \leqslant \nu_\sigma \leqslant 0,$$

where $\mu_* = \min_t \mu(t)$ and where $\mu(t)$ is the smallest singular value of $A(t)$. In particular, $\nu_\sigma = -\mu_*^2$ is obtained when $\sigma(s) = s$.

Proof. (1) We compute

$$\langle f(t,z_2) - f(t,z_1), z_2 - z_1\rangle = -\langle \nabla_z V(t,z_2) - \nabla_z V(t,z_1), z_2 - z_1\rangle$$

and define

$$\phi(\xi) = \frac{\mathrm{d}}{\mathrm{d}\xi} V(t, \xi z_2 + (1-\xi) z_1)$$

such that

$$\langle \nabla_z V(t,z_2) - \nabla_z V(t,z_1), z_2 - z_1\rangle = \phi(1) - \phi(0) = \int_0^1 \phi'(\xi)\,\mathrm{d}\xi.$$

Therefore, by the convexity of $V(t,z)$,

$$\langle f(t,z_2) - f(t,z_1), z_2 - z_1\rangle = -\left\langle \int_0^1 \nabla_z^2 V(t, \xi z_2 + (1-\xi) z_1)\,\mathrm{d}\xi\ (z_2 - z_1), z_2 - z_1 \right\rangle \leqslant 0.$$

(2) Let z_1 and z_2 be vectors in $\mathbb{R}^M$. Using (2.4) while suppressing the t-dependence in the parameters, we find

$$\langle f(t,z_2) - f(t,z_1), z_2 - z_1\rangle = -\langle \sigma(Az_2 + b) - \sigma(Az_1 + b), A(z_2 - z_1)\rangle. \tag{2.5}$$

For real scalars ζ, η and β we have

$$(\sigma(\zeta + \beta) - \sigma(\eta + \beta))(\zeta - \eta) = \int_\eta^\zeta \sigma'(s + \beta)\ \mathrm{d}s\,(\zeta - \eta)$$

and since $0 \leqslant \sigma'(s + \beta) \leqslant 1$ a.e. we have

$$0 \leqslant (\sigma(\zeta + \beta) - \sigma(\eta + \beta))(\zeta - \eta) \leqslant (\zeta - \eta)^2.$$

Using this inequality in (2.5) we obtain

$$-\|Az_2 - Az_1\|^2 \leqslant \langle f(t,z_2) - f(t,z_1), z_2 - z_1\rangle \leqslant 0.$$

Since $\|A(z_2 - z_1)\|^2 \geqslant \mu_*^2 \|z_2 - z_1\|^2$ the result follows. □

Remark. *In Theorem 2.1 we restricted the class of activation functions to be absolutely continuous with $0 \leqslant \sigma'(s) \leqslant 1$. This is true for many of the activation functions proposed in the literature, in particular for the commonly used activation functions ReLU $\sigma = \tanh$ and the sigmoid $\sigma(s) = 1/(1 + \exp(-s))$.*

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

2.1.2 *Hamiltonian vector fields*

One may take inspiration from mechanical systems and introduce the Hamiltonian framework. Separable Hamiltonian systems are defined in terms of kinetic and potential energy functions T and V

$$H(t,z,p) = T(t,p) + V(t,z).$$

This leads to differential equations of the form

$$\dot{z} = \partial_p H = \partial_p T, \tag{2.6}$$

$$\dot{p} = -\partial_z H = -\partial_z V. \tag{2.7}$$

There are different ways to construct a Hamiltonian system from (1.8). In [23] the following model is suggested:

$$\dot{z} = A_1(t)^T \sigma_1(A_1(t)p + b_1(t)),$$
$$\dot{p} = -A_2(t)^T \sigma_2(A_2(t)z + b_2(t)).$$

Let $\gamma_i : \mathbb{R} \to \mathbb{R}$ be such that $\gamma_i'(t) = \sigma_i(t),\ i = 1, 2$. The corresponding Hamiltonian is

$$T(t,p) = \gamma_1(A_1(t)p + b_1(t))\mathbf{1}, \quad V(t,z) = \gamma_2(A_2(t)z + b_2(t))\mathbf{1},$$

where $\mathbf{1} = (1, \ldots, 1)^T$. A simple case is obtained by choosing $\sigma_1(s) := s$, $A_1(t) \equiv I$, $b_1(t) \equiv 0$ and $\sigma_2(s) := \sigma(s)$, which after eliminating p yields the second-order ODE

$$\ddot{z} = -\partial_z V = -A_2(t)^T \sigma(A_2(t)z + b_2(t)).$$

A second example is considered by Chang et al. [23], who set $\sigma_1 = \sigma_2 = \sigma$ and apply the resulting deep Hamiltonian neural network discretised by an explicit symplectic integrator to several standard image classification tasks. They find that its performance is competitive with state-of-the-art ResNet networks, requiring a similar number of parameters but much less memory. When the training set was drastically reduced or the number of layers was very large (up to 1202), performance was better than ResNet, a result they attribute to the intrinsic stability of the network architecture.

From the outset, it might not be so clear which geometric features such a non-autonomous Hamiltonian system has. There seem to be at least two ways to understand this problem [97]. Let us assume that $u = (z,p) \in T^*\mathbb{R}^M \equiv \mathbb{R}^M \oplus \mathbb{R}^M$ with 'positions' z and 'momenta' p forming the phase space. We have the natural symplectic form $\omega_0 = \mathrm{d}p \wedge \mathrm{d}z$ on the phase space. This form can be represented using the Darboux matrix J as

$$\omega_0(\xi, \eta) = \langle \xi, J\eta \rangle, \quad J = \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix}$$

and the Hamiltonian vector field $f(t,z,p)$ defined via $\mathrm{d}H(\cdot) = \omega_0(f(t,z,p), \cdot)$.

Let us now introduce as phase space $T^*\mathbb{R}^M \times \mathbb{R}$ by including the time t as a dependent variable. The natural projection is $\tau : (z,p,t) \mapsto (z,p)$. The space $T^*\mathbb{R}^M \times \mathbb{R}$ can then be furnished with a *contact structure* defined as

$$\omega_H = \tau^* \omega_0 - \mathrm{d}H \wedge \mathrm{d}t.$$

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

The first term is just the original form, ignoring the t-component of the tangents, while the second form is only concerned with the 't-direction'. One can write the matrix of ω_H by adding a row and a column to J coming from the second term $-\mathrm{d}H \wedge \mathrm{d}t$

$$J_H = \left(\begin{array}{c|c} J & -\nabla_u H \\ \hline (\nabla_u H)^T & 0 \end{array}\right).$$

The resulting vector field is then $f_H = (f^T, f_t)^T$ and can be expressed through the equations

$$i_{f_H}\omega_H = 0, \quad i_{f_H}\mathrm{d}t = 1$$

where i_{f_H} stands for the interior derivative of the form by f_H, i.e. i_{f_H} applied to the two-form ω_H is the one-form $\alpha = i_{f_H}\omega_H$ such that $\alpha(\eta) = \omega(f_H, \eta)$ for all vector fields η. The form ω_H is preserved along the flow, but H is not.

Extended phase space. One can in fact recover an autonomous Hamiltonian problem from the non-autonomous one by adding two extra dependent variables, say t and p_t. We do this by considering the manifold $T^*(\mathbb{R}^M \times \mathbb{R})$. One needs a new projection $\mu\colon (z, p, t, p_t) \mapsto (z, p, t)$ and we can define an extended (autonomous) Hamiltonian on $T^*(\mathbb{R}^M \times \mathbb{R})$ as

$$K = H \circ \mu + p_t$$

and two-form

$$\Omega_0 = \mu^* \tau^* \omega_0 + \mathrm{d}p_t \wedge \mathrm{d}t.$$

The corresponding matrix, J_E, is just the original Darboux matrix J where each of the $M \times M$ identity matrices has been replaced by $(M+1) \times (M+1)$ identity matrices. The extended Hamiltonian K is exactly preserved along the flow so that the new conjugate momentum variable p_t will keep track of how $H(z, p, t)$ varies along solutions. The resulting vector field f_E can be written as $dK(\cdot) = \Omega_0(f_E, \cdot)$ and in coordinates the ODE system becomes

$$\dot{z} = \partial_p H, \quad \dot{p} = -\partial_z H, \quad \dot{t} = 1, \quad \dot{p}_t = -\partial_t H. \tag{2.8}$$

We see at once that since the equations for $\dot{z}, \dot{p}$ do not depend on p_t and since the solution for t is explicitly given, we solve the same problem as before. After solving for z and p, we obtain p_t independently by integration. The second thing one may observe is that if a numerical method ϕ_h has the property that[1]

$$\phi_h \circ \mu = \mu \circ \phi_h,$$

then what we obtain by just considering the first $2M$ components of the numerical solution to the extended system is precisely the same as what we would have obtained applying the same method to the original non-autonomous problem. This observation was used by Asorey et al. [7] to define what is meant by canonical transformations in the non-autonomous case, and we refer to this paper for further results on the structural connections between the two systems. In applications to deep learning, one should note that geometric properties of the solution can

[1] It is common to assume that a given numerical integrator is defined on systems in any dimension.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

mostly be deduced from the extended system rather than the original non-autonomous one; there are numerical methods, which preserve energy or the symplectic form Ω_0 and rigorous results can be proved for the long time behaviour of such integrators [62].

2.2 Structure-preserving numerical methods for the ODE model

The rationale behind proposing ODE formulations with geometric structure is to enforce a controlled behaviour of the solution as it is propagated through the hidden layers of the network. It is therefore also important that when the ODE is approximated by a numerical method, this behaviour should be retained by the numerical scheme.

2.2.1 *Numerical methods preserving dissipativity*

When numerically solving ODEs, which satisfy the one-sided Lipschitz condition (2.2), a desirable property of the numerical scheme would be that it contracts two nearby numerical solutions whenever $\nu \leqslant 0$. That is, it should satisfy

$$\|z_1^{k+1} - z_2^{k+1}\| \leqslant \|z_1^k - z_2^k\| \tag{2.9}$$

for each time step k, preferably without too severe restrictions on the time step h. Methods that can achieve this for any step size exist, and are called B-stable. There are many B-stable methods, and for Runge–Kutta methods, B-stability can be easily checked by the condition of algebraic stability. Examples of B-stable methods are the implicit Euler method and all implicit Runge–Kutta methods in the Gauss, Radau IA, Radau IIA and Lobatto IIIC classes [64]. In deep learning algorithms it has been more usual to consider explicit schemes since they are much cheaper per step than implicit ones, but they are subject to restrictions on the time step. For instance, the explicit Euler method is unstable for any step size when applied even to linear constant coefficient systems where there are eigenvalues of the coefficient matrix on the imaginary axis. To consider contractivity of explicit schemes, we need to replace (2.2) by a different monotonicity condition

$$\langle f(t, z_2) - f(t, z_1), z_2 - z_1 \rangle \leqslant \bar{\nu} \|f(t, z_2) - f(t, z_1)\|^2,$$

where we assume $\bar{\nu} < 0$. For every Runge–Kutta method with strictly positive weights b_i there is a constant r such that the numerical solution is contractive whenever the step size satisfies

$$h < -2\bar{\nu} r.$$

The value of r can be calculated for every Runge–Kutta method with positive weights. For example, for the explicit Euler method as well as for the classical fourth-order method of Kutta one has $r = 1$ [38].

2.2.2 *Numerical methods preserving energy or symplecticity*

For autonomous Hamiltonian systems there are two important geometric properties which are conserved. One is the energy or Hamiltonian $H(z, p)$; the other is the symplectic structure, a closed non-degenerate differential two-form. These two properties are also the main targets for structure-preserving numerical methods.

All Hamiltonian deep learning models presented here can be extended to separable non-autonomous canonical systems (2.6)–(2.7). Such systems preserve the symplectic two-form

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

$\mathrm{d}p \wedge \mathrm{d}q$ and there are many examples of explicit numerical methods that also preserve this same form in the sense that $\mathrm{d}p^{k+1} \wedge \mathrm{d}q^{k+1} = \mathrm{d}p^k \wedge \mathrm{d}q^k$. The simplest example of such a scheme is the symplectic Euler method, defined for the variables z and p as

$$z^{k+1} = z^k + h\,\partial_p T(t^{k+1}, p^k), \quad p^{k+1} = p^k - h\,\partial_z V(t^{k+1}, z^{k+1}). \tag{2.10}$$

The symplectic Euler method is explicit for separable Hamiltonian systems and is an example of a splitting method [100]. Many other examples and a comprehensive treatment of symplectic integrators can be found in [62]. When applying symplectic integrators to Hamiltonian problems, one has the important notion of backward error analysis. The numerical approximations obtained can be interpreted as the exact flow of a perturbed system with Hamiltonian $\tilde{H}(z,p) = H(z,p) + hH_2(z,p) + \cdots$[2]. This partly explains the popularity of symplectic integrators, since many of the characteristics of Hamiltonian flows are inherited by the numerical integrator.

Similarly, there exist many numerical methods that preserve energy exactly for autonomous problems. For instance, there is a large class of schemes based on discrete gradients [101]. A discrete gradient of a function $H(z)$ is a continuous map $\overline{\nabla} H : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}^M$ which satisfies the two conditions

$$H(z_2) - H(z_1) = \langle \overline{\nabla} H(z_1, z_2), z_2 - z_1 \rangle, \quad \overline{\nabla} H(z,z) = \nabla H(z), \quad \forall z, z_1, z_2 \in \mathbb{R}^M.$$

For the Hamiltonian ODE $\dot{z} = J\nabla H(z)$ it is easily seen that the method defined by

$$\frac{z^{k+1} - z^k}{h} = J\overline{\nabla} H(z^k, z^{k+1})$$

is energy preserving in the sense that $H(z^k) = H(z^0)$ for all $k > 0$. There are many choices of discrete gradients, but most of them lead to implicit schemes and therefore have the disadvantage of being computationally expensive.

Another disadvantage is that even if it makes sense to impose energy conservation for the extended autonomised system explained above for deep learning models, it is not clear what that would mean for the original problem. It remains an open problem to understand the potential for and the benefits of using energy-preserving schemes for non-autonomous Hamiltonian systems in deep learning.

2.2.3 *Splitting methods and shears*

Splitting methods are very popular time integration methods that can be easily applied to preserve geometric structures of the underlying ODE problems such as symplecticity, invertibility and some symmetries. The idea of splitting and composition is simply to split the vector field in the sum of two (or more) vector fields, to integrate separately each of the parts and compose the corresponding flows. For example, splitting a Hamiltonian vector field into the sum of two Hamiltonian vector fields and composing their flows results into a symplectic integrator. If the individual parts are easy to integrate exactly, the resulting time integration method has often low computational cost. We refer to [100] for an overview on splitting methods.

[2] This is a divergent asymptotic series, but truncation is possible at the expense of an exponentially small remainder term.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

An ODE on $\mathbb{R}^d$ is a shear if there exist a basis of $\mathbb{R}^d$ in which the ODE takes the form

$$\dot{y}_i = 0, \quad i = 1, \dots, k,$$
$$\dot{y}_i = f_i(y_1, \dots, y_k), \quad i = k+1, \dots, d.$$

A diffeomorphism on $\mathbb{R}^d$ is called a shear if it is the flow of a shear. Splitting vector fields into the sum of shears allows the computation of the individual flows exactly simply applying the forward Euler method to each term.

Consider the shears $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}^n$:

$$(z, p) \mapsto (z + g(p), p),$$
$$(z, p) \mapsto (z, p + f(z)).$$

In the case of autonomous, separable, Hamiltonian systems, the symplectic Euler method (2.10) can be seen as the composition of two such maps where

$$g(p) := h\partial_p T(p), \qquad f(z) := -h\partial_z V(z).$$

Another popular example is the Störmer–Verlet integrator or leapfrog integrator which is also the composition of two shears. It is possible to represent quite complicated functions with just two shears. The 'standard map' (also known as Chirikov–Taylor map) is an area preserving, chaotic composition of two shears, much studied in dynamics and accelerator physics.

Shears are useful tools to construct numerical time integration methods that preserve geometric features. As already mentioned symplecticity is one such property (if f and g are gradients), another is the preservation of volume and, as we will see in Section 3, shears can be used to obtain invertible neural networks.

