Manuscript submitted to
AIMS' Journals
Volume **00**, Number **0**, Xxxx **XXXX**

doi:10.3934/xx.xx.xx.xx

pp. **000–000**

DEEP LEARNING AS OPTIMAL CONTROL PROBLEMS: MODELS AND NUMERICAL METHODS

Martin Benning

School of Mathematical Sciences, Queen Mary University of London
London E1 4NS, UK

Elena Celledoni

Department of Mathematical Sciences, NTNU
7491 Trondheim, Norway

Matthias J. Ehrhardt

Institute for Mathematical Innovation, University of Bath
Bath BA2 7JU, UK

Brynjulf Owren

Department of Mathematical Sciences, NTNU
7491 Trondheim, Norway

Carola-Bibiane Schönlieb*

Department of Applied Mathematics and Theoretical Physics, University of Cambridge
Cambridge CB3 0WA, UK

(Communicated by the associate editor name)

2010 *Mathematics Subject Classification.* Primary: 65L10, 65K10, 68Q32; Secondary: 49J15.

Key words and phrases. Deep learning, optimal control, Runge–Kutta methods, Hamiltonian boundary value problems.

* Corresponding author: Carola-Bibiane Schönlieb.

ABSTRACT. We consider recent work of [17] and [9], where deep learning neural networks have been interpreted as discretisations of an optimal control problem subject to an ordinary differential equation constraint. We review the first order conditions for optimality, and the conditions ensuring optimality after discretisation. This leads to a class of algorithms for solving the discrete optimal control problem which guarantee that the corresponding discrete necessary conditions for optimality are fulfilled. The differential equation setting lends itself to learning additional parameters such as the time discretisation. We explore this extension alongside natural constraints (e.g. time steps lie in a simplex). We compare these deep learning algorithms numerically in terms of induced flow and generalisation ability.

1. **Introduction.** Deep learning has had a transformative impact on a wide range of tasks related to Artificial Intelligence, ranging from computer vision and speech recognition to playing games [23, 27].

Despite impressive results in applications, the mechanisms behind deep learning remain rather mysterious, resulting in deep neural networks mostly acting as black-box algorithms. Consequently, also theoretical guarantees for deep learning are scarce, with major open problems residing in the mathematical sciences. An example are questions around the stability of training as well as the design of stable architectures. These questions are fed by results on the possible instabilities of the training (due to the high-dimensional nature of the problem in combination with its non-convexity) [43, 11] which are connected to the lack of generalisability of the learned architecture, and adversarial vulnerability of trained networks [44] that can result in instabilities in the solution and gives rise to systematic attacks with which networks can be fooled [16, 24, 33]. In this work we want to shed light on these issues by interpreting deep learning as an optimal control problem in the context of binary classification problems. Our work is mostly inspired by a very early paper by LeCun [29], and a series of recent works by Haber, Ruthotto et al. [17, 9].

Classification in machine learning: *Classification* is a key task in machine learning; the goal is to learn functions, also known as *classifiers*, that map their input arguments onto a discrete set of labels that are associated with a particular class. A simple example is image classification, where the input arguments are images that depict certain objects, and the classifier aims to identify the class to which the object depicted in the image belongs to. We can model such a classifier as a function $g : \mathbb{R}^n \to \{c^0, c^1, \dots, c^{K-1}\}$ that takes n-dimensional

real-valued vectors and maps them onto a discrete set of K class labels. Note that despite using numerical values, there is no particular ordering of the class labels. The special case of $K = 2$ classes (and class labels) is known as *binary classification*; for simplicity, we strictly focus on binary classification for the remainder of this paper. The extension to multi-class classification is straightforward, see e.g. [4].

In supervised machine learning, the key idea is to find a classifier by estimating optimal parameters of a parametric function given pairs of data samples $\{(x_i, c_i)\}_{i=1}^m$, for $c_i \in \{c^0, c^1\}$, and subsequently defining a suitable classifier that is parameterised with these parameters. The process of finding suitable parameters is usually formulated as a generalised regression problem, i.e. we estimate parameters u, W, μ by minimising a cost function of the form[1]

$$\frac{1}{2}\sum_{i=1}^{m} |\, \mathcal{C}\left(Wh(x_i, u) + \mu\right) - c_i \,|^2 + \mathcal{R}(u)\,, \tag{1}$$

with respect to u, W and μ. Here h is a model function parameterised by parameters u that transforms inputs $x_i \in \mathbb{R}^n$ onto n-dimensional outputs. The vector $W \in \mathbb{R}^{1\times n}$ is a *weight* vector that weights this n-dimensional model output, whereas $\mu \in \mathbb{R}$ is a scalar that allows a *bias* of the weighted model output, and $\mathcal{C} : \mathbb{R}^n \to \mathbb{R}$ is the so-called *hypothesis-function* (cf. [20, 17]) that maps this weighted and biased model output to a scalar value that can be compared to the class label c_i. The function $\mathcal{R}$ is a regularisation function that is chosen to ensure some form of regularity of the parameters u and existence of parameters that minimise (1). Typical regularisation functions include the composition of the squared 2-norm with a linear operator (Tikhonov–Phillips regularisation [47, 34]; in statistics this technique is called ridge regression, while in machine learning it is known as weight decay [35]) or the 1-norm to induce sparsity of the weights [38, 46]. However, depending on the application, many different choices of regularisation functions are possible.

Note that minimising (1) yields parameters that minimise the deviation of the output of the hypothesis function and the given labels. If we denote those parameters that minimise (1) by $\hat{W}$, $\hat{\mu}$ and $\hat{u}$, and if the hypothesis function $\mathcal{C}$ maps directly onto the discrete set $\{c_0, c_1\}$, then a suitable classifier can simply be defined via

$$g(x) := \mathcal{C}\left(\hat{W}h(x, \hat{u}) + \hat{\mu}\right)\,.$$

[1] One can of course use other cost functions such as the cross-entropy [4]. Our theory includes all smooth cost functions.