2.3 Features evolving on Lie groups or homogeneous manifolds

If the data belongs naturally to a differentiable manifold $\mathcal{M}$ of finite dimension and $f(z, \theta)$ in (1.7) is a vector field on $\mathcal{M}$ for all $\theta \in \Theta$ then $z(t) \in \mathcal{M}$. Concrete examples of datasets in this category are manifold-valued images and signals. One example is diffusion tensor imaging consisting of tensor data, which at each point (i, j) in space corresponds to a 3×3 matrix $A_{i,j} \in \mathrm{Sym}^+(3)$, the 3×3 symmetric positive definite matrices [36]. If $m \times l$ is the number of pixels in the image then $\mathcal{M} = \mathrm{Sym}^+(3)^{m \times l}$. Another example is InSAR imaging data taking values on the unit circle S^1, in which $\mathcal{M} = \left(S^1\right)^{m \times l}$ [115]. In both these examples denoising autoencoder neural networks [126] can be used to learn image denoising. The loss function (1.4) can take the form

$$L(\Psi(x, \theta), y) = \sum_{i,j=1}^{l,m} \mathrm{d}(\Psi(x, \theta)_{i,j}, y_{i,j}),$$

where $x = (x_{1,1}, \dots, x_{l,m}) \in \mathcal{M}$ is the noisy image $y = (y_{1,1}, \dots, y_{l,m}) \in \mathcal{M}$ is the source image and d is a distance function on $\mathrm{Sym}^+(3)$ or S^1, respectively. For example, in the case of $\mathrm{Sym}^+(3)$ with $T_A\mathrm{Sym}^+(3) = \mathrm{Sym}(3)$, a possible choice of Riemannian metric is

$$\langle X, Y \rangle_{T_A\mathcal{M}} := \mathrm{trace}(A^{-\frac{1}{2}} X A^{-1} Y A^{-\frac{1}{2}}),$$

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

where X and Y are symmetric 3×3 and A is symmetric and positive definite, and with trace denoting the trace of matrices. With this metric $\mathrm{Sym}^+(3)$ is a Riemannian symmetric space and is geodesically complete [130].

A third example concerns classification tasks where the data are Lie group valued curves for activity recognition. Here, $\mathcal{M} = G^m$ with m the number of points where the curve is sampled and $G = \mathrm{SO}(3)$ is the group of rotations [18]. A loss function can be built using the following distance between two sampled curves $c_1 = (c_{1,1}, \ldots, c_{1,m}) \in G^m$, $c_2 = (c_{2,1}, \ldots, c_{2,m}) \in G^m$:

$$\mathrm{d}(c_1, c_2) = \sum_{i=1}^{m-1} \left\| \frac{\log(c_{1,i+1}c_{1,i}^{-1})}{\| \log(c_{1,i+1}c_{1,i}^{-1})\|_{\mathfrak{g}}^{\frac{1}{2}}} - \frac{\log(c_{2,i+1}c_{2,i}^{-1})}{\| \log(c_{2,i+1}c_{2,i}^{-1})\|_{\mathfrak{g}}^{\frac{1}{2}}} \right\|_{\mathfrak{g}},$$

where $\mathfrak{g}$ is the Lie algebra of G, $\|\cdot\|_{\mathfrak{g}}$ is a norm deduced from an inner product on $\mathfrak{g}$, and $\log\colon G \to \mathfrak{g}$ denotes the matrix logarithm. An important feature of this distance is that it is re-parametrisation invariant and taking the infimum over all (discrete) parametrisations of the second curve one obtains a well defined distance for curves independent of their parametrisation [19].

The ODE (1.7) should in this setting be adapted to be an ODE on $\mathcal{M}$. If $\mathcal{M}$ is a $(d-p)$-dimensional submanifold of $\mathbb{R}^d$, $\mathcal{M} := \{y \mid g(y) = 0\}$, $g\colon \mathbb{R}^p \to \mathbb{R}^d$, then the ODE (1.7) can be modified adding constraints. Alternatively, sometimes one can consider intrinsic manifold formulations, which allow the representation of the data with $d-p$ degrees of freedom instead of d. The numerical integration of this ODE must then respect the manifold structure.

2.4 Open problems

2.4.1 *Geometric properties of Hamiltonian models*

Autonomous Hamiltonian systems and their numerical solution are by now very well understood. For such models, one has conservation of energy that is attractive when considering stability of the neural network map. The same cannot, to our knowledge, be said about the non-autonomous case. One can approach this problem in different ways. One is to consider canonicity in the sense of [7] and study the geometric properties of canonical transformations via the extended autonomous Hamiltonian system. This is, however, not so straightforward for a number of reasons. One issue is that every level set of the extended Hamiltonian will be non-compact. Another issue is that the added conjugate momentum variable, denoted p_t above, is artificial and is only used to balance the time-varying energy function.

One should also consider the effect of regularisation, since one may expect that smoothing of the parameters will cause the system to behave more similarly to the autonomous problem. See Section 4.2 of this paper.

2.4.2 *Measure-preserving models*

The most common example of an invariant measure is phase space volume. In invertible networks, some attention is given to volume-preserving maps, see [113, 40] and also Section 3 of this paper. For the ODE model, this amounts to the vector field $f(t, z)$ being divergence free, and there are several ways to parametrise such vector fields. All Hamiltonian vector fields are

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

volume preserving, but the converse is not true. Volume-preserving integrators can be obtained, for example, via splitting and composition methods [100].

2.4.3 *Manifold based models*

A generalisation of most of the approaches presented in this section to the manifold setting is still missing. For data evolving on manifolds the ODE (1.7) should be adapted to be an ODE on $\mathcal{M}$. Just as an example, a gradient flow on $\mathcal{M}$ analogous to (2.4) can be considered starting from defining a function $V\colon \mathcal{M} \times \Theta \to \mathbb{R}$ using the antiderivative of the activation function σ and the Riemannian metric. The Hamiltonian formulations of Section 2.1.2 could be also generalised to manifolds in a similar way, starting from the definition of the Hamiltonian function.

An appropriate numerical time discretisation of the ODEs for deep learning algorithms must also guarantee that the evolution through the layers remains on the manifold so that one can make use of the Riemannian structure of $\mathcal{M}$ and obtain improved convergence of the gradient descent optimisation, see Section 6.2. The numerical time discretisation of this ODE must then respect the manifold structure and there is a vast literature on this subject, see, for example, [62, Chapter IV]. For numerical integration methods on Lie groups and homogeneous manifolds including symplectic and energy-preserving integrators, see [72, 22].

3 Invertible neural networks and normalising flows

In the previous sections, we have viewed certain neural networks as discretisations of continuous systems. In the following, we will return to the classical view of discrete networks, which are additionally endowed with a certain structure. *Invertible neural networks*, i.e. bijective neural networks, are such an example. They have been an emerging topic in deep learning over the last few years. They are naturally connected to neural networks that are inspired by ordinary differential equations, as flows of ODEs are themselves invertible. Two of the main applications of invertible neural networks lie in memory-efficient backpropagation as well as generative modelling with density estimation. For simplicity, in the following we will use the abbreviation $f_k = f^k(\cdot, \theta^k)$ when appropriate. Much like, in general, invertible networks are typically parametrised as a function composition $\Psi = f_K \circ \cdots \circ f_1$, with the additional constraint that each layer f_k is a bijective function. The inverse can then simply be computed via $\Psi^{-1} = {f_1}^{-1} \circ \cdots \circ {f_K}^{-1}$. A main restriction this imposes on the architecture is the fact that for each layer $f^k : \mathcal{X}^k \times \Theta^k \to \mathcal{X}^{k+1}$, one has $\dim(\mathcal{X}^k) = \dim(\mathcal{X}^{k+1})$, where the $\mathcal{X}^k$ are the feature spaces. As a consequence, the dimension of the input space determines the dimensions of the feature spaces in a (fully) invertible neural network.

3.1 Types of invertible layers

Invertible networks and layers can roughly be divided into two categories: Those that are algebraically invertible and those that are inverted with a numerical approximation scheme.

3.1.1 *Coupling layers*

Coupling layers [40] work by splitting a layer's input into two parts and applying a suitable transformation that is easily invertible for one of the two parts. Mathematically, for an M-dimensional input, the index set $I = \{1, \ldots, M\}$ is partitioned into two sets. In convolutional

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

networks, the partition is often done channel-wise, i.e. the channels are split into two sets. A different type of partitioning is *invertible downsampling* (also known as a *masking* or *squeeze* operation [40]). With $x = (x_1, x_2) \in \mathbb{R}^{M_1} \times \mathbb{R}^{M_2}$ and $y = (y_1, y_2) \in \mathbb{R}^{M_1} \times \mathbb{R}^{M_2}$, a coupling layer $g\colon x \mapsto y$ is defined via the mapping

$$\begin{aligned} y_1 &= x_1, \\ y_2 &= \gamma(x_2, f(x_1)), \end{aligned} \tag{3.1}$$

where $f : \mathbb{R}^{M_1} \to \mathbb{R}^{M_2}$ and $\gamma : \mathbb{R}^{M_2} \times \mathbb{R}^{M_1} \to \mathbb{R}^{M_2}$ is invertible in the first argument. Then $g^{-1} : y \mapsto x$ is given by

$$\begin{aligned} x_1 &= y_1, \\ x_2 &= \gamma^{-1}(y_2, f(y_1)), \end{aligned} \tag{3.2}$$

where γ^{-1} is the inverse of γ in its first argument (for fixed second argument). The *coupling law* γ can be as simple as $\gamma : (a, b) \mapsto a + b$, with which f is called an *additive coupling layer* [40], such that $\gamma^{-1} : (c, b) \mapsto c - b$. Another commonly used class of coupling layers are *affine coupling layers* [40, 41]. More complex, but in theory, more expressive coupling laws can be constructed from strictly monotonic splines [45]. Note that (3.1) and (3.2) are shear mappings (Section 2.2.3), which in the case of ODEs are used to construct, e.g. symplecticity-preserving numerical solutions.

There is in principle no restriction on the function f. This allows for the utilisation of arbitrarily expressive subnetworks f (e.g. a sequence of convolutional layers with non-linear activation functions), without rendering g (algebraically) non-invertible. The numerical stability of this inversion may still pose an issue. Stability guarantees (both for the forward and the inverse mapping) can, for instance, be controlled via the Lipschitz constant of f and γ, as shown in [10]. This mirrors stability considerations of ODEs (see Section 2.1), where guarantees can be formulated as conditions on the Lipschitz constants. Note that while f has to map to $\mathbb{R}^{M_2}$, the individual layers which comprise f may change the dimensionality throughout this subnetwork – only the final layer has to transform the data to the required space $\mathbb{R}^{M_2}$. Since coupling layers perform learnable computations only on a part of the input, the partitioning should change throughout the network, for example, by switching the roles of x_1 and x_2.

As demonstrated by Teshima et al. [120], neural networks built using coupling layers are *universal approximators* of diffeomorphic functions. This means that they can approximate a diffeomorphic function up to arbitrary precision, given a sufficiently large architecture. This can be seen as an extension of various other universal approximation theorems (cf. [37, 69]), which state similar results for other classes of neural networks and functions.

3.1.2 *Invertible layers through iterative schemes*

Another class of invertible layers are those that are invertible with an iterative scheme. Invertible residual networks [9] are a special case of the commonly used residual networks [65], which allow for the inversion with a simple fixed-point iteration. As in Section 1.2, a residual layer can be framed as a function $g\colon \mathbb{R}^M \to \mathbb{R}^M$ with

$$g(x) = x + f(x), \tag{3.3}$$

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

where $f\colon \mathbb{R}^M \to \mathbb{R}^M$ is a function parametrised by neural network layers, called the *residual block*. For fixed x, let $y := x + f(x)$, so that $x = y - f(x)$. A fixed point z^* of the function $\Phi_y\colon z \mapsto y - f(z)$ is thus in the preimage of y under g, i.e. $g(z^*) = y$. Note that

$$\|\Phi_y(a) - \Phi_y(b)\| = \|f(b) - f(a)\| \leqslant \mathrm{Lip}(f) \cdot \|b - a\|, \tag{3.4}$$

where $\mathrm{Lip}(f)$ is the Lipschitz constant of f. According to Banach's fixed-point theorem, if $\mathrm{Lip}(f) < 1$, then Φ_y is guaranteed to have a *unique* fixed point, which is the inverse of y under g, i.e. $z^* = g^{-1}(y)$. This inverse can be approximated to arbitrary precision via the iteration

$$x^i = y - f(x^{i-1}) \tag{3.5}$$

for any initial value x^0. While most common layer types (such as dense or convolutional layers equipped with activation functions with bounded derivative) are Lipschitz, the condition $\mathrm{Lip}(f) < 1$ is not necessarily met. Behrmann et al. [9] enforce the required Lipschitz constraint by *spectral normalisation*: Let the linear layer A_θ depend linearly on its parameters θ (e.g. convolutional layers without non-linear activation functions and biases; these are linear layers that linearly depend on their kernel). Then $A_{\bar\theta}$ parametrised by

$$\bar\theta := c \cdot \theta / \|A_\theta\|_2 \tag{3.6}$$

has Lipschitz constant $\mathrm{Lip}(A_{\bar\theta}) = c$. As a consequence, the layer $f(x) = \sigma(A_{\bar\theta} \cdot x + b)$ has Lipschitz constant $\mathrm{Lip}(f) \leqslant c$ for any activation function σ with $\mathrm{Lip}(\sigma) \leqslant 1$, where b is a bias vector. Thus, by updating the layers' parameters via spectral normalisation according to (3.6) after each gradient step, the required Lipschitz constraint for invertibility is guaranteed, if one chooses $c < 1$. In practice, the spectral norm $\|A_\theta\|_2$ is calculated with the power method. To save computations, Behrmann et al. [9] only perform a single power iteration, but reuse the estimation of the leading singular vector from the previous training step as the initial guess. While the power method (with a finite number of iterations) technically only provides a lower bound to $\|A_\theta\|_2$, the Lipschitz constraint is still met in practice [9].

In Figure 4, the dynamics of an invertible residual network and its non-invertible counterpart in 1D are compared. Both images depict a 30-layer residual network, which was trained to learn the identity function on $[-2, 2]$ on 15 noisy samples. The Lipschitz constant of the residual blocks f were restricted to be smaller than one, whereas no such restriction was put on the non-invertible network. As invertible functions on the real line are strictly monotonic, the paths created by the invertible networks are non-crossing, which can happen in non-invertible networks.

In the case of invertible residual networks, the connection to neural networks as numerical solutions to ODEs is particularly strong. As noted in Section 1.2, ResNets can generally be viewed as Euler discretisations of an ODE, if one views the activations, weights and biases as observations of time-dependent variables. Fittingly, Behrmann et al. [9] originally motivate the inversion of such a ResNet by looking at the dynamics of the associated ODE backwards in time.

3.1.3 *Linear invertible components*

Above, two general approaches for constructing invertible, non-linear layers were presented. In the following, we list a few *linear*, invertible layers, which are used to increase the expressivity of invertible networks.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

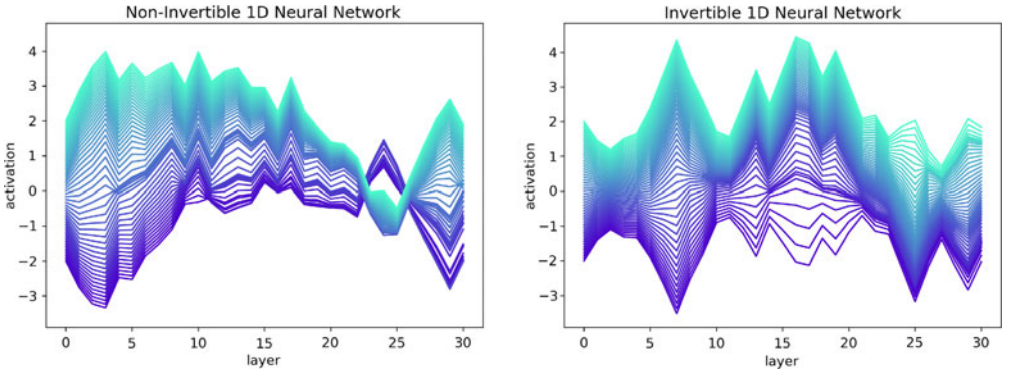

FIGURE 4. Visualisation of the dynamics of a standard (non-invertible) ResNet compared to an invertible ResNet in 1D, adapted from [9]. Each line represents the activations at different layers for a particular input of the network. The networks were trained to approximate the identity function from 15 noisy samples.

Coupling layers (Section 3.1.1) work by dimension splitting, which in the context of convolutional neural networks usually consists in splitting the channels into two groups, which are processed independently from one another. This is in contrast to standard, multichannel convolutions, where each output channel depends on all input channels. In order to overcome this limitation, the channels can subsequently be linearly combined (via an endomorphism). If the matrix of these coefficients is invertible, the automorphism created this way can be integrated as a linear layer into the invertible network framework. These invertible matrices can be parametrised in different ways: One approach [78] is to directly parametrise the desired matrix via a LU factorisation (with additional fixed permutation), which can then be inverted. While not discussed in the original publication, the diagonal entries themselves can be, e.g. parametrised to be larger than some $\varepsilon > 0$ to enforce stable invertibility. This is known as *invertible* 1×1 *convolution*, because this 'channel mixing' can equivalently be realised as a convolution with (in 2D) one-by-one filters. An extension to this idea to *convolutional* layers (with larger filters) is presented in [67]. A computationally particularly cheaply (and stably) invertible class of matrices are orthogonal matrices. In [110], these are parametrised as a sequence of Householder transformations. Other possible parametrisations of (special) orthogonal matrices include exponentials of skew-symmetric matrices, Cayley transforms or sequences of Givens rotations. For a discussion about the numerical optimisation of such classes of parameter matrices compare also the forthcoming Section 6 and in particular (6.7).

Another type of linear, invertible layer is determined by invertible down- and up-sampling operations for image data. While classical down- and up-sampling methods (such as bilinear interpolation or nearest-neighbour methods) from image processing are inherently non-invertible (as they change the dimensionality of the input), it is possible to construct *invertible* down- and up-sampling methods. Since the dimensionality of their output must be the same as the dimensionality of their input, decreasing (or increasing) the spatial dimensionality of an image *must* be accompanied by a suitable increase (respectively decrease) in the number of channels. One such transformation for invertible down-sampling is the *pixel shuffle* transform [40], which subsamples the pixels and reorders them into new channels. The inverse of this transformation is an invertible up-sampling operation. In [52], this is generalised to learnable invertible down- and up-sampling operations, which contain the (inverse) pixel shuffle as a special case. The general

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

idea is to construct *orthogonal* strided convolutional operators, where the kernel size matches the stride $s \in \mathbb{N}^d$, with spatial dimensionality d. It can be shown that by a specific reordering of an orthogonal τ-by-τ matrix (where $\tau = s_1 \cdots s_d$) into a filter kernel, the resulting convolution is orthogonal. This implies that the inverse is simply given by the corresponding transposed convolution. The authors propose to parametrise the required orthogonal matrices via exponentiating skew-symmetric matrices. As the matrix exponential is a surjective mapping from the Lie algebra of skew-symmetric matrices to the special orthogonal matrices, *any* special orthogonal matrix can be parametrised this way.

3.2 Applications

3.2.1 *Memory-efficient backpropagation*

One possible area of application for invertible neural networks is memory-efficient backpropagation [57]. Let $z^{k+1} = f(z^k, \theta^k)$ be the output of a neural network's layer with non-linear mapping $f : \mathcal{X}^k \times \Theta^k \to \mathcal{X}^{k+1}$, where θ^k are the layer's parameters. Let L be the loss of the network. Then

$$\nabla_{\theta^k} L = \left(\partial_{\theta^k} z^{k+1}\right)^* \nabla_{z^{k+1}} L = \left(\partial_{\theta^k} f(z^k, \theta^k)\right)^* \nabla_{z^{k+1}} L$$

provides the weight gradient necessary for the training of the network. For the calculation of this gradient, one needs both the gradient of the loss with respect to the output node (i.e. $\nabla_{z^{k+1}} L$), as well as the ability to calculate the derivative $\partial_{\theta^k} f(z^k, \theta^k)$. Unless f is linear, the calculation of the derivative requires access to z^k, meaning that z^k needs to be *stored in memory*. This typically represents the bulk of the memory demand in training neural networks. If, however, f is *invertible*, one can simply calculate z^k from z^{k+1} via

$$z^k = f^{-1}(z^{k+1}, \theta^k)$$

for the additional computational cost of calculating the inverse (where f^{-1} is the inverse of f in its first argument). Thus, instead of storing activations in memory during the forward pass, intermediate activations can simply be successively reconstructed from the output layer's activation. The memory requirement of training an invertible network using this scheme is thus independent of the number of invertible layers.

3.2.2 *Invertible networks as subnetworks*

In practice, many classical applications of neural networks such as classification, regression or segmentation do not typically map between vector spaces of the same dimensionality. Hence, a bijective function that maps between these two spaces does not exist, which is why a fully invertible neural network typically cannot solve the desired task. It is, however, possible to use an invertible neural networks as a *subnetwork* in a neural network. For example, if $\Psi : \mathcal{X} \to \mathcal{X}$ is an invertible network and $\eta : \mathcal{X} \to \mathcal{Y}$ is a suitable non-invertible layer, the combined network $\eta \circ \Psi : \mathcal{X} \to \mathcal{Y}$ can be used for a classification task of predicting labels in $\mathcal{Y}$ from features $\mathcal{X}$. As demonstrated in [74], such a neural network can have competitive performance to a comparably sized residual network [65] on ImageNet [39]. Adding an output layer which transforms between spaces mirrors the approach for discretised ODEs, see Section 1.2.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

The use of invertible networks as subnetworks allows for the utilisation of the memory efficient backpropagation for these subnetworks (cf. Section 3.2.1), while the respective gradients of non-invertible subnetworks can be calculated conventionally.

3.2.3 *Density estimation and generative modelling*

Aside from generative adversarial networks (GANs) and variational autoencoders (VAEs), *normalising flows* are another class of machine learning models that can be used to artificially generate data. While GANs are able to generate realistic-looking images [75], they lack the ability to estimate the likelihood of data under the generative model at hand. Likewise, VAEs can only estimate a lower bound of the likelihood – the variational lower bound. This is in contrast to normalising flows, which are trained via maximum likelihood estimation.

Like most generative models, normalising flows generate data from a simple base distribution (usually Gaussian) via a learned transformation. Let the random variable z have probability density function q, for which we will write $z \sim q(z)$. For any diffeomorphism f, it holds that

$$x := f^{-1}(z) \sim q(z) \cdot \left|\det(\partial_z f^{-1}(z))\right|^{-1}$$

due to the change-of-variables theorem[3] [14]. This means that for the probability density of x (denoted P), the log-likelihood of x can be expressed as

$$\log P(x) = \log q(f(x)) + \log \left|\det(\partial_x f(x))\right| . \tag{3.7}$$

Let f be parametrised by an invertible neural network, i.e. $f(x) = \Psi(x, \theta)$ for all $x \in \mathcal{X}$. Given training data $(x_n)_{n=1}^N$, the minimiser of

$$\min_{\theta \in \Theta} \left\{ E(\theta) = -\frac{1}{N} \left(\sum_{n=1}^{N} - \log q(\Psi(x_n, \theta)) + \log \left|\det(\partial_{x_n} \Psi(x_n, \theta))\right| \right) \right\} \tag{3.8}$$

is a maximum likelihood estimator of the training data under the neural network. Models trained this way are called *normalising flows*, because they are usually trained to convert complicated data into normal distributions. In this framework, artificial data can be generated by sampling $f^{-1}(z)$, where $z \sim q(z)$. An example of learning the 'two half-moons' dataset this way is provided in Figure 5. This case highlights that continuously transforming a normal distribution into a distribution whose probability density function has disconnected support is not possible. However, the normalising flow instead assigns low probability density to areas outside of the support, which results in a distribution that resembles the true distribution.

One of the main difficulties in designing normalising flows lies in the evaluation (respectively differentiation) of the determinant term. Note that for a neural network of the form (1.2) it holds that

$$\det(\partial_{z^0} \Psi(z^0, \theta)) = \det(\partial_{z^{K-1}} z^K) \cdots \det(\partial_{z^0} z^1),$$

[3]Here, it suffices to consider invertible functions f that are only *locally* diffeomorphic in x. This is of practical relevance, because many neural networks are only locally differentiable due to the use of piecewise differentiable activation functions such as ReLU.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

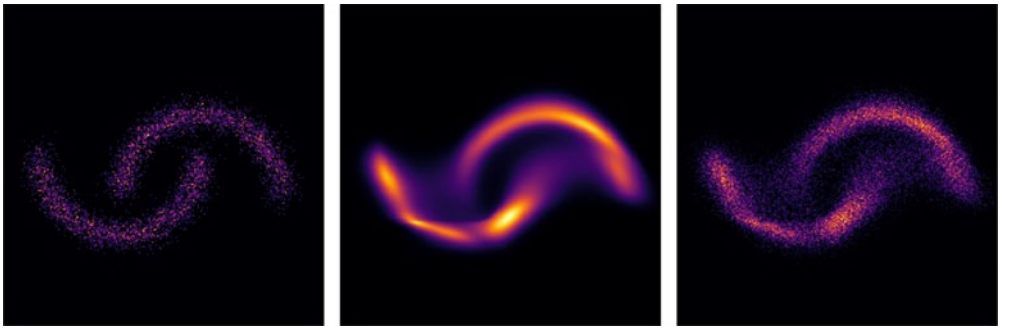

FIGURE 5. Left: 'Two half-moons' dataset. Middle: An approximation of the probability density function of the 'two half-moons' dataset by a normalising flow, generated by the code from [24]. Right: Points sampled from the approximated distribution.

such that the computation of the determinant for the whole network can be performed in a layer-by-layer fashion. Still, naïvely computing each determinant by the Leibniz formula yields prohibitive computation times. Depending on the types of layers used, the structure of the individial layers' Jacobians makes employing different determinant identities possible. If, for example, the Jacobian is lower triangular (as is the case with coupling layers [40]), the determinant reduces to the product of the Jacobian's diagonal entries. Invertible residual networks [9] on the other hand induce no such structure. By using the spectral norm constraint on f in equation (3.3), it holds that $\left|\det(\partial_{z^0}\Psi(z^0,\theta))\right| = \det(\partial_{z^0}\Psi(z^0,\theta))$ and using the identity from [131], the authors show that

$$\log\left|\det(\partial_{z^0}\Psi(z^0,\theta))\right| = \operatorname{trace}\left(\log\partial_{z^0}\Psi(z^0,\theta)\right)$$

holds. The matrix logarithm is then approximated as a truncated Taylor series, whereas for the evaluation of the trace, the Hutchinson estimator [70] is employed. An extension to this concept is presented in [24], where the truncation with a fixed number of steps is replaced by a 'Russian roulette' estimator, a type of Monte Carlo estimator. This estimator introduces a stochastic truncation of the series, which results in an unbiased estimator of the log-likelihood (3.7).

In the context of normalising flows, another connection to ODEs becomes apparent: As noted in [25], a continuous version of the change-of-variables theorem may be used to formulate a continuous normalising flow, which in turn can be solved numerically via a discretisation scheme. A maximum likelihood estimator can in turn be obtained by training a neural ODE [25].

3.3 Open problems

3.3.1 *Fast inverses without coupling*

Coupling layers (Section 3.1.1) allow both for quick forward and inverse computations. Their expressivity is, however, hindered by the fact that expressive, learned transformations are only applied to a part of their input, while the remaining portion of the input is not transformed. However, while this problem can be mitigated by using multiple coupling layers (with different partitions of the input), it may be desirable to construct invertible layers where each output

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

neuron depends on all input neurons (as is the case with 'conventional' fully connected or convolutional layers). Invertible residual layers (Section 3.1.2) on the other hand fulfil this criterion, but they rely on a numerical inversion via a fixed-point iteration, which requires multiple evaluations of the forward operation. Furthermore, in order to guarantee their invertibility, a Lipschitz constant needs to be controlled, which requires additional computation time. For practical purposes, it would be desirable to combine both methods' advantages in order to obtain both fast and expressive invertible layers.

3.3.2 *Stability guarantees*

As discussed in [10], controlling the Lipschitz constant of invertible layers is a conceptionally simple, but practically costly method of guaranteeing a certain stability of the inverse. Furthermore, as discussed above, they need to be controlled in order to guarantee the convergence of the iterative scheme for the inversion of residual layers. In the interest of computational efficiency, is there a computationally cheaper way to control the Lipschitz constant than via the proposed power method? Work in this direction has already been performed in [24], as the authors experiment with Lipschitz constants corresponding to mixed norms.

In a similar vein, are there alternative ways of guaranteeing stability other than by controlling the Lipschitz constant?

4 Deep Learning meets optimal control

As outlined in the introduction, supervised deep learning with the ResNet architecture can be seen as solving a discretised optimal control problem. This observation has been made in [29, 46, 60, 93] with further extensions to PDEs described in [116]. An advantage of this strategy is that many tools from scientific computing for the solution of optimal control problems, ODEs and PDEs can be utilised, see, for example, [59, 106] for layer-parallel training.

Recall (1.4)

$$\min_{\theta\in\Theta}\left\{E(\theta)=\frac{1}{N}\sum_{n=1}^{N}L_n(\Psi(x_n,\theta))+R(\theta)\right\}, \tag{4.1}$$

where the neural network $\Psi(\cdot,\theta):\mathcal{X}\to\mathcal{X}$ is either defined by a recursion like (1.5) or the solution at the final time of an ODE (1.7). Another approach is to view the training as an optimisation problem over $\Theta\times\mathcal{X}^N$ where $\mathcal{X}$ is the space of the dynamics z, i.e.

$$\min_{(\theta,z)\in\Theta\times\mathcal{X}^N}\left\{E(\theta,z)=\frac{1}{N}\sum_{n=1}^{N}L_n(z_n(T))+R(\theta)\right\} \tag{4.2a}$$

$$\text{such that}\quad \dot{z}_n=f(z_n,\theta(t)),\quad z_n(0)=x_n,\quad n=1,\dots,N. \tag{4.2b}$$

In machine learning the reduced formulation (4.1) is much more common than (4.2).

4.1 Training algorithms

The discrete or continuous optimal control problem can be solved in multiple ways, some of which we will discuss next.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

4.1.1 *Derivative-based algorithms*

The advantage of the reduced problem formulation (4.1) is that it is a smooth and unconstrained optimisation problem such that if Θ is a Hilbert space, derivative-based algorithms such as gradient descent are applicable. Due to the non-linearity of Ψ we can at most expect to converge to a critical point $E'(\theta) = 0$ with a derivative-based algorithm. The most basic algorithm to train neural networks is stochastic gradient descent (SGD) [114]. Given an initial estimate θ^0, SGD consists of iterating two simple steps. First, sample a set of data points $\mathcal{S}^j \subset \{1, \dots, N\}$ and then iterate

$$\theta^{j+1} = \theta^j - \tau^j \frac{1}{|\mathcal{S}^j|} \sum_{n \in \mathcal{S}^j} (L_n \circ \Psi(x_n, \cdot))'(\theta^j). \tag{4.3}$$

Other first-order algorithms used for deep learning are the popular Adam method [77] and the Gauss–Newton [60] method. A discussion on the convergence of SGD is out of the scope of this paper. Instead, we focus on how to compute the derivatives $(L_n \circ \Psi(x_n, \cdot))'(\theta^j)$ in the continuous model which is the central building block for all first-order methods. This following theorem is very common and the main idea dates back to Pontryagin [109]. This formulation is inspired by Bonnans [16, Lemma 2.47].

Theorem 4.1 *Assume that f and L_n are of class C^1 and that f is Lipschitz with respect to z. Let $z_n \in W^{1,\infty}([0, T], \mathbb{R}^M)$ be the solution of the ODE (4.2b) with initial condition x_n and p_n the solution of the adjoint equation*

$$\dot{p}_n = -\partial_z f(z_n(t), \theta(t))^T p_n, \quad p_n(T) = L_n'(z_n(T)). \tag{4.4}$$

Then the Fréchet derivative of $A := L_n \circ \Psi(x_n, \cdot) : L^\infty := L^\infty([0, T], \mathbb{R}^{M^2+M}) \to \mathbb{R}$ at $\theta \in L^\infty$ is the linear map $B := A'(\theta) : L^\infty \to \mathbb{R}$, where

$$Bh = \int_0^T \langle \partial_\theta f(z_n(t), \theta(t))^T p_n(t), h(t) \rangle \, \mathrm{d}t. \tag{4.5}$$

For finite dimensional θ a similar theorem can be proven, which dates back to Grönwall in 1919, see [58, 63]. Defining neural networks via ODEs and computing gradients via continuous formulas similar to Theorem 4.1 has been first proposed in [25] with extensions in [56] and [44]. In the deep learning community this is referred to as *Neural ODEs*.

Similar to Theorem 4.1 a discrete version can be derived when the ODE (1.7) is discretised with a Runge–Kutta method [12, 61]. For simplicity we just state the special case of the explicit Euler discretisation (ResNet (1.5)) here.

Theorem 4.2 ([12]) *Let z_n be the solution of the ResNet (1.5) with initial condition x_n. If p_n satisfies the recursion*

$$p_n^{k+1} = p_n^k - h \partial_z f(z_n^k, \theta^k)^T p_n^{k+1}, \quad k = 0, \dots, K-1, \quad p_n^K = L_n'(z_n^K), \tag{4.6}$$

then the derivative of $A := L_n \circ \Psi(x_n, \cdot)$ is given by $\partial_{\theta^k} A(\theta) = h \partial_\theta f(z_n^k, \theta^k)^T p_n^{k+1}$.