However, in practice the hypothesis function is often rather continuous and does not map directly on the discrete values $\{c_0, c_1\}$. In this scenario, a classifier can be defined by subsequent thresholding. Let c_0 and c_1 be real numbers and w.l.o.g. $c_0 < c_1$, then a suitable classifier can for instance be defined via

$$g(x) := \begin{cases} c_0 & \mathcal{C}\left(\hat{W}h(x,\hat{u}) + \hat{\mu}\right) \leq \frac{c_0+c_1}{2} \\ c_1 & \mathcal{C}\left(\hat{W}h(x,\hat{u}) + \hat{\mu}\right) > \frac{c_0+c_1}{2} \end{cases}.$$

Deep learning as an optimal control problem: One recent proposal towards the design of deep neural network architectures is [13, 17, 9, 48, 31]. There, the authors propose an interpretation of deep learning by the popular Residual neural Network (ResNet) architecture [21] as discrete optimal control problems. Let

$$u^{[j]} := \left(K^{[j]}, \beta^{[j]}\right), \qquad j = 0, \ldots, N-1, \qquad u = \left(u^{[0]}, \ldots, u^{[N-1]}\right),$$

where $K^{[j]}$ is a $n \times n$ matrix of weights, $\beta^{[j]}$ represents the biases, and N is the number of layers.

In order to use ResNet for binary classification we can define the output of the model function h in (1) as the output of the ResNet. With the ResNet state variable denoted by $y = (y^{[0]}, \ldots, y^{[N]})$, $y^{[j]} = (y_1^{[j]}, \ldots, y_n^{[j]})$, this implies that the classification problem (1) can be written as a constraint minimisation problem of the form

$$\min_{y,u,W,\mu} \sum_{i=1}^{m} \left| \mathcal{C}\left(W y_i^{[N]} + \mu\right) - c_i \right|^2 + \mathcal{R}(u), \tag{2}$$

subject to the constraint

$$y_i^{[j+1]} = y_i^{[j]} + \Delta t\, f(y_i^{[j]}, u^{[j]}), \qquad j = 0, \ldots, N-1, \qquad y_i^{[0]} = x_i\,. \tag{3}$$

Here Δt is a parameter which for simplicity at this stage can be chosen to be equal to 1 and whose role will become clear in what follows. The constraint (3) is the ResNet parametrisation of a neural network [21]. In contrast, the widely used feed-forward network that we will also investigate later is given by

$$y_i^{[j+1]} = f(y_i^{[j]}, u^{[j]}), \qquad j = 0, \ldots, N-1, \qquad y_i^{[0]} = x_i\,. \tag{4}$$

For deep learning algorithms, one often has

$$f(y_i^{[j]}, u^{[j]}) := \sigma\left(K^{[j]}\, y_i^{[j]} + \beta^{[j]}\right), \tag{5}$$

where σ is a suitable activation function acting component-wise on its arguments. For a more extensive mathematical introduction to deep learning we recommend [22].

Suppose, in what follows, that $y_i = y_i(t)$ and $u = u(t) = (K(t), \beta(t))$, $t \in [0, T]$, are functions of time and $y_i^{[j]} \approx y_i(t_j)$. To view (2) and (3) as a discretisation of an optimal control problem [9], one observes that the constraint equation (3) is the discretisation of the ordinary differential equation (ODE) $\dot{y}_i = f(y_i, u)$, $y_i(0) = x_i$, on $[0, T]$, with step-size Δt and with the forward Euler method. In the continuum, the following optimal control problem is obtained [9],

$$\min_{y,u,W,\mu} \sum_{i=1}^{m} \left| \mathcal{C}\left(W\, y_i(T) + \mu\right) - c_i \right|^2 + \mathcal{R}(u) \tag{6}$$

subject to the ODE constraint

$$\dot{y}_i = f(y_i, u), \quad t \in [0, T], \quad y_i(0) = x_i. \tag{7}$$

Assuming that problem (6)-(7) satisfies necessary conditions for optimality [42, ch. 9], and that a suitable activation function and cost function have been chosen, a number of new deep learning algorithms can be generated. For example, the authors of [10, 32] propose to use accurate approximations of (7) obtained by black-box ODE solvers. Alternatively, some of these new strategies are obtained by considering constraint ODEs (7) with different structural properties, e.g. taking f to be a Hamiltonian vector field, and by choosing accordingly the numerical integration methods to approximate (7), [9, 17]. This entails augmenting the dimension, e.g. by doubling the number of variables in the ODE, a strategy also studied in [15]. Stability is perhaps not important in networks with a fixed and modest number of layers. However, in designing and understanding deep neural networks, it is of importance to analyse its behaviour when the depth grows towards infinity. The stability of neural networks has been an important issue in many papers in this area. The underlying continuous dynamical system offers a common framework for analysing the behaviour of different architectures, for instance through backward error analysis, see e.g. [19]. The optimality conditions are useful for ensuring consistency between the discrete and continuous optima, and possibly the adjoint variables can be used to analyse the sensitivity of the network to perturbations in initial data. We also want to point out that the continuous limit of neural networks is not only relevant for the study of optimal control problems, but also for optimal transport [41] or data assimilation [1] problems.

Our contribution: The main purpose of this paper is the investigation of different discretisations of the underlying continuous deep learning problem (6)-(7). In [17, 9] the authors investigate different

ODE-discretisations for the neural network (7), with a focus on deriving a neural network that describes a 'stable' flow, i.e. the solution $y(T)$ should be bounded by the initial condition $y(0)$.

Our point of departure from the state-of-the-art will be to outline the well established theory on optimal control, numerical ODE problems based on [18, 37, 39, 30], where we investigate the complete optimal control problem (6)-(7) under the assumption of necessary conditions for optimality [42, ch. 9].

The formulation of the deep learning problem (2)-(3) is a first-discretise-then-optimise approach to optimal control, where ODE (7) is first discretised with a forward Euler method to yield an optimisation problem which is then solved with gradient descent (direct method). In this setting the forward Euler method could be easily replaced by a different and more accurate integration method, but the back-propagation for computing the gradients of the discretised objective function will typically become more complicated to analyse.

Here, we propose a first-optimise-then-discretise approach for deriving new deep learning algorithms. There is a two-point boundary value Hamiltonian problem associated to (6)-(7) expressing first order optimality conditions of the optimal control problem [36]. This boundary value problem consists of (7), with $y_i(0) = x$, together with its adjoint equation with boundary value at final time T, and in addition an algebraic constraint. In the first-optimise-then-discretise approach, this boundary value problem is solved by a numerical integration method. It is natural to solve equation (7) forward in time with a Runge–Kutta method (with non vanishing weights b_i, $i = 1, \dots, s$), while the adjoint equation must be solved backward in time and with a matching Runge–Kutta method (with weights satisfying (19)) and imposing the constraints at each time step. If the combination of the forward integration method and its counterpart used backwards in time form a symplectic partitioned Runge–Kutta method then the overall discretisation is equivalent to a first-discretise-then-optimise approach, but with an efficient and automatic computation of the gradients [18, 39], see Proposition 1.