Some readers will spot the similarity between Theorem 4.2 and what is called *backpropagation* in the deep learning community. This observation was already made in the 1980s, see, e.g. [83].

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

If all functions in (4.5) are discretised by constant functions on $[t^k, t^{k+1}]$, then the gradient of the discrete problem (4.6) approximates the Fréchet derivative of the continuous problem (4.5). In more detail, let $e^{kj} \in L^\infty$ be supported on $[t^k, t^{k+1}]$ and $e^{kj}(t) = e^j \in \mathbb{R}^{M^2+M}$ with $e^j_i = 1$ if $j = i$ and 0 else. If we denote by $A = L_n \circ \Psi(x_n, \cdot)$ the data fit using the ODE solution (1.7) and $\tilde{A} = L_n \circ \Psi(x_n, \cdot)$ the data fit with the ResNet (1.5), then

$$A'(\theta)e^{kj} = \int_{t^k}^{t^{k+1}} \langle \partial_\theta f(z_n(t), \theta(t))^T p_n(t), e^{kj}(t) \rangle \, \mathrm{d}t \tag{4.7}$$

$$\approx h \langle \partial_\theta f(z_n(t^k), \theta(t^k))^T p_n(t^{k+1}), e^j \rangle = \partial_{\theta^k_j} \tilde{A}(\theta). \tag{4.8}$$

In other words, in this case of piecewise constant functions, discretise-then-optimise is the same as the optimise-then-discretise.

4.1.2 *Method of successive approximation*

Another strategy to train neural networks has been recently proposed by [88] and is not based on the gradients of the reduced formulation (4.1) but on the necessary conditions of (4.2) also known as Pontryagin's maximum principle [109] instead. Given an initial estimate of θ^0, each iteration of the *method of successive approximation* has three simple steps. First, solve (4.2b) with θ^j which we denote by z^j_n, i.e.

$$\dot{z}^j_n = f(z^j_n, \theta^j(t)), \quad z^j_n(0) = x_n, \quad n = 1, \dots, N. \tag{4.9}$$

Then solve the adjoint equation (4.4) with θ^j, z^j_n and denote the solution by p^j_n, i.e.

$$\dot{p}^j_n = -\partial_z f(z^j_n(t), \theta^j(t))^T p^j_n, \quad p^j_n(T) = L'_n(z^j_n(T)), \quad n = 1, \dots, N. \tag{4.10}$$

The third and final step is to maximise the Hamiltonian (1.9) given p^j_n, z^j_n. That is, for any $t \in [0, T]$ the update is defined as

$$\theta^{j+1}(t) := \arg\max_\theta \left\{ H(z^j_n(t), p^j_n(t), \theta) = \frac{1}{N} \sum_{n=1}^{N} \langle p^j_n(t), f(z^j_n(t), \theta) \rangle \right\}. \tag{4.11}$$

Note that this algorithm is potentially well-defined also in the non-smooth case, i.e. if f is not differentiable with respect to θ. If f is indeed smooth, $R = 0$ and $\theta^{j+1} = \theta^j$, then

$$\frac{1}{N} \sum_{n=1}^{N} \partial_\theta f(z^j_n(t), \theta^j(t))^T p^j_n(t) = 0. \tag{4.12}$$

and the Fréchet derivative of (4.1) vanishes. Analysis, extensions and numerical examples of the method of successive approximation are presented in [88].

4.2 Regularisation

The training of a neural network (1.4) may include explicit regularisation and several different regularisers R have been proposed in the literature, e.g. [12, 122, 60, 103, 111, 49], which we want to discuss in this section. Before we dive into the specifics of these regularisers, we would like to answer the question if regularisation is necessary for training neural networks.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

Example 4.3 In order to shed some light on this, we consider the most trivial example which is taken from [122]. To this end we use ResNet (1.5) and let $N = 1$, $K = 1$, $M = 1$, $L_1(z) = (z-1)^2$, $x_1 = 0$ and $\sigma = \tanh$. When no regulariser is present, $R = 0$, the training problem (1.4) simplifies to

$$\min_{A\in\mathbb{R}, b\in\mathbb{R}} \left\{ E(A,b) = (\tanh(b) - 1)^2 \right\}. \tag{4.13}$$

Since $\tanh(\mathbb{R}) = (-1, 1)$, there are two possible problems here. First, (4.13) does not have a solution, so the task of training does not really make sense. Second, even if we ignore the first problem and just apply a descent algorithm on (4.13), then we encounter another problem: minimising sequences are not bounded. For example, let $A^j := 0$, $b^j := j$, then $\lim_{j\to\infty} E(A^j, b^j) = 0$ but $(A^j, b^j)_{j\in\mathbb{N}}$ is unbounded and does not even contain a convergent subsequence. Thus, we cannot expect our training algorithm to converge.

The key problem in the previous example was that the range of the neural network $\Psi(x_n, \cdot)$ was not closed. The non-closedness of the range is a characterisation of ill-posedness for linear inverse problems in infinite dimensions, see, e.g. [30, Theorem 3.7]. While this may never be the case for finite-dimensional linear inverse problems, the non-linearity in (4.13) results in exactly this property.

In order to overcome this problem, we can either pose constraints on the data fidelity (network architecture, link function, etc.) or we cure the ill-posedness by regularisation as is classically done when considering ill-posed inverse problems [50, 73]. To overcome the problem in (4.13) it is sufficient to add a regulariser R which is *coercive*, meaning that for any sequence of parameters $(\theta^j)_{j\in\mathbb{N}}$ it holds that

$$\lim_{j\to\infty} \|\theta^j\| = \infty \quad \Rightarrow \quad \lim_{j\to\infty} R(\theta^j) = \infty. \tag{4.14}$$

This condition implies that minimising sequences, which are sequences $(\theta^j)_{j\in\mathbb{N}}$ such that $\lim_{j\to\infty} E(\theta^j) = \inf_{\theta\in\Theta} E(\theta)$, are bounded. In reflexive Banach spaces, this is sufficient to guarantee at least a convergent subsequence. For more information on regularisation of non-linear ill-posed inverse problems, we refer to classical textbooks [50, 73].

In the remainder of this section, we discuss a couple of specific choices for the regulariser R. In finite dimensions, due to the equivalence of all norms, any norm is coercive. In addition, the regulariser may impose additional properties on the estimated parameters. By now, it is standard to use the 1-norm $\|\theta\|_1 = \sum_i |\theta_i|$ or the squared 2-norm $\|\theta\|_2^2 = \sum_i |\theta_i|^2$ for regularisation in deep learning to promote solutions, which are sparse or have small coefficients, respectively, [103, 111]. The interpretation of a deep neural network as a process that changes with time naturally calls for other norms. For instance, the next section relies on the squared H^1-norm as a regularisation. For $\theta : [0, T] \to \mathbb{R}^M$ it is defined as

$$\|\theta\|_{H^1}^2 = \|\theta(0)\|^2 + \int_0^T \|\partial_t \theta(t)\|^2 \, \mathrm{d}t. \tag{4.15}$$

This regularisation and its discrete counterpart will promote solutions which vary smoothly across the layers [60, 122].

Finally, the connection of deep neural networks to discretised ODEs motivate other non-standard regularisers, too. To this end we consider ResNet with time-varying discretisation

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

$$z^{k+1} = z^k + h^k \sigma(A^k z^k + b^k) \tag{4.16}$$

and extend the parameters to $\theta = (A^k, b^k, h^k)_{k=0}^{K-1}$. Then it is natural to ensure that the time steps $h := (h^k)_{k=0}^{K-1}$ are non-negative and sum to T. More precisely, we regularise the time steps with $R\colon \mathbb{R}^K \to \mathbb{R}_\infty$,

$$R(h) = \begin{cases} 0, & \text{if} \quad h^k \geqslant 0 \quad k = 0, \dots, K-1, \quad \sum_{k=0}^{K-1} h^k = T, \\ \infty, & \text{otherwise.} \end{cases} \tag{4.17}$$

Note that R is non-smooth and convex and since its proximal operator can be computed efficiently [43], proximal algorithms can be efficiently employed. This regulariser has been used for deep learning in [12]. For residual neural networks, it has also been proposed to penalise the 'weighted path norm', which takes into account the 'effective' depth of a network [49].

It should be noted that the 'problem' of non-existent solutions is usually ignored in practice since most optimisation algorithms do not converge (or at least not in reasonable time) and even if they converge, they may not converge to the global solution. For some neural networks, one can prove that even if the iterates do not converge, an associated quantity called 'normalized margin' still converges [94].

4.3 Deep limits

Let $\theta^{(K)}$ denote a minimiser of (1.4) with a ResNet (1.5) with K layers and by θ^∞ a minimiser of (1.4) with the ODE constraint (1.7). In what sense and under which conditions do discrete solutions $\theta^{(K)}$ converge when the number of layers K tends to infinity? If these converge, do they converge to a solution of the optimal control problem θ^∞?

In order to answer these questions one needs a topology in which we can compare $\theta^{(K)} \in (\mathbb{R}^{M^2+M})^K$ and $\theta^\infty : [0, T] \to \mathbb{R}^{M^2+M}$. With the discrete measure $\mu^{(K)} = \frac{1}{K} \sum_{k=0}^{K-1} \delta_{kT/K}$ we make the identification $(\mathbb{R}^{M^2+M})^K \cong L^2(\mu^{(K)}, [0, T], \mathbb{R}^{M^2+M}) =: L^2(\mu^{(K)})$ where by abusing notation, we associate the discrete object $\theta^{(K)}$ with the piecewise constant function $\theta^{(K)} : [0, T] \to \mathbb{R}^{M^2+M}$ with $\theta^{(K)}(t) = \theta_k^{(K)}$ if $t \in [t^k, t^{k+1})$. It turns out that $H^1 := H^1([0, T], \mathbb{R}^{M^2+M})$ is a suitable solution space for the optimal control problem, so that $L^2 := L^2([0, T], \mathbb{R}^{M^2+M})$ is a natural candidate for the convergence of $\theta^{(K)}$ to θ^∞ since both $L^2(\mu^{(K)})$ and H^1 can be embedded into L^2.

The following theorem is a special case of Theorem 2.1 in [122] which answers the question of deep limits for ResNet (1.6). Its proof relies on the equivalence of convergence in L^2 and a certain transport metric [55], see [122] for more details.

Theorem 4.4 *Let* $E^{(K)} : L^2(\mu^{(K)}) \to \mathbb{R}$

$$E^{(K)}(\theta) = \frac{1}{N} \sum_{n=1}^{N} L_n(\Psi(x_n, \theta)) + \lambda \left(\|\theta(0)\|^2 + K \sum_{k=0}^{K-1} \|\theta(t^{k+1}) - \theta(t^k)\|^2 \right) \tag{4.18}$$

with Ψ *being the discrete ResNet* (1.6) *and* $E^\infty : H^1 \to \mathbb{R}$

$$E^\infty(\theta) = \frac{1}{N} \sum_{n=1}^{N} L_n(\Psi(x_n, \theta)) + \lambda \left(\|\theta(0)\|^2 + \int_0^T \|\partial_t \theta(t)\|^2 \, \mathrm{d}t \right) \tag{4.19}$$

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

with Ψ given by the ODE (1.7). Let σ be Lipschitz continuous with $\sigma(0) = 0$ and L_n be continuous and non-negative for all $n = 1, \dots, N$. If $\lambda > 0$, then

(1) *minimisers of $E^{(K)}$ and E^{∞} exist for all $K \in \mathbb{N}$,*
(2) *minimal values converge, i.e. $\lim_{K\to\infty} \min_{L^2(\mu^{(K)})} E^{(K)} = \min_{H^1} E^{\infty}$ and*
(3) *any sequence of minimisers of $E^{(K)}$, $\{\theta^{(K)}\}_{K\in\mathbb{N}} \subset L^2$, is relatively compact, and any limit point of $\{\theta^{(K)}\}_{K\in\mathbb{N}}$ is a minimiser of E^{∞}.*

4.4 Open problems

Connecting deep learning to optimal control has opened up new avenues to advance the field of deep learning. In this section we discussed algorithms motivated by this connection which are based on derivatives and necessary optimality conditions. We discussed the need and potential for variational regularisation of deep learning and understanding the behaviour of deep neural networks as we increase the number of layers. All of these routes have natural extensions, which will pave the way for better understanding of deep learning and even more powerful tools.

4.4.1 *Algorithms with built-in errors*

Using ODEs as a network architecture and computing gradients via the adjoint, i.e. Theorem 4.1, is theoretically appealing. However, in practice, both the forward and the adjoint ODE have to be solved numerically which induces errors into the gradient. When using off-the-shelf first-order methods like SGD, it is assumed that the gradients are computed exactly, which may either hinder performance or require prohibitively accurate computations, see, for instance, the discussion in [56]. That being said, state-of-the-art algorithms like SGD and Adam are stochastic and the updates are not guaranteed to decrease the objective so the impact of discretisation errors is not clear. Since the numerical solution of the ODE can be computed to any given tolerance, this naturally poses the question how to use such knowledge and control over the accuracy in optimisation algorithms. Some algorithms have been extended to include such errors, see for instance [35, Chapter 8] and [133].

4.4.2 *Algorithms without gradients*

The method of successive approximations and its extended version have been proposed for deep learning [88, 89], but potentially more development is needed to fully exploit this direction including stochastic updates with respect to the data points and efficient maximisation of the Hamiltonian.

The method of successive approximations exploits the structure of deep learning only to the point of optimal control (4.2) but is generic in terms of the architecture, e.g. the choice of f. For discretised system, a more tailored algorithm has been proposed in [119] but without any theoretical guarantees on its convergence.

4.4.3 *Architectures, rates and topologies for deep limits*

The question if the ResNet converges with increasing number of layers was satisfactorily answered in [121], but these results still leave a number of questions unanswered. First, do other architectures also have deep limits? The most likely candidates here are discretisations of ODEs.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

Second, are these results tightly linked to H^1-regularisaton or can they be extended to other topologies, e.g. the one induced by the total variation? Third, are there convergence rates for these limits?

Another work on the convergence of discretised ODEs with finer discretisations is [61]. In particular this work not only proves convergence but also convergence rates. It utilises assumptions on smoothness and coercivity of certain Hessians but it is not clear if these are met in a deep learning context. It would be interesting to verify or falsify the assumptions for relevant learning problems. In case these are not met, a different architecture or regularisation might be a way out.

5 Equivariant neural networks

In recent years, the use of convolutional architectures in deep neural networks [84] for imaging tasks in machine learning has proven to be an extremely fruitful idea. A particularly well-known example of this is the work of Krizhevksy et al. [82], where deep convolutional neural networks (CNNs) were used to achieve state-of-the-art performance in the ImageNet contest (a challenging image classification task), outperforming other approaches by a large margin. Another striking example of the power of CNNs is given by Ulyanov et al. [124], where it is shown that even an untrained CNN can be used as an effective prior for natural images. It is commonly understood that the effectiveness of CNNs in imaging tasks is in large part due to them being in some sense right for the problems at hand. CNNs combine the flexibility of neural networks (in the form of many learnable filters) with the known symmetries of images: both convolutions and pointwise non-linearities commute with translations of the underlying domain, so that the outputs of a CNN transform in a predictable way when their inputs are translated. By constraining the search space in a principled way (in theory, fully connected networks are at least as expressive as CNNs), CNNs can make efficient use of training data to learn to perform tasks to a higher standard than fully connected networks and it is easier to interpret the action of a CNN on its inputs than it is do the same for a fully connected network.

Given the success of CNNs in machine learning, it is natural to ask whether the concept of convolutional architectures can be generalised to incorporate other symmetries into neural network architectures. One current line of research in this direction is the study of group equivariant neural networks, which has been gaining considerable traction since the work on group convolutional neural networks by Cohen and Welling [33]. A neural network can be thought of as a function taking inputs from a feature space $\mathcal{X}^1$ and returning outputs from a feature space $\mathcal{X}^2$. We will call a function $F : \mathcal{X}^1 \to X^2$ G-equivariant if there are group actions of G on $\mathcal{X}^1$ and $\mathcal{X}^2$ (denoted by $T^{\mathcal{X}^1}$ and $T^{\mathcal{X}^2}$ respectively to emphasise that the group actions need not be of the same nature) such that

$$F\left(T_g^{\mathcal{X}^1} x\right) = T_g^{\mathcal{X}^2}[F(x)] \quad \text{for all } x \in \mathcal{X}^1, g \in G. \tag{5.1}$$

To elucidate this definition, let us note some examples of behaviour covered by it:

- if G acts trivially on $\mathcal{X}^2$, i.e. $T_g^{\mathcal{X}^2} = \mathrm{id}$, we recover invariance of F to group transformations of its input, which is often a desirable property of an image classifier;
- whereas if $\mathcal{X}^1 = \mathcal{X}^2$ and $T^{\mathcal{X}^1} = T^{\mathcal{X}^2}$, the output of F transforms in exactly the same way as the input does, which is a useful property to have in many image-to-image tasks such as segmentation.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

If we are given two G-equivariant functions $F_1 \colon \mathcal{X}^1 \to \mathcal{X}^2$, $F_2 \colon \mathcal{X}^2 \to \mathcal{X}^3$ (with the same action of G on $\mathcal{X}^2$ for both functions), their composition is easily seen to be G-equivariant:

$$F_2(F_1(T_g^{\mathcal{X}^1} x)) = F_2(T_g^{\mathcal{X}^2}[F_1(x)]) = T_g^{\mathcal{X}^3}[F_2(F_1(x))].$$

Appealing to this result and noting the usual structure of a neural network as a composition of an alternating sequence of affine maps and non-linearities, there is a promising way of designing G-equivariant neural networks: design G-equivariant linear maps and G-equivariant non-linearities, add biases as appropriate to get affine maps from the linear maps and compose the affine maps and non-linearities as you would in an ordinary neural network.

As it turns out, when the group actions considered above are in fact group representations (i.e. they act linearly), the problem of finding and characterising G-equivariant linear maps reduces to the well-studied problem of finding and characterising intertwiners in representation theory. This insight has recently been used to unify existing approaches to G-equivariant neural networks and to show that G-equivariant linear maps necessarily take the form of a type of group convolution [31]. Let us describe this work in some more detail here.

5.1 Equivariant transformations of feature maps on homogeneous spaces

5.1.1 *Homogeneous spaces*

It has been observed [31, 81] that there is a common setting unifying many of the existing approaches to G-equivariant neural networks. We are given a topological group G, which acts continuously and transitively on a domain X (X is a so-called homogeneous space of G). With this assumption we can cover the cases where $X = \mathbb{R}^d$ and $G = \mathrm{SE}(d) := \mathbb{R}^d \rtimes \mathrm{SO}(d)$, the group of rotations and translations, and where $X = S^{d-1}$ and $G = \mathrm{SO}(d)$, which are two commonly studied cases. Fixing a point $p \in X$ as the origin and denoting by $G_p = \{g \in G \mid gp = p\}$ the stabiliser of p, we can identify X with the quotient space G/G_p: since G acts transitively on X, for any $q \in X$ there is a $g \in G$ such that $gp = q$. On the other hand if $g_1 p = g_2 p$, then $g_2^{-1} g_1 p = p$, so that $g_2^{-1} g_1 \in G_p$, or $g_2 G_p = g_1 G_p$. Hence, there is a well-defined bijective map $X \to G/G_p$ mapping q to gG_p, where $g \in G$ is such that $gq = p$. Conversely, given a closed subgroup $H < G$, G acts transitively on the quotient space G/H by left multiplication, making it into a homogeneous space of G. From this reasoning, we see that the homogeneous spaces of G can be identified precisely with the quotient spaces G/H as H ranges over closed subgroups of G. Henceforth, we will consider two arbitrary subgroups $H_1, H_2 < G$ and the associated homogeneous spaces $G/H_1, G/H_2$.

5.1.2 *Equivariant linear maps*

Scalar-valued features on these spaces can be modelled as functions $G/H_i \to \mathbb{R}$ and these can be arbitrarily stacked to get feature maps $x \colon G/H_i \to \mathbb{R}^C$ that transform under the action of G according to $[\pi_1(g)x](p) = x(g^{-1}p)$, but in this setting one can also consider more general features: given a representation (V_i, ρ_i) of H_i, we can consider signals to be fields of V_i-valued features on G/H_i, which are strictly more general than the stacks of scalar-valued fields since their components are mixed under the action of G. This is mathematically formalised by noting that G is a principal H_i-bundle, constructing from this the associated vector bundle P_i with fibre space V_i, the sections of which, $\Gamma(P_i)$, are the signals of interest. Under the

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

action of the group G, these signals naturally transform according to the representation of G induced by H, $\pi_{H_i}^G$. To put this in the notation of (5.1), we are taking $\mathcal{X}^1 = \Gamma(P_1)$, $\mathcal{X}^2 = \Gamma(P_2)$ and $T_g^{\mathcal{X}^1} = \pi_{H_1}^G(g)$, $T_g^{\mathcal{X}^2} = \pi_{H_2}^G(g)$ and we are asking what the general form is of a linear map $F\colon \mathcal{X}^1 \to \mathcal{X}^2$ that satisfies (5.1) in this case. There are multiple ways in which $\Gamma(P_i)$ and $\pi_{H_i}^G$ can be modelled, but for this purpose it is easiest to consider Mackey functions: we identify $x \in \Gamma(P_i)$ with $x\colon G \to V_i$ satisfying $x(gh) = \rho_i(h^{-1})x(g)$ for all $h \in H_i$, in which case the induced representation is just given by $[\pi_{H_i}^G(g)x](g') = x(g^{-1}g')$. With this notation in place, the following result on equivariant linear maps can be proved (see also Theorem 6.1 in [31]):

Theorem 5.1 *Assume that G is a locally compact unimodular group and that $H_1, H_2 < G$ are closed subgroups. Suppose that $F : \mathcal{X}^1 \to \mathcal{X}^2$ is an integral operator (with respect to the Haar measure on G):*

$$F(x)(g) = \int_G K(g, g')x(g') \, \mathrm{d}g.$$

Then F is equivariant, in the sense that

$$F(\pi_{H_1}^G(g)x) = \pi_{H_2}^G(g)[F(x)] \quad \forall x \in \mathcal{X}^1, g \in G,$$

if and only if F is given by convolution with a kernel $k : G \to \mathrm{Hom}(V_1, V_2)$

$$F(x)(g) = \int_G k(g^{-1}g')x(g') \, \mathrm{d}g'$$

satisfying the additional constraint

$$k(h_2gh_1) = \rho_2(h_2)k(g)\rho_1(h_1) \quad \forall h_1 \in H_1, g \in G, h_2 \in H_2.$$

Derivation. As alluded to in the statement of the theorem, given the assumptions on G, there is a unique (up to multiplication by a scalar constant) Haar measure on G that is both left invariant and right invariant. Writing F as integration against a kernel $K\colon G \times G \to \mathrm{Hom}(V_1, V_2)$, the equivariance condition tells us that

$$\begin{aligned}\int_G K(g'^{-1}g, g'')x(g'') \, \mathrm{d}g'' &= [\pi_{H_2}^G(g')F(x)](g) \\ &= F([\pi_{H_1}^G(g')x])(g) \\ &= \int_G K(g, g'')x(g'^{-1}g'') \, \mathrm{d}g''.\end{aligned}$$

The final expression on the right-hand side can be rewritten using left invariance of the measure to give

$$\int_G (K(g'^{-1}g, g'') - K(g, g'g''))x(g'') \, \mathrm{d}g = 0 \quad \forall x \in \Gamma(P_1),$$

which is the case if and only if

$$K(g'^{-1}g, g'') = K(g, g'g'') \quad \forall g, g', g'' \in G.$$

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

This final condition tells us that $K(g,g') = K(e,g^{-1}g') =: k(g^{-1}g')$, so we find that F is equivariant and only if it is given by a convolution-type operation:

$$F(x)(g) = \int_G k(g^{-1}g')x(g')\,\mathrm{d}g'. \tag{5.2}$$

Since we have assumed that $F(x) \in \Gamma(P_2)$ is a Mackey function, we must have

$$\begin{aligned}\int_G \rho_2(h_2^{-1})k(g^{-1}g')x(g')\,\mathrm{d}g' = \rho_2(h_2^{-1})F(x)(g) &= F(x)(gh_2)\\ &= \int_G k((gh_2)^{-1}g')x(g')\,\mathrm{d}g'\\ &= \int_G k(h_2^{-1}g^{-1}g')x(g')\,\mathrm{d}g'.\end{aligned}$$

Hence $\rho_2(h_2)k(g) = k(h_2 g)$ for all $h_2 \in H_2, g \in G$. On the other hand, $x \in \Gamma(P_1)$ is also a Mackey function, so for any $h_1 \in H_1$

$$\begin{aligned}\int_G k(g^{-1}g')x(g')\,\mathrm{d}g' &= \int_G k(g^{-1}g')\rho_1(h_1)\rho_1(h_1^{-1})x(g')\,\mathrm{d}g'\\ &= \int_G k(g^{-1}g')\rho_1(h_1)x(g'h_1)\,\mathrm{d}g'\\ &= \int_G k(g^{-1}g'h_1^{-1})\rho_1(h_1)x(g')\,\mathrm{d}g'.\end{aligned}$$

Here we have used right invariance of the measure. This implies that $k(gh_1) = k(g)\rho_1(h_1)$ for $h_1 \in H_1, g \in G$. Taking together the above results, the condition for equivariance of an integral operator $F\colon \Gamma(P_1) \to \Gamma(P_2)$ is that we perform a convolution-type operation against a kernel k satisfying the linear constraints

$$k(h_2 g h_1) = \rho_2(h_2)k(g)\rho_1(h_1) \quad \forall h_1 \in H_1, g \in G, h_2 \in H_2. \qquad \square$$

Note 5.2 The assumptions on the group G are immediately satisfied for compact groups such as SO(3), which is the case that is considered in rotationally equivariant neural networks for spherical image data [32, 51, 80] (if $G = \mathrm{SO}(3)$ and $H_i = \mathrm{SO}(2)$, then $S^2 = G/H_i$). It is not necessary for G to be compact though; much of the work on equivariant neural networks has been focused on the case where G is a group of rigid motions of a Euclidean space [33, 34, 128, 129, 132]. In that case $G = \mathrm{T} \rtimes \mathrm{R}$ where $\mathrm{T} = \mathbb{R}^d$ or $\mathrm{T} = \mathbb{Z}^d$ and R is a group of rotations and reflections (such as $\mathrm{SO}(d)$, $\mathrm{O}(d)$ or finite subgroups thereof). With any of these choices G is a locally compact unimodular group.

The constraints on the convolution kernels stated in Theorem 5.1 can be solved once the type of features V_i have been chosen (so before training time) to find a basis for the convolution kernels that give rise to equivariant linear maps, and at training time we can learn equivariant linear maps by learning the parameters of an expansion in this basis.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

5.1.3 *Equivariant non-linearities*

Having established the general form for a large class of equivariant linear maps, the question remains how to choose equivariant non-linearities. If we are to propose a non-linearity $F\colon \Gamma(P_1) \to \Gamma(P_1)$, we cannot in general just apply a pointwise non-linearity as in ordinary neural networks: this will only work if the chosen representation of H_1 does not mix components, as is the case for instance if the representation of H_1 is trivial (which is always the case if $H_1 = \{e\}$) or if the regular representation of H_1 is chosen [129]. One way to make this work, in general, is by having the first layer of the network be a linear layer mapping feature maps defined on the base domain $X = G/H_1$ to feature maps defined on the group $G = G/\{e\}$ and letting all feature maps after that be defined on the group [33, 11]. If the chosen representation does not work well with pointwise non-linearities, another thing that can be done is to take the pointwise norm of the feature map (which is a scalar-valued feature map), pass it through a pointwise non-linearity, and multiply this by the feature map: $F(x)(p) = f(\|x(p)\|)x(p)$, as is done for instance in [132, 121]. Note that f could include a learnable parameter such as a bias parameter. Another option is to combine features of different types in a tensor product [80], which is particularly convenient when working in Fourier space: in Fourier space the convolution operation becomes a pointwise multiplication, but it is not immediately obvious how to apply equivariant non-linearities, so other methods performing the convolution in Fourier space have to transform back to 'real' space (which is computationally expensive) before applying the non-linearity [32, 51].

5.2 A numerical demonstration of the use of equivariant neural networks

In this section, let us consider an example that demonstrates some of the advantages that can be gained by using equivariant neural networks. We will use a supervised learning approach to learn a denoiser: we assume that we are given pairs of random variables representing clean and noisy images (x^*, x) and attempt to solve the following optimisation problem:

$$\min_{\Psi \in C} \frac{1}{2}\mathbf{E}\Big[\|\Psi(x) - x^*\|_2^2 + \lambda\|\nabla(\Psi(x) - x^*)\|_{1,\varepsilon}\Big]. \tag{5.3}$$

Here, C is a class of functions, $\lambda \geqslant 0$ is a small constant, ∇ is a finite difference image gradient operator and $\|\cdot\|_{1,\varepsilon}$ is a smoothed version of the 1-norm. In this specific experiment, we let the clean images x^* be of size 60×60, containing random rectangles with sides aligned to the grid and with random colours and we let the noisy images be generated from clean images by adding Gaussian white noise (as shown in the top right row of Figure 6). It is natural to ask for the denoiser to commute with rotations and translations, that is, for the denoiser to be equivariant with respect to rotations and translations. We compare two choices of C in Problem (5.3), which we will refer to as the class of ordinary and equivariant denoisers respectively. In both cases, the functions in C the form of a ResNet (as defined in (1.5)) preceded by a learnable lifting layer and succeeded by a learnable projection layer. The distinction between the ordinary and equivariant denoisers, is that the ordinary denoiser uses ordinary convolution operations and a pointwise ReLU non-linearity (resulting in translation equivariance), whereas the equivariant denoiser uses the rotation and translation-equivariant versions of these operations as defined in [132]. To give a fair comparison, we fix the same width and depth in both classes. In this case, the equivariant denoiser has a number of degrees of freedom that is less than 10% of the number of degrees of freedom of the ordinary denoiser (62,994 vs. 741,891). The denoisers are

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

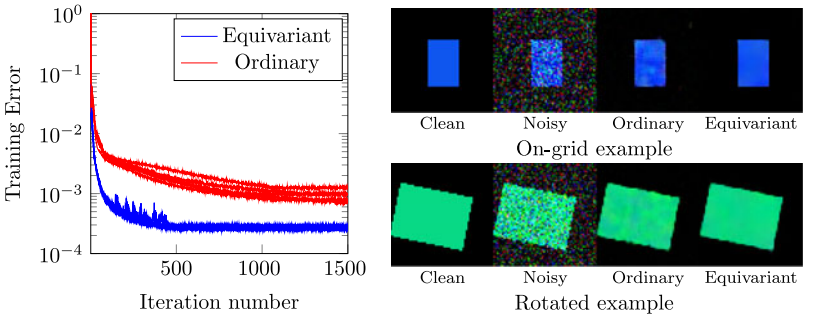

FIGURE 6. A demonstration of the use of equivariance in a denoising task. On the left, we have plotted training errors for five runs of each denoiser. On the right, we have displayed the outputs of the denoisers on an example (the 'on-grid' example) similar to the ones used for training and on a rotated version.

trained by performing 1500 iterations of Adam [77] on minibatches of size 64, decreasing the step size when validation error stagnates, and for each denoiser, we perform 5 training runs. The results are shown in Figure 6: as we see in the plot of the training errors, the equivariant denoiser consistently converges in fewer iterations than the ordinary denoiser and achieves a better objective function value. Besides this, we can be confident that the equivariant denoiser will generalise to rotated examples despite not having seen them in training, whereas it is hard to say anything about how the ordinary denoiser generalises to rotated examples.

5.3 Open problems

5.3.1 *Sample complexity bounds*

One of the main motivations that is given for the use of equivariant neural networks is that they should be able to use training data more efficiently than neural networks that are not designed to be equivariant. This is particularly important in applications in which training data are in short supply, as is often the case in inverse problems [6]. In the machine learning community, the number of samples needed to estimate a given object is known as the sample complexity. There is recent work [42] establishing sample complexity bounds for some simple CNN models, and showing that these bounds compare favourably to the corresponding ones for fully connected networks. In a similar vein, it would be interesting to establish rigorous sample complexity bounds for equivariant neural network models that guarantee their data efficiency.

5.3.2 *Approximation properties*

When applying existing group equivariant neural network architectures as in the framework described above, there are a large number of choices that need to be made: the domains on which the feature maps are defined, the choices of the types of features (the representation of H that is used), the choice of non-linearity. While there is a vast amount of literature on the approximation properties for ordinary neural networks (including older works such as [37, 69] and some more recent works that apply to CNNs [15, 108]), there is not yet the same theoretical guidance on how the choices of the various aspects of group equivariant neural networks can be made to guarantee that the networks are sufficiently expressive. There is some theoretical work

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

on hypothetical equivariant neural networks and approximation results relating to them [136], but it is not yet of practical use in choosing an equivariant architecture.