We implement discretisation strategies based on different Runge–Kutta methods for (6)-(7). To make the various methods comparable to each other, we use the same learned parameters for every Runge–Kutta stage, in this way the total the number of parameters will not depend on how many stages each method has. The discretisations are adaptive in time, and learning the step-sizes the number of layers is determined automatically by the algorithms. From the optimal control formulation we derive different instances of deep learning algorithms

(2)-(3) by numerical discretisations of the first-order optimality conditions using a partitioned Runge–Kutta method.

Outline of the paper: In Section 2 we derive the optimal control formulation of (6)-(7) and discuss its main properties. In particular, we derive the variational equation, the adjoint equation, the associated Hamiltonian and first-order optimality conditions. Different instances of the deep learning algorithm (2)-(3) are derived in Section 3 by using symplectic partitioned Runge–Kutta methods applied to the constraint equation (9), and the adjoint equations (12) and (13). Using partitioned Runge–Kutta discretisation guarantees equivalence of the resulting optimality system to the one derived from a first-discretise-then-optimise approach using gradient descent, cf. Proposition 1. In Section 4 we derive several new deep learning algorithms from such optimal control discretisations, and investigate by numerical experiments their dynamics in Section 5 on a selection of toy problems for binary classification in two dimensions.

2. Properties of the optimal control problem.

In this section we review established literature on optimal control which justifies the use of the numerical methods of the next section.

2.1. Variational equation.

In this section, we consider a slightly simplified formulation of (6)-(7). In particular, for simplicity we discard the term $\mathcal{R}(u)$ in (6), and remove the index "i" in (6)-(7) and the summation over the number of data points. Moreover, as we here focus on the ODE (7), we also remove the dependency on the classification parameters W and μ for now. We rewrite the optimal control problem in the simpler form

$$\min_{y,u} \mathcal{J}(y(T)), \tag{8}$$

subject to the ODE constraint

$$\dot{y} = f(y,u), \quad y(0) = x. \tag{9}$$

Then, the variational equation for (8)-(9) reads

$$\frac{d}{dt}v = \partial_y f(y(t),u(t))\, v + \partial_u f(y(t),u(t))\, w \tag{10}$$

where $\partial_y f$ is the Jacobian of f with respect to y, $\partial_u f$ is the Jacobian of f with respect to u, and v is the variation in y, while w is the variation in u[2]. Since $y(0) = x$ is fixed, $v(0) = 0$.

[2] $\tilde{y}(t) = y(t) + \xi v(t)$ for $|\xi| \to 0$ and similarly for $\tilde{u}(t) = u(t) + \xi w(t)$ $|\xi| \to 0$.

2.2. **Adjoint equation.** The adjoint of (10) is a system of ODEs for a variable $p(t)$, obtained assuming

$$\langle p(t), v(t)\rangle = \langle p(0), v(0)\rangle, \qquad \forall t \in [0,T]. \tag{11}$$

Then (11) implies

$$\langle p(t), \dot{v}(t)\rangle = -\langle \dot{p}(t), v(t)\rangle,$$

an integration-by-parts formula which together with (10) leads to the following equation for p:

$$\frac{d}{dt}p = -\left(\partial_y f(y(t),u(t))\right)^T (p)\ , \tag{12}$$

with constraint

$$\left(\partial_u f(y(t),u(t))\right)^T p = 0\,, \tag{13}$$

see [39]. Here we have denoted by $(\partial_y f)^T$ the transpose of $\partial_y f$ with respect to the Euclidean inner product $\langle\cdot,\cdot\rangle$, and similarly $(\partial_u f)^T$ is the transpose of $\partial_u f$.

2.3. **Associated Hamiltonian system.** For such an optimal control problem, there is an associated Hamiltonian system with Hamiltonian

$$\mathcal{H}(y,p,u) := \langle p, f(y,u)\rangle$$

with

$$\dot{y} = \partial_p \mathcal{H}, \quad \dot{p} = -\partial_y \mathcal{H}, \quad \partial_u \mathcal{H} = 0, \tag{14}$$

where we recognise that the first equation $\dot{y} = \partial_p \mathcal{H}$ coincides with (9), the second $\dot{p} = -\partial_y \mathcal{H}$ with (12) and the third $\partial_u \mathcal{H} = 0$ with (13).

The constraint Hamiltonian system is a differential algebraic equation of index one if the Hessian $\partial_{u,u}\mathcal{H}$ is invertible. In this case, by the implicit function theorem there exists φ such that

$$u = \varphi(y,p), \quad \text{and} \quad \bar{\mathcal{H}}(y,p) = \mathcal{H}(y,p,\varphi(y,p)),$$

where the differential algebraic Hamiltonian system is transformed into a canonical Hamiltonian system of ODEs with Hamiltonian $\bar{\mathcal{H}}$. Notice that it is important to know that φ exists, but it is not necessary to compute φ explicitly for discretising the problem.

2.4. **First order necessary conditions for optimality.** The solution of the two point boundary value problem (9) and (12),(13) with $y(0) = x$ and

$$p(T) = \partial_y \mathcal{J}(y)\big|_{y=y(T)}\,,$$

has the following property

$$\langle \partial_y \mathcal{J}(y)\big|_{y=y(T)}\,, v(T)\rangle = \langle p(T), v(T)\rangle = \langle p(0), v(0)\rangle = 0,$$

so the variation $v(T)$ is orthogonal to the gradient of the cost function $\partial_y \mathcal{J}(y)|_{y=y(T)}$. This means that the solution $(y(t), v(t), p(t))$ satisfies the first order necessary conditions for extrema of $\mathcal{J}$ (Pontryagin maximum principle) [36], see also [42, ch. 9.2].

3. **Numerical discretisation of the optimal control problem.** We consider a time discrete setting $y^{[0]}, y^{[1]}, \dots, y^{[N]}$, $u^{[0]}, \dots, u^{[N-1]}$ and a cost function $\mathcal{J}(y^{[N]})$, assuming to apply a numerical time discretisation $y^{[j+1]} = \Phi_{\Delta t}(y^{[j]}, u^{[j]})$, $j = 0, \dots, N-1$ of (9), the discrete optimal control problem becomes

$$\min_{(y^{[j]}, u^{[j]})} \mathcal{J}(y^{[N]}),$$

subject to

$$y^{[j]} = \Phi_{\Delta t}(y^{[j-1]}, u^{[j-1]}), \quad y^{[0]} = x. \tag{15}$$

Here the subscript Δt denotes the discretisation step-size of the time interval $[0, T]$. This discrete optimal control problem corresponds to a deep learning algorithm with the outlined choices for f and $\mathcal{J}$, see for example [17].