5.3.3 *Approximate equivariance*

Many of the symmetries we would like to work with in equivariant neural networks are continuous, but when we implement them in practice it is necessary to discretise them. Generally in the literature, one of two approaches is taken: the group is discretised and exact group convolutions are taken with respect to the discrete subgroup (in which case we have exact equivariance to the discrete symmetry), or the group convolutions are derived in the continuous setting and eventually discretised (so that we have approximate equivariance to the continuous symmetry). In either case, it has not been studied in detail how much of an error we make when we make these discretisations and whether there is an optimal discretisation to use. It would be of great interest to provide bounds on the equivariance error, as these could be used to decide, for example, whether a theoretically invariant classifier is actually invariant in practice.

6 Structure-exploiting learning

The training of neural networks amounts to the numerical optimisation of typically smooth, but high-dimensional and highly non-linear objectives as in (1.4), and as discussed in the optimal control framework in Section 4. The most widely used numerical method for (1.4) is Adam [77].

As before let $\Psi : \mathcal{X} \times \Theta \to \mathcal{X}$ be a (deep) neural network that depends on the data and the parameters $\theta \in \Theta$. The training of Ψ amounts to the optimisation over the parameters θ with respect to a loss function

$$\min_{\theta\in\Theta} \left\{ E(\theta) = \frac{1}{N}\sum_{n=1}^{N} L(\eta \circ \Psi(x_n, \theta), y_n) + R(\theta) \right\},$$

as in (1.4). While the loss function L usually is a convex function, the dependence of the parameter θ on the network Ψ is in general highly non-linear which makes the overall optimisation problem in (1.4) non-convex. For what follows, let us denote by $L^j(\theta) = \frac{1}{|\mathcal{S}^j|}\sum_{n\in\mathcal{S}^j}(L(\eta \circ \Psi(x_n, \theta), y_n) + \frac{1}{N}R(\theta))$, for $j = 0, 1, \ldots$, where $\mathcal{S}^j$ is a randomly chosen set of indices from the N training samples. For fixed positive parameters α and $\beta_1, \beta_2 < 1$, and for appropriate initialisations for the parameters and the auxiliary moment functions m_0 and v_0, Adam amounts to the following iteration:

$$\begin{aligned} m^j &= \frac{1}{(1-\beta_1^j)}\left(\beta_1 m^{j-1} + (1-\beta_1)\nabla L^j(\theta^{j-1})\right), \\ v^j &= \frac{1}{(1-\beta_2^j)}\left(\beta_2 v^{j-1} + (1-\beta_2)\nabla L^j(\theta^{j-1})\cdot\nabla L^j(\theta^{j-1})\right), \\ \theta^j &= \theta^{j-1} - \alpha\frac{m^j}{\sqrt{v^j}+\varepsilon}, \end{aligned} \tag{6.1}$$

for a small $\varepsilon > 0$ and where $\sqrt{v^j}$ is taken component-wise. Stochasticity, in the form of choosing subsets $\mathcal{S}^j$ randomly in every iteration, is crucial to deal with high dimensionality of the problem.

Adam is being used for almost all of neural network training because of its easy implementation, its robustness to rescaling, its computational efficiency and small memory requirements and

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

for its suitability for problems which are large-scale in terms of training data and parameters. On the other hand, its theoretical convergence properties do in general not guarantee convergence to a solution of (1.4), cf. [112] where the authors propose a convergent variant in the convex setting and [137], which provides convergence guarantees in the non-convex and stochastic setting.

The literature for optimising smooth (non-convex) objectives is, however, much richer than Adam alone. It is tightly linked to developments in convex analysis and operations research, as well as the numerical discretisation of dynamical systems, ODEs and PDEs as discussed in parts in Sections 2 and 4. In the context of neural network training other optimisation schemes have also been investigated recently. Here, we will mainly focus on those that have some structure-preserving property such as Hamiltonian descent [95, 105], which are guaranteed to dissipate a Hamiltonian energy, optimisation approaches when the features or network parameters are elements in a Riemannian rather than Euclidean space [1, 87, 123], and – as a special case of the latter – information geometry approaches that describe optimisation on statistical manifolds [4].

6.1 Conformal Hamiltonian systems

The most classic approach for minimising $E(\cdot)$ over $\mathbb{R}^L$ is gradient descent, i.e. to seek a stationary point of E by evolving

$$\dot{\theta} = -\nabla E(\theta). \tag{6.2}$$

Several optimisation methods for E can then be derived through different discretisations of (4.6), with the simplest one being explicit gradient descent and its stochastic versions [114]. Another route for the derivation of optimisers for E is obtained by replacing the gradient flow (6.2) with a Hamiltonian flow as in (1.10) with dissipation, for example, a conformal Hamitonian system, i.e.

$$\begin{aligned} \dot{p} &= -\nabla E(\theta) - \gamma p, \\ \dot{\theta} &= \frac{p}{\mu}, \end{aligned} \tag{6.3}$$

with $\gamma, \mu > 0$. Rewriting the system (6.3) in one equation gives

$$\ddot{\theta} + \gamma \dot{\theta} = -\frac{1}{\mu} \nabla E(\theta),$$

which is gradient descent accelerated by momentum, cf. also [118, 90]. More generally, (6.3) is a special case of a conformal Hamiltonian system of the form

$$\begin{aligned} \dot{p} &= -\nabla_\theta H(p, \theta) - \gamma p, \\ \dot{\theta} &= \nabla_p H(p, \theta), \end{aligned} \tag{6.4}$$

for a *separable* Hamiltonian function $H(\theta, p) = T(p) + E(\theta)$ as in Section 2.1.2. Taking $T(p) = \frac{1}{2\mu}\|p\|_2^2$ we get (6.3). Taking $T(p) = \sqrt{\|p\|^2 + \varepsilon}$ we get a new optimisation approach for E which is called relativistic gradient descent [53]

$$\begin{aligned} \dot{p} &= -\nabla E(\theta) - \gamma p, \\ \dot{\theta} &= \frac{p}{\sqrt{\varepsilon + \|p\|^2}}. \end{aligned} \tag{6.5}$$

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

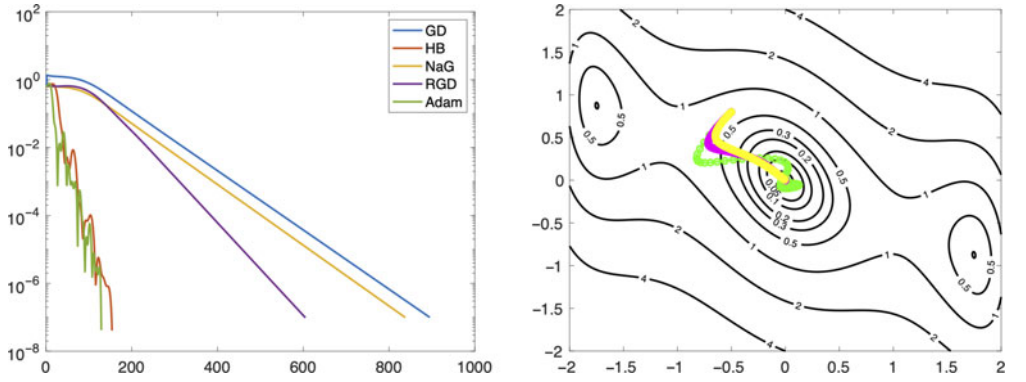

FIGURE 7. Optimisation of $V(x,y) = 2x^2 - 1.05x^4 + 1/6x^6 + xy + y^2$. Comparison of gradient descent, Hamiltonian descent and Adam iterations, initial guess $[-0.5, 0.8]$, global minimum at $[0, 0]$. Methods and parameters: Gradient descent (GD) learning rate $h = 0.01$, Heavy ball (HB) $\mu = 0.9$, $h = 0.01$, Nesterov Accelerated Gradient (NaG) $\mu = 0.012$, $h = 0.01$, Relativistic Gradient Descent (RGD) [53] $h = 0.0001$, $\mu = 0.9259$, Adam $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e-8$, $\alpha = h = 0.1$. Left: value of the loss function versus number of iterations. Right: trajectory of approach to the optimum (NaG $h = 0.01$ yellow, RGD $h = 0.0001$ magenta, Adam $h = 0.1$ green).

While the gradient system (6.2) dissipates E, for a conformal Hamiltonian system (6.4) the Hamiltonian H is dissipated as

$$\frac{\mathrm{d}}{\mathrm{d}t}H(p,\theta) = -\frac{\gamma}{2\mu}p^T p.$$

For separable Hamiltonians with the kinetic energy T chosen so it has a global minimiser in 0, limit points of (6.4) recover stationary points of E. More precisely, for $H = T(p) + E(\theta)$ being separable the equilibria of (6.4) fulfil

$$\gamma\, p = -\nabla E(\theta),$$
$$0 = \nabla T(p).$$

If T is chosen so it has a unique global minimum in $p^* = 0$ (e.g. in the case of (6.3)), we have that $(0, \theta^*)$ is a zero of $\nabla H(p,\theta)$ if and only if u^* is a zero of $\nabla E(\theta)$, i.e. θ^* is a stationary point of $E(\theta)$. Moreover, we have that $(0, \theta^*)$ is the solution of the conformal Hamiltonian system as $t \to \infty$. Then, using a numerical integration method that preserves the property of Hamiltonian energy dissipation (such a numerical method is called conformal symplectic scheme [13]), we get an approximation of $(\theta^*, 0)$, where $(\theta^*, 0)$ is an equilibrium of (6.4) and θ^* a stationary point of E [99].

In recent work, Franą et al. [53] take advantage of the connection between optimisation schemes for E and conformal symplectic Hamiltonian schemes for $H = T + E$ for the design of new optimisation approaches for E – similar to the connections we have seen before, e.g. between Adam and conformal Hamiltonian descent. See also Figure 7 for a comparison between different optimisation methods applied to the camelback function.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

6.2 Learning in Riemannian metric spaces

After identifying the dissipation of an appropriate Hamiltonian as a structure worth preserving for numerical optimisation of neural networks, as discussed in the last section, explicit conditions on the parameter matrices themselves, such as orthogonality, seem to impose structure on the optimisation that can be of advantage and in particular lead to better (at least empirical) convergence rates, better generalisability and accuracy [87]. Such conditions usually pose a parameter optimisation problem within a Riemannian metric space rather than Euclidean space. Moreover, there are several important applications, cf. in particular the next subsection, where the training data and the parameters naturally lie in a Riemannian space.

While in Section 2.3 we discussed the case when the training data and features lie on a Riemannian manifold, in this section, we consider the setting in which the parameters belong to a manifold. This in particular means that ordinary gradient descent a la (6.2), i.e. with the gradient ∇ being the ordinary gradient in $\mathbb{R}^L$, does in general not describe steepest descent of E in such a Riemannian metric space. Instead descent with respect to a metric-induced gradient needs to be considered [5]. In what follows, we consider some representative examples of how Riemannian optimisation can arise in the context of deep learning.

6.2.1 *Network parameters evolving on manifolds*

The parameters to be learned evolve on a manifold when additional structure or constraints are imposed on the parameters. This is done to improve stability of the training algorithm and of the trained model. Pioneering work advocating the use of Riemannian gradient and orthogonality constraints can be found in [2, 3, 17] and there is an extensive follow-up literature. Examples of this procedure in the context of deep learning have recently appeared in [91] and earlier in the context of CNNs in [8, 28]. It is crucial to implement the Riemannian gradient descent efficiently, making good use of the tensor structure of the data and of the layers of the neural network. One way to proceed is to introduce an evolution equation for θ and replace (1.7) by the extended system of ODEs

$$\dot{z} = f(z, \theta), \tag{6.6}$$

$$\dot{\theta} = g(\theta). \tag{6.7}$$

Of particular importance is the case where the second equation evolves on a Lie group G or on a homogeneous manifold $\mathcal{M} = G/H$.[4] In [3] G is simply the general linear group, while orthogonality constraints have been adopted in [71] in the context of independent component analysis. Other concrete examples that we have in mind include the affine group, the special Euclidean group $\mathrm{SE}(n)$, the special orthogonal group $\mathrm{SO}(n)$, the Stiefel manifold $\mathrm{V}_{n,p} = \mathrm{SO}(n)/\mathrm{SO}(n-p)$ and the Grassmann manifold $\mathrm{G}_{n,p} = \mathrm{SO}(n)/(\mathrm{SO}(n-p) \times \mathrm{SO}(p))$ including the n-sphere.

For matrix Lie groups G (and homogeneous manifolds) (6.7) can simply take the form of a linear matrix differential equation

$$\dot{\theta} = M(t)\,\theta, \qquad \theta \in \mathcal{M}$$

[4] Here, H is a closed Lie subgroup of G, and $\mathcal{M} := G/H$ the quotient with the manifold structure turning $\pi : G \to G/H$, $g \mapsto gH$ into a submersion. Then $\mathcal{M}$ becomes a homogeneous space for G with respect to the transitive Lie group action induced by left multiplication.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

and $M(t) \in \mathfrak{g}$, and the optimisation can be performed in terms of the variable M. To preserve the manifold structure one can consider a discretisation of this equation of the form

$$\theta_{k+1} = \exp(h\, M_k)\theta_k,$$

where $\exp : \mathfrak{g} \to G$ is the matrix exponential. This discretisation can be seen as the prototype for constructing local coordinates[5] on the manifold of parameters. Alternatively, any approximation of the exponential map $\phi \approx \exp$ (e.g. by a rational approximant) preserving the property $\phi : \mathfrak{g} \to G$, can be used to construct local parametrisations of the manifold[6]. On homogeneous manifolds such as $\mathrm{V}_{n,p}$ and $\mathrm{G}_{n,p}$ additional structure on $M(t)$ can be exploited to reduce the computational cost of matrix exponentials and similar mappings [20].

After time discretisation, this approach guarantees that the parameters at each layer of the network belong to manifolds which are all naturally equipped with Riemannian metrics and some of which are compact. In particular, a metric on G that is H-right invariant (i.e. the right multiplication R_h with $h \in H$ is an isometry) descends to a Riemannian metric on $\mathcal{M} = G/H$ [54]. Gradient descent techniques to train the network should then exploit the Riemannian structure [2, 91]. A sufficient condition for convergence of Riemannian gradient descent is that the manifolds are geodesically complete [130, 123]. All Riemannian homogeneous manifolds as well as compact Riemannian manifolds are geodesically complete [79, IV.4].

6.2.2 *Information geometry*

A special case of the Riemannian structure discussed in the previous paragraph arises when taking into account the inherent statistical properties of the underlying unknown distributions of training pairs and, connected to this, statistical properties of the network parameters θ.

Treating the parameters θ as probability distributions, they can be modelled as elements on a statistical manifold with an appropriate metric. A statistical manifold is a Riemannian manifold whose points correspond to probability distributions. Gradient descent on statistical manifolds is studied in information geometry. Here, the so-called natural gradient is the proposed notion of gradient on a statistical manifold [2]. For an L-dimensional parameter space, equipped with a Riemannian metric tensor $G = G(\theta) = (g_{ij}(\theta)) \in \mathbb{R}^{L\times L}$ that depends on θ, the natural gradient of $E(\theta)$ is defined as

$$\tilde{\nabla} E(\theta) = G^{-1}(\theta)\nabla E(\theta),$$

where $\nabla E(\theta)$ denotes the ordinary gradient of E in $\mathbb{R}^L$. The natural metric considered in this context is the Fisher information, with G being the Fisher information matrix of the parameters θ. Natural gradient descent then reads

$$\dot{\theta} = -G^{-1}(\theta)\nabla E(\theta). \tag{6.8}$$

[5] Otherwise called as retraction maps [1].

[6] An example for the case of $\mathrm{SO}(n)$ and $\mathrm{Sp}(n)$ is the Cayley transformation, which is used in the context of invertible networks in Section 3.1.3.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

In the case of $E(\theta) = \frac{1}{2N}\sum_{n=1}^{N} \|\Psi(x_n,\theta) - y_n\|^2$ a squared error loss and G being the Fisher information matrix, we have

$$G(\theta) = \frac{1}{N}\sum_{n=1}^{N} J_n^T J_n,$$

where J_n is the Jacobian of $\Psi(x_n,\theta)$ with respect to θ, cf. [96]. Note that in this case natural gradient descent, discretised with forward Euler, is equivalent to Gauss–Newton iteration [104]. This connection between natural gradient descent and (extended) Gauss–Newton methods can be extended to more general losses as well [107]. In the continuum limit, (6.8) is a gradient flow with respect to the Fisher-Rao metric, see [102, Definition 3.1].

Several works [2, 4, 107] have demonstrated advantages of using the natural gradient over the ordinary gradient in (6.2) for neural network training. The Riemannian structure seems to guard against the gradient descent being trapped in flat areas of the loss function's surface [134], and as a result, the network to feature better generalisation capabilities, cf. [66, 76] for different characterisations of flatness of the loss function's surface.

6.3 Optimisation of two-layer ReLU neural networks as Wasserstein gradient flows

Continuing our discussion on the mathematical structure of training deep neural networks, we now consider using such structure to analyse optimality and generalisation properties of trained networks. More precisely, the inherent gradient flow structure of training neural networks also appears when studying global optimality and generalisation properties of trained networks. Bach and Chizat [26] pick up the gradient flow formulation of neural network learning and study convergence of the learning problem (1.4) to a global minimiser of E for two-layer ReLU neural networks Φ in the ∞-width limit. In particular, their analysis makes use of the structure of a Wasserstein gradient flow formulation of (6.2) over the space of probability measures in the ∞-width network limit. Proving convergence to a global minimiser of E also allows the study of optimal generalisation capabilities of the trained two-layer ReLU neural network by characterising the limit (for a certain class of loss functions with exponential tails) as a max-margin classifier [27].

In [26] they consider a two-layer neural network of the form

$$\Phi(\theta, x) = \frac{1}{J}\sum_{j=1}^{J} f(\theta_j, x), \tag{6.9}$$

where

$$f(\theta_j, x) = c_j \ \max\{a_j^t x + b_j, 0\},$$

for $x \in \mathbb{R}^M$ and $\theta_j = (a_j, b_j, c_j) \in \mathbb{R}^{M+2}$. The weights θ in (6.9) are learned by minimising a loss function such as (1.4) with optional regularisation $R(\theta) = \frac{\lambda}{L}\sum_{j=1}^{L} \|\theta_j\|_2^2$. In this setting, they investigate the performance (generalisation capabilities) of the learned network Φ by studying the associated gradient flow of the loss function E for initial weights $\theta(0) = \theta_0 \ \sim$ i.i.d. $\mu_0 \in P_2(\mathbb{R}^{n+1})$

$$\dot{\theta} = -m\nabla E(\theta, x). \tag{6.10}$$

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

In particular, they analyse convergence of (6.10) in the ∞-width limit, which they prove can be written as a Wasserstein gradient flow. More precisely, they parametrise the network Φ with probability measures $\mu \in P_2(\mathbb{R}^{d+2})$ as

$$\Phi(\mu, x) = \int \Phi(\theta, x)\, \mathrm{d}\mu(\theta),$$

and the associated loss function as

$$E(\mu) = \frac{1}{N} \sum_{n=1}^{N} L(\Phi(\mu, x_n), y_n) + \lambda \int \|\theta\|_2^2 \, \mathrm{d}\mu(\theta). \tag{6.11}$$

In this setting they prove the following results.

Theorem 6.1 ([26]) *Assume that*

$$spt(\mu_0) \subset \{|c|^2 = \|a\|_2^2 + |b|^2\}.$$

As $L \to \infty$, $\mu_{t,L} = \frac{1}{L} \sum_{j=1}^{L} \delta_{\theta_j}(t)$ *converges in* $P_2(\mathbb{R}^{d+1})$ *to* μ_t, *the unique Wasserstein gradient flow of* E *in* (6.11) *starting in* μ_0.

Moreover, assuming μ_0 *is 'diverse' enough (cf.* [26] *for details). If* μ_t *converges to* μ_∞ *in* $P_2(\mathbb{R}^{d+1})$, *then* μ_∞ *is a global minimiser of* E.

Note that this gradient flow formulation of neural network training also has connections to several interesting works on the mean-field theory of neural networks, see for instance [48].

6.4 Open problems

6.4.1 *Port-Hamiltonian optimisation methods*

Generally, investigating new optimisation methods for E by considering different instances of Hamiltonian descent is a promising research direction. Here, different choices of Hamiltonians or special cases of Hamiltonian systems might be advantageous for classes of loss functions and network architectures, imposing different descent dynamics. For example, interesting schemes arise when symplectic numerical integration is applied to port-Hamiltonian systems (with different Hamiltonians), cf. [125] for an introduction to port-Hamiltonian systems and [21] for the development of structure-preserving numerical integrators for port-Hamiltonian systems by using discrete gradient and splitting approaches. Massaroli et al. [98] design a loss function and parameter optimisation dynamics of the network in such a way that the neural network itself behaves like an autonomous port-Hamiltonian system. This, in turn, allows a proof of convergence of the optimisation algorithm to a minimum of the loss. Taking this a step further, port-Hamiltonian systems also lend themselves to the design of locally adaptive optimisation schemes as the port-Hamiltonian structure is preserved under concatenation of port-Hamiltonians with orthogonal input–output relation.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

6.4.2 *Convergence analysis for natural gradient optimisation*

While several papers seem to suggest (mostly empirically or for linear networks) that natural gradient descent helps to mitigate nuisance curvature in the neural network parameter optimisation, very little seems to have been done on this theoretically, in particular for non-linear neural networks [138]. In general, while for particular choices of the Riemannian metric (6.8) boils down to classical optimisation schemes, such as Gauss–Newton for G being the Fisher information metric, analytic results on convergence properties of natural gradient flow discretisations are generally open. Indeed, the study of (6.8) for different choices of G could be very interesting. For instance, if G is a BFGS approximation to the Hessian of E, then a stochastic method was proposed and analysed by Wang et al. [127].

6.4.3 *Studying generalisation properties of neural networks by metric gradient flows*

As the example of the Wasserstein gradient flow in Section 6.3 has shown, metric gradient flows can serve as a useful tool for studying convergence of the network training and for the study of generalisation properties of the minimisers [26, 27]. It would be interesting to see if other metric gradient flows would also lend themselves to such an analysis. In connection with information geometry in Section 6.2.2 we have seen the gradient flow with respect to the Fisher-Rao metric appearing. Could this be used to study convergence properties for other network architectures, beyond two-layer ReLU? Are there other metrics that could be interesting to investigate for that purpose?

7 Conclusion

Structure-preserving approaches to deep learning are a means to design neural networks with guaranteed mathematical properties, to derive optimisation schemes that improve their training and to analyse their global optimality and generalisation capabilities. In this paper, we discuss some recent examples from this emerging topic of structure-preserving deep learning. These include ODE and PDE parametrisations of neural networks for improved stability properties, an optimal control formulation for neural network training that gives rise to systematic training and regularisation procedures, invertible neural networks for large-scale deep learning, equivariant neural network architectures for the design of neural networks that preserve group transformations and structure-preserving training of neural networks by means of Hamiltonian descent and Riemannian metric gradient flows. Together with the discussion of state-of-the-art results, we also suggest a range of open problems that we identified as interesting mathematical avenues that could help to shed light on the systematic design and training of deep neural networks.

Acknowledgements

MJE would like to thank Matt Thorpe for fruitful discussions. MJE acknowledges support from the EPSRC grants EP/S026045/1 and EP/T026693/1, the Faraday Institution via EP/T007745/1, and the Leverhulme Trust fellowship ECF-2019-478. CE and CBS acknowledge support from the Wellcome Innovator Award RG98755. CBS acknowledges support from the Leverhulme Trust project on ‘breaking the non-convexity barrier, the Philip Leverhulme Prize, the EPSRC

grants EP/S026045/1 and EP/T003553/1, the EPSRC Centre Nr. EP/N014588/1, European Union Horizon 2020 research and innovation programmes under the Marie Skłodowska-Curie grant agreement No. 777826 NoMADS and No. 691070 CHiPS, the Cantab Capital Institute for the Mathematics of Information and the Alan Turing Institute. FS acknowledges support from the Cantab Capital Institute for the Mathematics of Information. EC and BO thank the SPIRIT project (No. 231632) under the Research Council of Norway FRIPRO funding scheme. The authors would like to thank the Isaac Newton Institute for Mathematical Sciences, Cambridge, for support and hospitality during the programmes *Variational methods and effective algorithms for imaging and vision (2017)* and *Geometry, compatibility and structure preservation in computational differential equations (2019)* where work on this paper was undertaken, EPSRC grant EP/K032208/1.

Conflict of interest

None.

References

[1] ABSIL, P.-A., MAHONY, R. & SEPULCHRE, R. (2008) *Optimization Algorithms on Matrix Manifolds*, Princeton University Press, Princeton, NJ. With a foreword by Paul Van Dooren.

[2] AMARI, S.-I. (1998) Natural gradient works efficiently in learning. *Neural Comput.* **10**(2), 251–276.

[3] AMARI, S.-I., CICHOCKI, A. & YANG, H. H. (1996) A new learning algorithm for blind signal separation. In: *Advances in Neural Information Processing Systems*, pp. 757–763.

[4] AMARI, S.-I. & DOUGLAS, S. C. (1998) Why natural gradient? In: *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*, Vol. 2, IEEE, pp. 1213–1216.

[5] AMBROSIO, L., GIGLI, N. & SAVARÉ, G. (2008) *Gradient flows: in Metric Spaces and in the Space of Probability Measures*, Springer Science & Business Media, Berlin.

[6] ARRIDGE, S., MAASS, P., ÖKTEM, O. & SCHÖNLIEB, C.-B. (2019) Solving inverse problems using data-driven models. *Acta Numerica* **28**, 1–174.

[7] ASOREY, M., CARIÑENA, J. F. & IBORT, L. A. (1983) Generalized canonical transformations for time-dependent systems. *J. Math. Phys.* **24**(12), 2745–2750.

[8] BÉCIGNEUL, G. & GANEA, O.-E. (2019) Riemannian adaptive optimization methods. In: *International Conference on Learning Representations*.

[9] BEHRMANN, J., GRATHWOHL, W., CHEN, R. T. Q., DUVENAUD, D. & JACOBSEN, J.-H. (2019) Invertible residual networks. In: K. Chaudhuri and R. Salakhutdinov (editors), *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 97, Long Beach, California, USA, 09–15 June 2019, PMLR, pp. 573–582.

[10] BEHRMANN, J., VICOL, P., WANG, K. C., GROSSE, R. & JACOBSEN, J. H. (2021) Understanding and mitigating exploding inverses in invertible neural networks. In: *International Conference on Artificial Intelligence and Statistics*, PMLR, pp. 1792–1800.

[11] BEKKERS, E. J., LAFARGE, M. W., VETA, M., EPPENHOF, K. A. J., PLUIM, J. P. W. & DUITS, R. (2018) Roto-translation covariant convolutional networks for medical image analysis. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, Cham, pp. 440–448.

[12] BENNING, M., CELLEDONI, E., EHRHARDT, M. J., OWREN, B. & SCHÖNLIEB, C.-B. (2019) Deep learning as optimal control problems: models and numerical methods. *J. Comput. Dyn.* **6**(2), 171–198.

[13] BHATT, A., FLOYD, D. & MOORE, B. E. (2016) Second order conformal symplectic schemes for damped Hamiltonian systems. *J. Sci. Comput.* **66**(3), 1234–1259.

[14] BOGACHEV, V. I. (2007) *Measure Theory*, Vol. 1, Springer Science & Business Media, Berlin.
[15] BÖLCSKEI, H., GROHS, P., KUTYNIOK, G. & PETERSEN, P. (2019) Optimal approximation with sparsely connected deep neural networks. *SIAM J. Math. Data Sci.* **1**(1), 8–45.
[16] BONNANS, J. F. (2019) Course on Optimal Control. http://www.cmap.polytechnique.fr/~bonnans/notes/oc/ocbook.pdf.
[17] CARDOSO, J.-F. & LAHELD, B. H. (1992) Equivariant adaptive source separation. *IEEE Trans. Signal Process.* **44**, 3017–3030.
[18] CARNEGIE MELLON UNIVERSITY GRAPHICS LAB. (2003) Motion capture database. http://mocap.cs.cmu.edu/.
[19] CELLEDONI, E., ESLITZBICHLER, M. & SCHMEDING, A. (2016) Shape analysis on Lie groups with applications in computer animation. *J. Geom. Mech.* **8**(3), 273–304.
[20] CELLEDONI, E. & FIORI, S. (2004) Neural learning by geometric integration of reduced 'rigid-body' equations. *J. Comput. Appl. Math.* **172**(2), 247–269.
[21] CELLEDONI, E. & HØISETH, E. H. (2017) Energy-Preserving and Passivity-Consistent Numerical Discretization of Port-Hamiltonian Systems. arXiv preprint arXiv:1706.08621.
[22] CELLEDONI, E., MARTHINSEN, H. & OWREN, B. (2014) An introduction to Lie group integrators—basics, new developments and applications. *J. Comput. Phys.* **257**(part B), 1040–1061.
[23] CHANG, B., MENG, L., HABER, E., RUTHOTTO, L., BEGERT, D. & HOLTHAM, E. (2018) Reversible architectures for arbitrarily deep residual neural networks. In: *Thirty-Second AAAI Conference on Artificial Intelligence*, Vol. 32, AAAI Press, Palo Alto, pp. 2811–2818.
[24] CHEN, T. Q., BEHRMANN, J., DUVENAUD, D. & JACOBSEN, J.-H. (2019) Residual flows for invertible generative modeling. In: *Advances in Neural Information Processing Systems*, pp. 9913–9923.
[25] CHEN, T. Q., RUBANOVA, Y., BETTENCOURT, J. & DUVENAUD, D. (2018) Neural ordinary differential equations. In: *Advances in Neural Information Processing Systems*, pp. 6572–6583.
[26] CHIZAT, L. & BACH, F. (2018) On the global convergence of gradient descent for over-parameterized models using optimal transport. In: *Advances in Neural Information Processing Systems*, pp. 3036–3046.
[27] CHIZAT, L. & BACH, F. (2020) Implicit Bias of Gradient Descent for Wide Two-Layer Neural Networks Trained with the Logistic Loss. arXiv preprint arXiv:2002.04486.
[28] CHO, M. & LEE, J. (2017) Riemannian approach to batch normalization. In: *Advances in Neural Information Processing Systems*, pp. 5225–5235.
[29] CICCONE, M., GALLIERI, M., MASCI, J., OSENDORFER, C. & GOMEZ, F. (2018) NAIS-Net: stable deep networks from non-autonomous differential equations. In: *Advances in Neural Information Processing Systems*, pp. 3025–3035.
[30] CLASON, C. (2020) Regularization of Inverse Problems. arXiv:2001.00617.
[31] COHEN, T., GEIGER, M. & WEILER, M. (2019) A general theory of equivariant CNNs on homogeneous spaces. In: *Advances in Neural Information Processing Systems 32*, pp. 9145–9156.
[32] COHEN, T. S., GEIGER, M., KOEHLER, J. & WELLING, M. (2018) Spherical CNNs. arXiv:1801.10130.
[33] COHEN, T. S. & WELLING, M. (2016) Group equivariant convolutional networks. In: *International Conference on Machine Learning*, pp. 2990–2999.
[34] COHEN, T. S. & WELLING, M. (2017) Steerable CNNs, 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings.
[35] CONN, A. R., GOULD, N. I. M. & TOINT, P. L. (2000) *Trust-Region Methods*, MPS-SIAM Series on Optimization, Vol. 1. MPS/SIAM, Philadelphia.
[36] COOK, P., BAI, Y., NEDJATI-GILANI, S., SEUNARINE, K., HALL, M., PARKER, G. & ALEXANDER, D. (2006) Camino: open-source diffusion-MRI reconstruction and processing. In: *Proceedings of the 14th Scientific Meeting of ISMRM, Seattle WA, USA*, Vol. 2759.
[37] CYBENKO, G. (1989) Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.* **2**(4), 303–314.
[38] DAHLQUIST, G. (1979) Generalized disks of contractivity for explicit and implicit Runge-Kutta methods. Technical report, CM-P00069451.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