We assume that $\Phi_{\Delta t}$ is a Runge–Kutta method with non vanishing weights for the discretisation of (9). Applying a Runge–Kutta method to (9), for example the forward Euler method, we obtain

$$y^{[j+1]} = y^{[j]} + \Delta t\, f(y^{[j]}, u^{[j]}),$$

and taking variations $y^{[j+1]} + \xi v^{[j+1]}$, $y^{[j]} + \xi v^{[j]}$, , $u^{[j]} + \xi w^{[j]}$, for $\xi \to 0$ one readily obtains the same Runge–Kutta method applied to the variational equation

$$v^{[j]} = v^{[j+1]} + \Delta t \left[\partial_y f(y^{[j]}, u^{[j]})\, v^{[j]} + \partial_u f(y^{[j]}, u^{[j]})\, w^{[j]}\right].$$

This means that taking variations is an operation that commutes with applying Runge–Kutta methods. This is a well known property of Runge–Kutta methods and also of the larger class of so called B-series methods, see for example [19, ch. VI.4, p. 191] for details.

In order to ensure that the first order necessary conditions for optimality from Section 2.4 are satisfied also after discretisation, fixing a certain Runge–Kutta method $\Phi_{\Delta t}$ for (9), we need to discretise the adjoint equations (12) and (13) such that the overall method is a symplectic, partitioned Runge–Kutta method for the system spanned by (9), (12) and (13). This will in particular guarantee the preservation of the quadratic invariant (11), as emphasised in [39]. The general format of a partitioned Runge–Kutta method as applied to (9), (12) and (13)

is for $j = 0, \ldots, N-1$

$$y^{[j+1]} = y^{[j]} + \Delta t \sum_{i=1}^{s} b_i \, f_i^{[j]} \tag{16a}$$

$$f_i^{[j]} = f(y_i^{[j]}, u_i^{[j]}), \quad i = 1, \ldots, s, \tag{16b}$$

$$y_i^{[j]} = y^{[j]} + \Delta t \sum_{l=1}^{s} a_{i,l} \, f_l^{[j]}, \quad i = 1, \ldots, s \tag{16c}$$

$$p^{[j+1]} = p^{[j]} + \Delta t \sum_{i=1}^{s} \tilde{b}_i \, \ell_i^{[j]} \tag{17a}$$

$$\ell_i^{[j]} = -\partial_y f(y_i^{[j]}, u_i^{[j]})^T p_i^{[j]}, \quad i = 1, \ldots, s, \tag{17b}$$

$$p_i^{[j]} = p^{[j]} + \Delta t \sum_{l=1}^{s} \tilde{a}_{i,l} \ell_l^{[j]}, \quad i = 1, \ldots, s \tag{17c}$$

$$\Big(\partial_u f(y_i^{[j]}, u_i^{[j]})\Big)^T p_i^{[j]} = 0, \quad i = 1, \ldots, s \tag{18}$$

and boundary conditions $y^{[0]} = x$, $p^{[N]} := \partial \mathcal{J}(y^{[N]})$. We will assume $b_i \neq 0$, $i = 1, \ldots, s$. [3]

It is well known [19] that if the coefficients of a partitioned Runge–Kutta satisfy

$$b_i = \tilde{b}_i, \quad b_i \tilde{a}_{i,j} + \tilde{b}_j a_{i,j} - b_i \tilde{b}_j = 0, \quad c_i = \tilde{c}_i, \quad i, j = 1, \ldots, s, \tag{19}$$

then the partitioned Runge–Kutta preserves invariants of the form

$$S : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R},$$

where S is bilinear. As a consequence the invariant $S(v(t), p(t)) := \langle p(t), v(t) \rangle$ (11) will be preserved by such method. These partitioned Runge–Kutta methods are called symplectic.

The simplest symplectic partitioned Runge–Kutta method is the symplectic Euler method, which is a combination of the explicit Euler method $b_1 = 1$, $a_{1,1} = 0$, $c_1 = 0$ and the implicit Euler method $\tilde{b}_1 = 1$, $\tilde{a}_{1,1} = 1$ $\tilde{c}_1 = 1$. This method applied to (9), (12) and (13) gives

$$\begin{aligned} y^{[j+1]} &= y^{[j]} + \Delta t \, f(y^{[j]}, u^{[j]}), \\ p^{[j+1]} &= p^{[j]} - \Delta t \, \left(\partial_y f(y^{[j]}, u^{[j]})\right)^T p^{[j+1]}, \\ 0 &= \left(\partial_u f(y^{[j]}, u^{[j]})\right)^T p^{[j+1]}, \end{aligned}$$

for $j = 0, \ldots, N-1$ and with the boundary conditions $y^{[0]} = x$, and $p^{[N]} := \partial_y \mathcal{J}(y^{[N]})$.

[3] Generic b_i in the context of optimal control is discussed in [39].

Proposition 1. *If (9), (10) and (12) are discretised with (16)-(18) with $b_i \neq 0$, $i = 1, \dots, s$, then the first order necessary conditions for optimality for the discrete optimal control problem*

$$\min_{\substack{\{u_i^{[j]}\}_{j=0}^{N-1}, \\ \{y^{[j]}\}_{j=1}^{N}, \{y_i^{[j]}\}_{j=1}^{N},}} \mathcal{J}(y^{[N]}), \tag{20}$$

subject to

$$y^{[j+1]} = y^{[j]} + \Delta t \sum_{i=1}^{s} b_i \, f_i^{[j]} \tag{21}$$

$$f_i^{[j]} = f(y_i^{[j]}, u_i^{[j]}), \quad i = 1, \dots, s, \tag{22}$$

$$y_i^{[j]} = y^{[j]} + \Delta t \sum_{m=1}^{s} a_{i,m} \, f_m^{[j]}, \quad i = 1, \dots, s, \tag{23}$$

are satisfied.

Proof. See appendix A. □

In (16)–(18) it is assumed that there is a parameter set $u_i^{[j]}$ for each of the s stages in each layer. This may simplified by considering only one parameter set $u^{[j]}$ per layer. Discretisation of the Hamiltonian boundary value problem with a symplectic partitioned Runge–Kutta method yields in this case the following expressions for the derivative of the cost function with respect to the controls.

Proposition 2. *Let $y^{[j]}$ and $p^{[j]}$ be given by (16) and (17) respectively. Then the gradient of the cost function $\mathcal{J}$ with respect to the controls is given by*

$$\ell_i^{[j]} = -\partial_y f(y_i^{[j]}, u^{[j]})^T \left(p^{[j+1]} - \Delta t \sum_{k=1}^{s} \frac{a_{k,i} b_k}{b_i} \ell_k^{[j]} \right) \quad i = 1, \dots, s \tag{24a}$$

$$\partial_{u^{[j]}} \mathcal{J}(y^{[N]}) = \Delta t \sum_{i=1}^{s} b_i \partial_{u^{[j]}} f(y_i^{[j]}, u^{[j]})^T \left(p^{[j+1]} - \Delta t \sum_{k=1}^{s} \frac{a_{k,i} b_k}{b_i} \ell_k^{[j]} \right). \tag{24b}$$

Remark 1. *In the case that the Runge–Kutta method is explicit we have $a_{k,i} = 0$ for $i \geq k$. In this case the stages $\ell_s^{[j]}, \ell_{s-1}^{[j]}, \dots, \ell_1^{[j]}$ can be computed explicitly from (24a).*

Remark 2. *For the explicit Euler method, these formulas greatly simplify and the derivative of the cost function with respect to the controls can be computed as*

$$y^{[j+1]} = y^{[j]} + \Delta t\, f(y^{[j]}, u^{[j]}) \tag{25}$$

$$p^{[j+1]} = p^{[j]} - \Delta t\, \partial_y f(y^{[j]}, u^{[j]})^T p^{[j+1]} \tag{26}$$

$$\partial_{u^{[j]}} \mathcal{J}(y^{[N]}) = \Delta t\, \partial_{u^{[j]}} f(y^{[j]}, u^{[j]})^T p^{[j+1]}. \tag{27}$$

Remark 3. *The convergence of the outlined Runge–Kutta discretisations to the continuous optimal control problem has been addressed in* [18] *see also* [26] *and recently also in the context of deep learning in* [45].

4. Optimal Control motivated Neural Network Architectures.

An ODE-inspired neural network architecture is uniquely defined by choosing f and specifying a time discretisation of (9). Here we will focus on the common choice $f(u, y) = \sigma(Ky + \beta), u = (K, \beta)$ (e.g. ResNet) and also discuss a novel option $f(u, y) = \alpha\, \sigma(Ky + \beta), u = (K, \beta, \alpha)$ which we will refer to as ODENet.

4.1. Runge–Kutta networks, e.g. ResNet.

Here we choose $f(u, y) = \sigma(Ky + \beta), u = (K, \beta)$. For simplicity we focus on the simplest Runge–Kutta method—the explicit Euler. This corresponds to the ResNet in the machine learning literature.
In this case the network relation (forward propagation) is given by

$$y^{[j+1]} = y^{[j]} + \Delta t\, \sigma\left(K^{[j]} y^{[j]} + \beta^{[j]}\right) \tag{28}$$

and gradients with respect to the controls can be computed by first solving for the adjoint variable (backpropagation)

$$\gamma^{[j]} = \sigma'\left(K^{[j]} y^{[j]} + \beta^{[j]}\right) \odot p^{[j+1]} \tag{29}$$

$$p^{[j+1]} = p^{[j]} - \Delta t\, K^{[j],T} \gamma^{[j]} \tag{30}$$

and then computing

$$\partial_{K^{[j]}} \mathcal{J}(y^{[N]}) = \Delta t\, \gamma^{[j]}\, y^{[j],T} \tag{31}$$

$$\partial_{\beta^{[j]}} \mathcal{J}(y^{[N]}) = \Delta t\, \gamma^{[j]}. \tag{32}$$

4.2. ODENet.

In contrast to the models we discussed so far, we can also enlarge the set of controls to model varying time steps. Let $u = (K, \beta, \alpha)$ and define

$$f(u, y) = \alpha\, \sigma(Ky + \beta)\,. \tag{33}$$

Algorithm 1 Training ODE-inspired neural networks with gradient descent.

Input: initial guess for the controls u, step-size τ

1: **for** $k = 1, \ldots$ **do**
2: forward propagation: compute y via (16)
3: backpropagation: compute p via (17)
4: compute gradient g via (24) and (41)
5: update controls: $u = u - \tau g$

The function α can be interpreted as varying time steps. Then the network relation (forward propagation) is given by

$$y^{[j+1]} = y^{[j]} + \Delta t\, \alpha^{[j]}\, \sigma\left(K^{[j]} y^{[j]} + \beta^{[j]}\right) \tag{34}$$

and gradients with respect to the controls can be computed by first solving for the adjoint variable (backpropagation)

$$\gamma^{[j]} = \alpha^{[j]}\, \sigma'\left(K^{[j]} y^{[j]} + \beta^{[j]}\right) \odot p^{[j+1]} \tag{35}$$

$$p^{[j+1]} = p^{[j]} - \Delta t\, K^{[j],T} \gamma^{[j]} \tag{36}$$

and then computing

$$\partial_{K^{[j]}} \mathcal{J}(y^{[N]}) = \Delta t\, \gamma^{[j]}\, y^{[j],T} \tag{37}$$

$$\partial_{\beta^{[j]}} \mathcal{J}(y^{[N]}) = \Delta t\, \gamma^{[j]} \tag{38}$$

$$\partial_{\alpha^{[j]}} \mathcal{J}(y^{[N]}) = \Delta t\, \left\langle p^{[j+1]}, \sigma\left(K^{[j]} y^{[j]} + \beta^{[j]}\right)\right\rangle . \tag{39}$$

It is natural to assume the learned time steps α should lie in the set of probability distributions

$$S = \left\{ \alpha \;\middle|\; \alpha \geq 0,\ \int \alpha = 1 \right\},$$

or discretised in the simplex

$$S = \left\{ \alpha \in \mathbb{R}^N \;\middle|\; \alpha^{[j]} \geq 0,\ \sum_j \alpha^{[j]} = 1 \right\}. \tag{40}$$

This discretised constraint can easily be incorporated into the learning process by projecting the gradient descent iterates onto the constraint set S. Efficient finite-time algorithms are readily available [12].

5. Numerical results.

Algorithm 2 Training ODE-inspired neural networks with gradient descent and backtracking.