[39] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009) Imagenet: a large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, pp. 248–255.
[40] Dinh, L., Krueger, D. & Bengio, Y. (2014) NICE: Non-Linear Independent Components Estimation. arXiv preprint arXiv:1410.8516.
[41] Dinh, L., Sohl-Dickstein, J. & Bengio, S. (2016) Density Estimation Using Real NVP. arXiv preprint arXiv:1605.08803.
[42] Du, S. S., Wang, Y., Zhai, X., Balakrishnan, S., Salakhutdinov, R. & Singh, A. (2018) How Many Samples are Needed to Estimate a Convolutional or Recurrent Neural Network? arXiv:1805.07883.
[43] Duchi, J., Shalev-Shwartz, S., Singer, Y. & Chandra, T. (2008) Efficient projections onto the l_1-ball for learning in high dimensions. In: *Proceedings of the 25th International Conference on Machine Learning - ICML*, pp. 272–279.
[44] Dupont, E., Doucet, A. & Teh, Y. W. (2019) Augmented neural ODEs. In: *Advances in Neural Information Processing Systems*.
[45] Durkan, C., Bekasov, A., Murray, I. & Papamakarios, G. (2019) Neural spline flows. In: *Advances in Neural Information Processing Systems*, pp. 7509–7520.
[46] E, W. (2017) A proposal on machine learning via dynamical systems. *Commun. Math. Stat.* **5**(1), 1–11.
[47] E, W., Han, J. & Li, Q. (2018) A Mean-Field Optimal Control Formulation of Deep Learning. arXiv:1807.01083v1.
[48] E, W., Han, J. & Li, Q. (2019) A mean-field optimal control formulation of deep learning. *Res. Math. Sci.* **6**(1), 1–41.
[49] E, W., Ma, C. & Wang, Q. (2019) A Priori Estimates of the Population Risk for Residual Networks. arXiv, pp. 1–19.
[50] Engl, H. W., Hanke, M. & Neubauer, A. (1996) *Regularization of Inverse Problems*, Mathematics and Its Applications, Springer, Berlin.
[51] Esteves, C., Allen-Blanchette, C., Makadia, A. & Daniilidis, K. (2018) Learning SO(3) equivariant representations with spherical CNNs. In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 52–68.
[52] Etmann, C., Ke, R. & Schönlieb, C.-B. (2020) iUNets: Fully Invertible U-Nets with Learnable Up-and Downsampling. arXiv preprint arXiv:2005.05220.
[53] França, G., Sulam, J., Robinson, D. P. & Vidal, R. (2019) Conformal Symplectic and Relativistic Optimization. arXiv preprint arXiv:1903.04100.
[54] Gallot, S., Hulin, D. & Lafontaine, J. (2004) *Riemannian Geometry*, 3rd ed., Universitext, Springer-Verlag, Berlin.
[55] García Trillos, N. & Slepčev, D. (2016) Continuum limit of total variation on point clouds. *Arch. Ration. Mech. Anal.* **220**(1), 193–241.
[56] Gholami, A., Keutzer, K. & Biros, G. (2019) ANODE: unconditionally accurate memory-efficient gradients for neural ODEs. In: *IJCAI International Joint Conference on Artificial Intelligence*, Vol. 2019, pp. 730–736.
[57] Gomez, A. N., Ren, M., Urtasun, R. & Grosse, R. B. (2017) The reversible residual network: backpropagation without storing activations. In: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (editors), *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., pp. 2214–2224.
[58] Grönwall, T. H. (1919) Note on the derivatives with respect to a parameter of the solutions of a system of differential equations. *Ann. Math.* **20**(4), 292–296.
[59] Günther, S., Ruthotto, L., Schroder, J. B., Cyr, E. C. & Gauger, N. R. (2020) Layer-parallel training of deep residual neural networks. *SIAM J. Math. Data Sci.* **2**(1), 1–23.
[60] Haber, E. & Ruthotto, L. (2017) Stable architectures for deep neural networks. *Inverse Probl.* **34**(1), 014004.
[61] Hager, W. W. (2000) Runge-Kutta methods in optimal control and the transformed adjoint system. *Numerische Mathematik* **87**(2), 247–282.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

[62] HAIRER, E., LUBICH, C. & WANNER, G. (2006) *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, Vol. 31, Springer Science & Business Media, Berlin.
[63] HAIRER, E., NØRSETT, S. P. & WANNER, G. (1993) *Solving Ordinary Differential Equations I*, 2nd ed., Springer Series in Computational Mathematics, Springer-Verlag, Berlin, Heidelberg.
[64] HAIRER, E. & WANNER, G. (2010) *Solving Ordinary Differential Equations. II*, Springer Series in Computational Mathematics, Vol. 14, Springer-Verlag, Berlin. Stiff and differential-algebraic problems, Second revised edition, paperback.
[65] HE, K., ZHANG, X., REN, S. & SUN, J. (2016) Deep residual learning for image recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.
[66] HOCHREITER, S. & SCHMIDHUBER, J. (1997) Flat minima. *Neural Comput.* **9**(1), 1–42.
[67] HOOGEBOOM, E., VAN DEN BERG, R. & WELLING, M. Emerging convolutions for generative normalizing flows. In: K. Chaudhuri and R. Salakhutdinov (editors), *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 97, Long Beach, California, USA, 09–15 June 2019, PMLR, pp. 2771–2780.
[68] HOPFIELD, J. J. (1982) Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci.* **79**(8), 2554–2558.
[69] HORNIK, K. (1991) Approximation capabilities of multilayer feedforward networks. *Neural Networks* **4**(2), 251–257.
[70] HUTCHINSON, M. F. (1990) A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Commun. Stat. Simul. Comput.* **19**(2), 433–450.
[71] HYVÄRINEN, A. & OJA, E. (2000) Independent component analysis: algorithms and applications. *Neural Networks* **13**, 411–430.
[72] ISERLES, A., MUNTHE-KAAS, H. Z., NØRSETT, S. P. & ZANNA, A. (2000) Lie-group methods. In: *Acta Numerica, 2000*, Acta Numerica, Vol. 9, Cambridge University Press, Cambridge, pp. 215–365.
[73] ITO, K. & JIN, B. (2014) *Inverse Problems - Tikhonov Theory and Algorithms*, World Scientific, Singapore.
[74] JACOBSEN, J.-H., SMEULDERS, A. W. M. & OYALLON, E. (2018) i-RevNet: deep invertible networks. In: *International Conference on Learning Representations*.
[75] KARRAS, T., LAINE, S. & AILA, T. (2019) A style-based generator architecture for generative adversarial networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410.
[76] KESKAR, N. S., MUDIGERE, D., NOCEDAL, J., SMELYANSKIY, M. & TANG, P. T. P. (2017) On large-batch training for deep learning: generalization gap and sharp minima. In: *ICLR*.
[77] KINGMA, D. P. & BA, J. (2015) Adam: a method for stochastic optimization. In: *ICLR*.
[78] KINGMA, D. P. & DHARIWAL, P. (2018) Glow: generative flow with invertible 1x1 convolutions. In: *Advances in Neural Information Processing Systems*, pp. 10215–10224.
[79] KOBAYASHI, S. & NOMIZU, K. (1996) *Foundations of Differential Geometry*, Vol. I, Wiley Classics Library, John Wiley & Sons, Inc., New York. Reprint of the 1963 original, A Wiley-Interscience Publication.
[80] KONDOR, R., LIN, Z. & TRIVEDI, S. (2018) Clebsch–Gordan Nets: a Fully Fourier Space Spherical Convolutional Neural Network. *Advances in Neural Information Processing Systems*, **31**, 10117–10126.
[81] KONDOR, R. & TRIVEDI, S. (2018) On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups. arXiv:1802.03690.
[82] KRIZHEVSKY, A., SUTSKEVER, I. & HINTON, G. E. (2012) Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, **25**, 1097–1105.
[83] LECUN, Y. (1988) A theoretical framework for back-propagation. In: *Proceedings of the 1988 Connectionist Models Summer School*, Vol. 1, CMU, Morgan Kaufmann, Pittsburgh, PA, pp. 21–28.
[84] LECUN, Y. & BENGIO, Y. (1995) Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, **3361**(10), 1995.

[85] LeCun, Y., Bengio, Y. & Hinton, G. (2015) Deep learning. *Nature* **521**(7553), 436–444.
[86] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1989) Backpropagation applied to handwritten zip code recognition. *Neural Comput.* **1**(4), 541–551.
[87] Li, J., Li, F. & Todorovic, S. (2019) Efficient Riemannian optimization on the Stiefel manifold via the Cayley transform. In: *International Conference on Learning Representations*.
[88] Li, Q., Chen, L., Tai, C. & E, W. (2018) Maximum principle based algorithms for deep learning. *J. Mach. Learn. Res.* **18**, 1–29.
[89] Li, Q. & Hao, S. (2018) An optimal control approach to deep learning and applications to discrete-weight neural networks. In: *Proceedings of the 35th International Conference on Machine Learning*.
[90] Li, Q., Tai, C. & E, W. (2019) Stochastic modified equations and dynamics of stochastic gradient algorithms I: mathematical foundations. *J. Mach. Learn. Res.* **20**, 1–47.
[91] Li, S. T. J. & Fuxin, L. (2020) Efficient Riemannian optimization on the Stiefel manifold via the Cayley transform. In: *ICLR 2020*.
[92] Linnainmaa, S. (1970) *The Representation of the Cumulative Rounding Error of an Algorithm as a Taylor Expansion of the Local Rounding Errors*. Master's Thesis (in Finnish), University Helsinki, pp. 6–7.
[93] Lu, Y., Zhong, A., Li, Q. & Dong, B. (2018) Beyond finite layer neural networks: bridging deep architectures and numerical differential equations. In: *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*.
[94] Lyu, K. & Li, J. (2020) Gradient descent maximizes the margin of homogeneous neural networks. In: *International Conference on Learning Representations*.
[95] Maddison, C. J., Paulin, D., Teh, Y. W., O'Donoghue, B. & Doucet, A. (2018) Hamiltonian Descent Methods. arXiv preprint arXiv:1809.05042.
[96] Martens, J. (2014) New Insights and Perspectives on the Natural Gradient Method. arXiv preprint arXiv:1412.1193.
[97] Marthinsen, H. & Owren, B. (2016) Geometric integration of non-autonomous linear Hamiltonian problems. *Adv. Comput. Math.* **42**(2), 313–332.
[98] Massaroli, S., Poli, M., Califano, F., Faragasso, A., Park, J., Yamashita, A. & Asama, H. (2019) Port-Hamiltonian Approach to Neural Network Training. arXiv preprint arXiv:1909.02702.
[99] McLachlan, R. & Perlmutter, M. (2001) Conformal Hamiltonian systems. *J. Geom. Phys.* **39**(4), 276–300.
[100] McLachlan, R. I. & Quispel, G. R. W. (2002) Splitting methods. *Acta Numer.* **11**, 341–434.
[101] McLachlan, R. I., Quispel, G. R. W. & Robidoux, N. (1999) Geometric integration using discrete gradients. *R. Soc. Lond. Philos. Trans. Ser. A Math. Phys. Eng. Sci.* **357**(1754), 1021–1045.
[102] Modin, K. (2016) Geometry of Matrix Decompositions seen through Optimal Transport and Information Geometry. arXiv preprint arXiv:1601.01875.
[103] Ng, A. Y. (2004) Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In: *Proceedings of the 21 st International Conference on Machine Learning*.
[104] Nocedal, J. & Wright, S. (2006) *Numerical Optimization*, Springer Science & Business Media, Berlin.
[105] O'Donoghue, B. & Maddison, C. J. (2019) Hamiltonian descent for composite objectives. In: *Advances in Neural Information Processing Systems*, pp. 14443–14453.
[106] Parpas, P. & Muir, C. (2019) Predict Globally, Correct Locally: Parallel-in-Time Optimal Control of Neural Networks. arXiv, 1974.
[107] Pascanu, R. & Bengio, Y. (2013) Revisiting Natural Gradient for Deep Networks. arXiv preprint arXiv:1301.3584.
[108] Petersen, P. & Voigtlaender, F. (2019) Equivalence of approximation by convolutional neural networks and fully-connected networks. *Proc. Am. Math. Soc.* **148**(4), 1567–1581.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press

[109] PONTRYAGIN, L. S. (1987) *Mathematical Theory of Optimal Processes*, Classics of Soviet Mathematics, Taylor & Francis, Montreux.
[110] PUTZKY, P. & WELLING, M. (2019) Invert to learn to invert. In: *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., pp. 446–456.
[111] RANZATO, M. A., BOUREAU, Y.-L. & LE CUN, Y. (2009) Sparse feature learning for deep belief networks. In: *Advances in Neural Information Processing Systems 20 - Proceedings of the 2007 Conference*.
[112] REDDI, S. J., KALE, S. & KUMAR, S. (2018) On the convergence of Adam and beyond. In: *ICLR*.
[113] REZENDE, D. J. & MOHAMED, S. (2015) Variational inference with normalizing flows. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, JMLR.org., pp. 1530–1538.
[114] ROBBINS, H. & MONRO, S. (1951) A stochastic approximation method. *Ann. Math. Stat.*, **22**(3), 400–407.
[115] ROCCA, F., PRATO, C. M. & FERRETTI, A. (1997) An overview of ERS-SAR interferometry. In: *Proceedings of the 3rdERS Symposium on Space at the Service of our Environment, Florence, Italy*.
[116] RUTHOTTO, L. & HABER, E. (2019) *Deep neural networks motivated by partial differential equations.* Journal of Mathematical Imaging and Vision, 1–13. Springer, Berlin.
[117] SHALEV-SHWARTZ, S. & BEN-DAVID, S. (2014) *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, Cambridge.
[118] SU, W., BOYD, S. P. & CANDES, E. J. (2014) A differential equation for modeling nesterov's accelerated gradient method: theory and insights. In: *NIPS*, Vol. 14, pp. 2510–2518.
[119] TAYLOR, G., BURMEISTER, R., XU, Z., SINGH, B., PATEL, A. & GOLDSTEIN, T. (2016) Training neural networks without gradients: a scalable ADMM approach. In: *ICML*.
[120] TESHIMA, T., ISHIKAWA, I., TOJO, K., OONO, K., IKEDA, M. & SUGIYAMA, M. (2020) Coupling-based Invertible Neural Networks are Universal Diffeomorphism Approximators. arXiv preprint arXiv:2006.11469.
[121] THOMAS, N., SMIDT, T., KEARNES, S., YANG, L., LI, L., KOHLHOFF, K. & RILEY, P. (2018) Tensor Field Networks: Rotation- and Translation-Equivariant Neural Networks for 3D Point Clouds. arXiv:1802.08219.
[122] THORPE, M. & VAN GENNIP, Y. (2018) Deep Limits of Residual Neural networks. arXiv preprint arXiv:1810.11741.
[123] UDRIŞTE, C. (1994) *Convex Functions and Optimization Methods on Riemannian Manifolds*, Mathematics and its Applications, Vol. 297, Kluwer Academic Publishers Group, Dordrecht.
[124] ULYANOV, D., VEDALDI, A. & LEMPITSKY, V. (2018) Deep image prior. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9446–9454.
[125] VAN DER SCHAFT, A. & JELTSEMA, D. (2014) Port-Hamiltonian systems theory: an introductory overview. *Found. Trends Syst. Control* **1**(2–3), 173–378.
[126] VINCENT, P., LAROCHELLE, H., LAJOIE, I., BENGIO, Y. & MANZAGOL, P.-A. (2010) Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.* **11**, 3371–3408.
[127] WANG, X., MA, S., GOLDFARB, D. & LU, W. (2017) Stochastic quasi-Newton methods for nonconvex stochastic optimization. *SIAM J. Optim.* **27**(2), 927–956.
[128] WEILER, M., GEIGER, M., WELLING, M., BOOMSMA, W. & COHEN, T. (2018) 3D steerable CNNs: learning rotationally equivariant features in volumetric data. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 10402–10413.
[129] WEILER, M., HAMPRECHT, F. A. & STORATH, M. (2018) Learning steerable filters for rotation equivariant CNNs. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 849–858
[130] WEINMANN, A., DEMARET, L. & STORATH, M. (2014) Total variation regularization for manifold-valued data. *SIAM J. Imaging Sci.* **7**(4), 2226–2257.
[131] WITHERS, C. S. & NADARAJAH, S. (2010) log det A = tr log A. *Int. J. Math. Edu. Sci. Technol.* **41**(8), 1121–1124.

[132] WORRALL, D. E., GARBIN, S. J., TURMUKHAMBETOV, D. & BROSTOW, G. J. (2017) Harmonic networks: Deep translation and rotation equivariance. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5028–5037.

[133] XIE, Y., BYRD, R. H. & NOCEDAL, J. (2020) Analysis of the BFGS method with errors. *SIAM J. Optim.* **30**(1), 182–209.

[134] YANG, H. H. & AMARI, S.-I. (1997) Natural gradient descent for training multi-layer perceptrons. *Submitted to IEEE Trans. Neural Networks*.

[135] YANG, Z., LIU, Y., BAO, C. & SHI, Z. (2020) Interpolation between residual and non-residual networks. In: *International Conference on Machine Learning*, PMLR, pp. 10736–10745.

[136] YAROTSKY, D. (2018) Universal Approximations of Invariant Maps by Neural Networks. arXiv:1804.10306.

[137] ZAHEER, M., REDDI, S., SACHAN, D., KALE, S. & KUMAR, S. (2018) Adaptive methods for nonconvex optimization. In: *Advances in Neural Information Processing Systems*, pp. 9793–9803.

[138] ZHANG, G., MARTENS, J. & GROSSE, R. B. (2019) Fast convergence of natural gradient descent for over-parameterized neural networks. In: *Advances in Neural Information Processing Systems*, pp. 8080–8091.

[139] ZHANG, L. & SCHAEFFER, H. (2020) Forward stability of resNet and its variants. *J. Math. Imaging Vis.* **62**(3), 328–351.

https://doi.org/10.1017/S0956792521000139 Published online by Cambridge University Press