Input: initial guess for controls u and parameter L, hyperparameters $\overline{\rho} > 1$ and $\underline{\rho} < 1$.

1: forward propagation: compute y with controls u via (16) and $\phi = \mathcal{J}(y^{[N]})$
2: **for** $k = 1, \ldots$ **do**
3: backpropagation: compute p via (17)
4: compute gradient g via (24) and (41)
5: **for** $t = 1, \ldots$ **do**
6: update controls: $\tilde{u} = u - \frac{1}{L} g$
7: forward propagation: compute $\tilde{y}$ with controls $\tilde{u}$ and $\tilde{\phi} = \mathcal{J}(\tilde{y}^{[N]})$
8: **if** $\tilde{\phi} \leq \phi + \langle g, \tilde{u} - u \rangle + \frac{L}{2} \|\tilde{u} - u\|^2$ **then**
9: accept: $u = \tilde{u}, y = \tilde{y}, \phi = \tilde{\phi}$, $L = \underline{\rho} L$
10: **break** inner loop
11: **else** reject: $L = \overline{\rho} L$

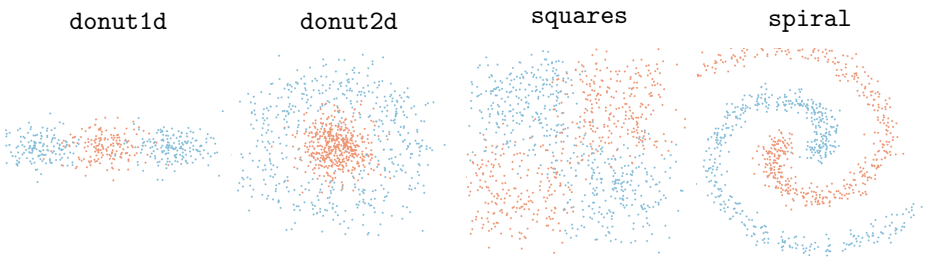

FIGURE 1. The four data sets used in the numerical study.

5.1. **Setting, Training and Data sets.** Throughout the numerical experiments we use labels $c_i \in \{0, 1\}$ and make use of the link function $\sigma(x) = \tanh(x)$ and hypothesis function $\mathcal{H}(x) = 1/(1 + \exp(-x))$. For all experiments we use two channels ($n = 2$) but vary the number of layers N[4].

In all numerical experiments we use gradient descent with backtracking, see Algorithm 2, to train the network (estimate the controls). The algorithm requires the derivatives with respect to the controls which we

[4]In this paper we make the deliberate choice of keeping the number of dimensions equal to the dimension of the original data samples rather than augmenting or doubling the number of dimensions as proposed in [15] or [17]. Numerical experiments after augmenting the dimension of the ODE (not reported here) led to improved performance for all the considered architectures.

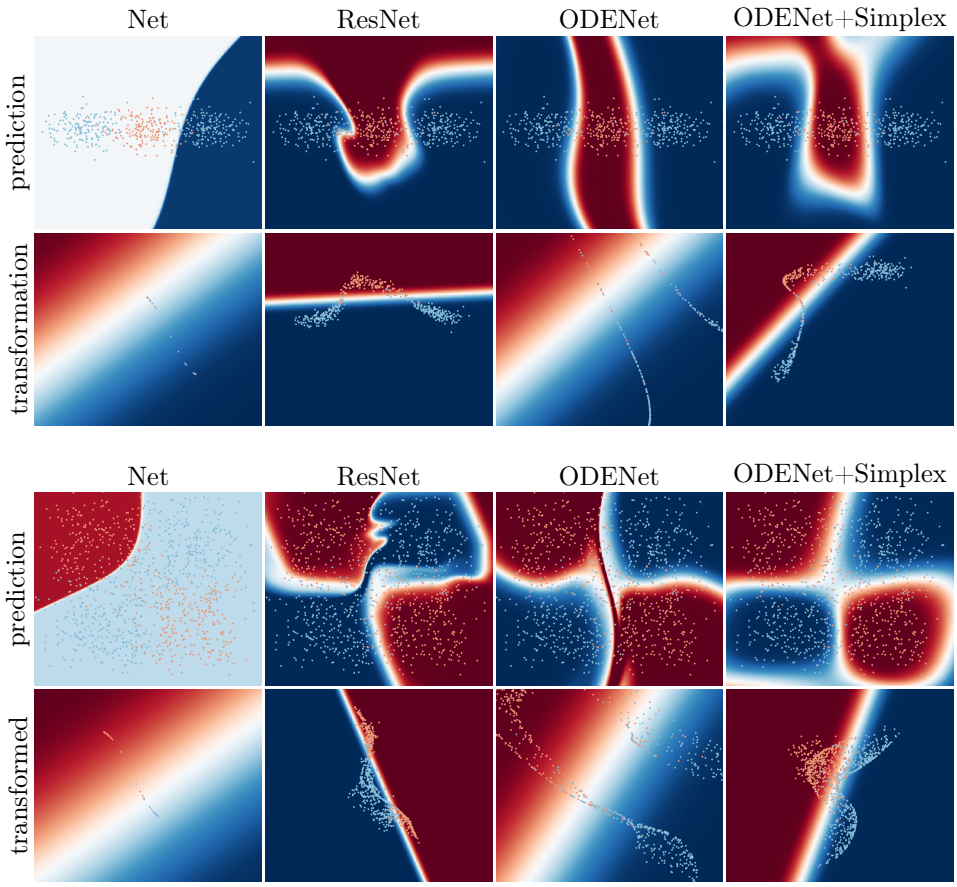

FIGURE 2. Learned transformation and classifier for data set `donut1d` (top) and `squares` (bottom).

derived in the previous section. Finally, the gradients with respect to W and μ of the discrete cost function are required in order to update these parameters with gradient descent, which read as

$$\gamma_i = \Big(\mathcal{C}(Wy_i^{[N]} + \mu) - c_i\Big) \odot \mathcal{C}'(Wy_i^{[N]} + \mu) \tag{41a}$$

$$\partial_W \mathcal{J}(y_i^{[N]}, W, \mu) = \gamma_i y_i^{[N],T}, \tag{41b}$$

$$\partial_\mu \mathcal{J}(y_i^{[N]}, W, \mu) = \gamma_i. \tag{41c}$$

We consider 4 different data sets (`donut1d`, `donut2d`, `squares`, `spiral`) that have different topological properties, which are illustrated in Figure 1. These are samples from a random variable with prescribed probability density functions. We use 500 samples for data set `donut1d` and each 1,000 for the other three data sets. For simplicity we chose not to

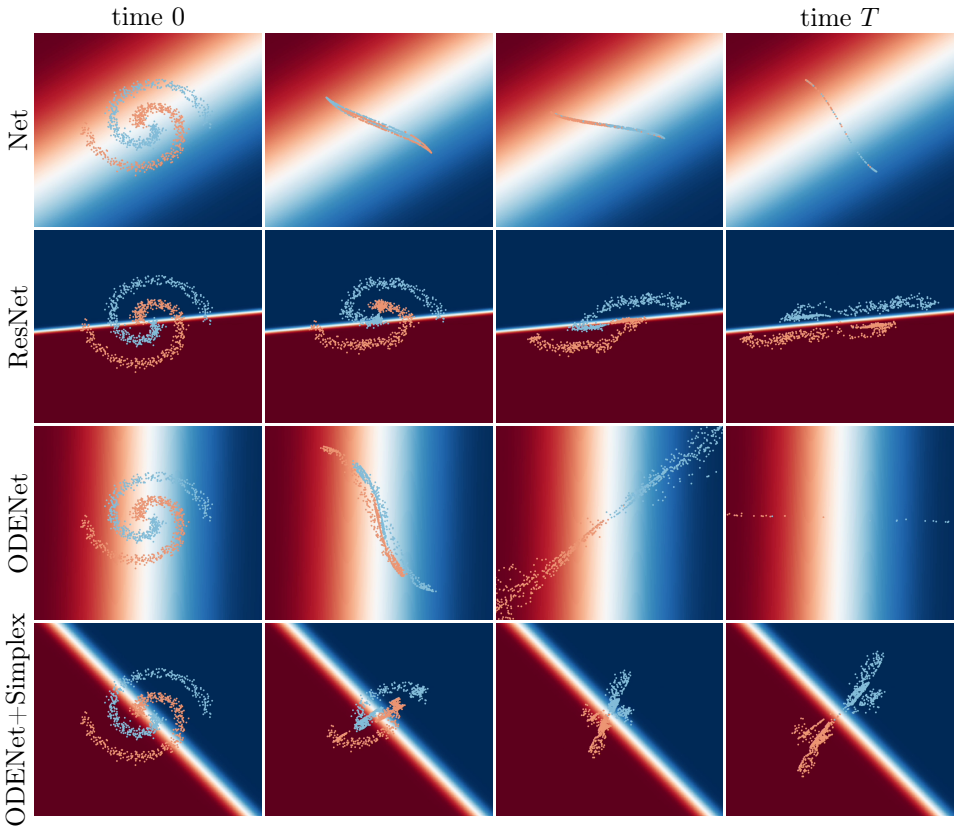

FIGURE 3. Snap shots of transformation of features for data set `spiral`.

use explicit regularisation, i.e. $\mathcal{R} = 0$, in all numerical examples. Code to reproduce the numerical experiments is available on the University of Cambridge repository under https://doi.org/10.17863/CAM.43231.

5.2. **Comparison of Optimal Control Inspired Methods.** We start by comparing qualitative and quantitative properties of four different methods. These are: 1) the standard neural network approach ((4) with (5)), 2) the ResNet ((3) with (5)), 3) the ODENet ((3) with (33)) and 4) the ODENet with simplex constraint (40) on the varying time steps. Throughout this subsection we consider networks with 20 layers.

5.2.1. *Qualitative Comparison.* We start with a qualitative comparison of the prediction performance of the four methods on `donut1d` and `spiral`, see Figure 2. The top rows of both figures show the prediction performance of the learned parameters. The data is plotted as dots in the foreground and the learned classification in the background. A good

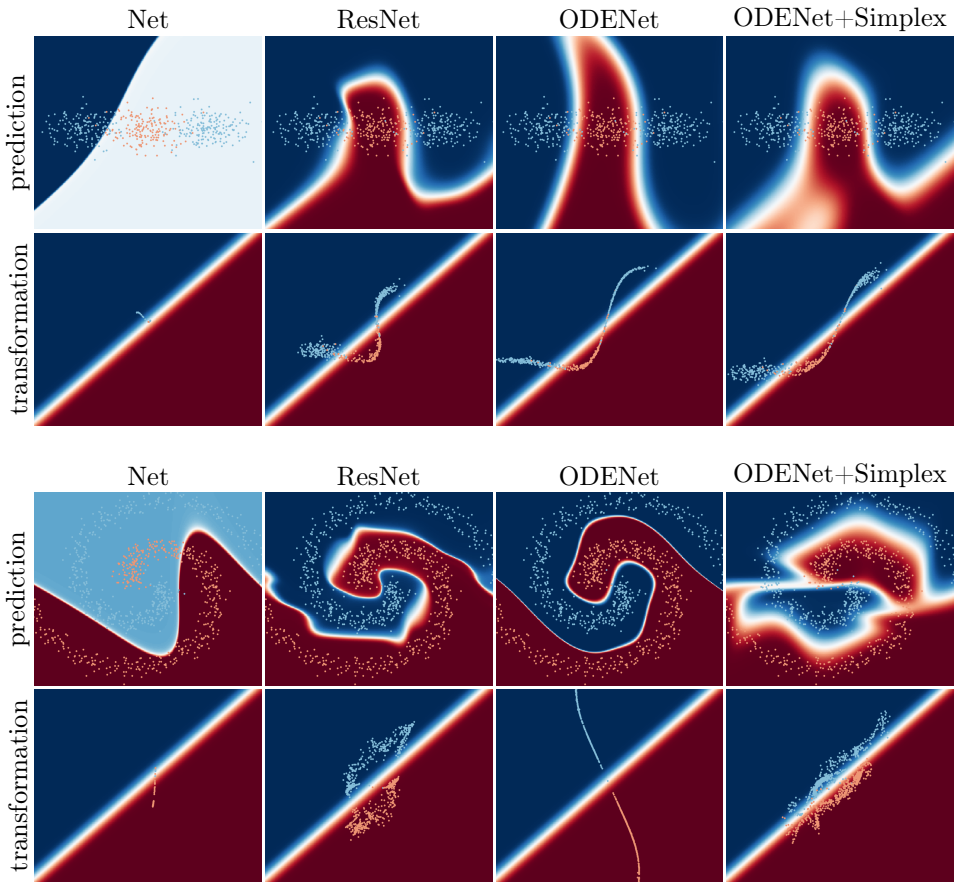

FIGURE 4. Learned transformation with fixed classifier for data set `donut1d` (top) and `spiral` (bottom).

classification has the blue dots in the dark blue areas and similarly for red. We can see that for both data sets Net classifies only a selection of the points correctly whereas the other three methods do rather well on almost all points. Note that the shape of the learned classifier is still rather different despite them being very similar in the area of the training data.

For the bottom rows of both figures we split the classification into the transformation and a linear classification. The transformation is the evolution of the ODE for ResNet, ODENet and ODENet + simplex. For Net this is the recursive formula (4). Note that the learned transformations are very different for the four different methods.

5.2.2. *Evolution of Features.* Figure 3 shows the evolution of the features by the learned parameters for the data set `spiral`. It can be seen that all four methods result in different dynamics, Net and ODENet

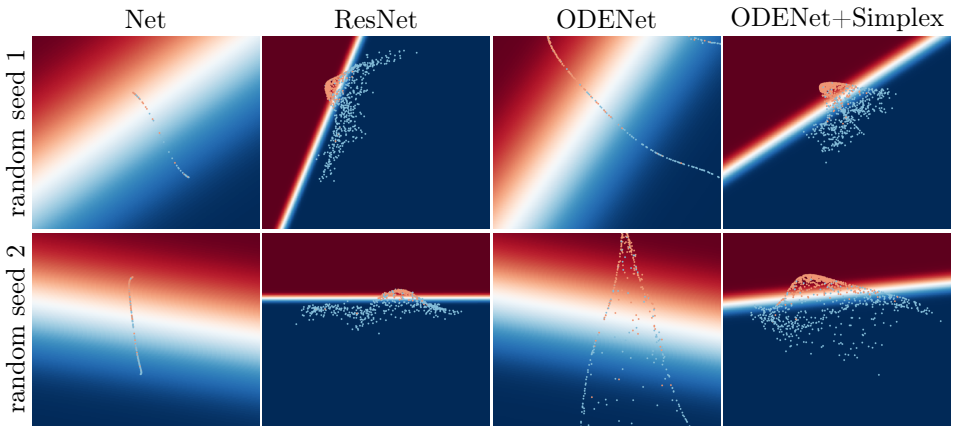

FIGURE 5. Robustness on random initialisation for transformed data `donut2d` and linear classifier for two different initialisations.

reduce the two dimensional point cloud to a one-dimensional string whereas ResNet and ODENet+simplex preserve their two-dimensional character. This observations seem to be characteristic as we qualitatively observed similar dynamics for other data sets and random initialisation (not shown).

Note that the dynamics of ODENet transform the points outside the field-of-view and the decision boundary (fuzzy bright line in the background) is wider than for ResNet and ODENet+simplex.

Intuitively, a scaling of the points and a fuzzier classification is equivalent to leaving the points where they are and a sharper classification. We tested the aforementioned effect by keeping a fixed classification throughout the learning process and only learning the transformation. The results in Figure 4 show that this is indeed the case.

5.2.3. *Dependence on Randomness.* We tested the dependence of our results on different random initialisations. For conciseness we only highlight one result in Figure 5. Indeed, the two rows which correspond to two different random initialisations show very similar topological behaviour.

5.2.4. *Quantitative Results.* Quantitative results are presented in Figures 6 and 7 which show the evolution of function values and the classification accuracy over the course of the gradient descent iterations. The solid lines are for the training data and dashed for the test data, which is an independent draw from the same distribution and of the same size as the training data.

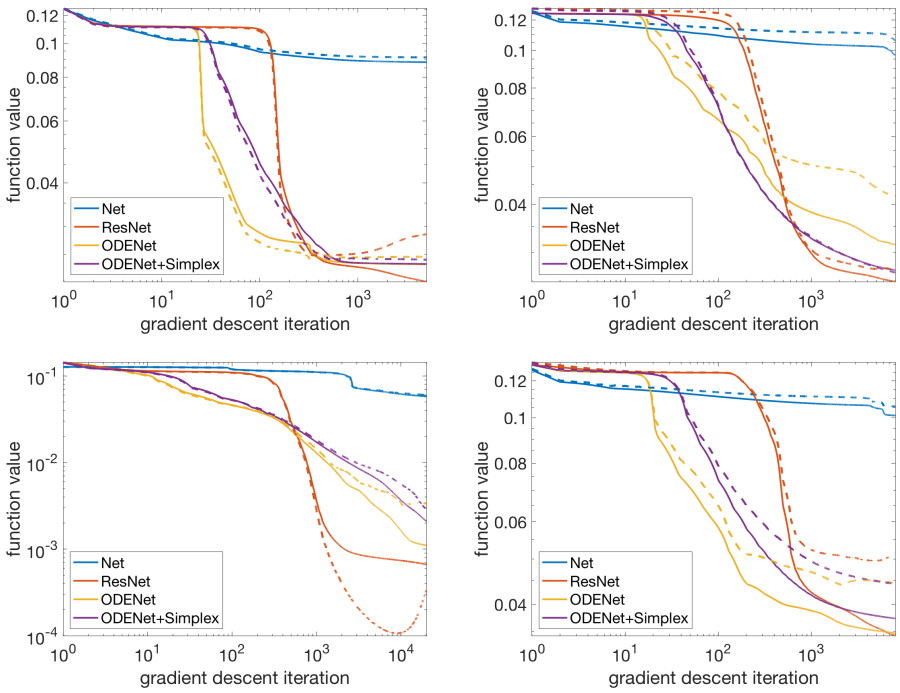

FIGURE 6. Function values over the course of the gradient descent iterations for data sets `donut1d`, `donut2d`, `spiral`, `squares` (left to right and top to bottom). The solid line represents training and the dashed line test data.

We can see that Net does not perform as well for any of the data sets than the other three methods. Consistently, ODENet is initially the fastest but at later stages ResNet overtakes it. All three methods seem to converge to a similar function value. As the dashed line follows essentially the solid line we can observe that there is not much overfitting taking place.

5.2.5. *Estimation of Varying Time Steps.* Figure 8 shows the (estimated) time steps for ResNet/Euler, ODENet and ODENet+simplex. While ResNet uses equidistant time discretisation, ODENet and ODENet+simplex learn these as part of the training. In addition, ODENet+simplex use a simplex constraint on these values which allow the interpretation as varying time steps. It can be seen consistently for all four data sets that ODENet chooses both negative and positive time steps and these are generally of larger magnitude than the other two methods. Moreover, these are all non-zero. In contrast, ODENet+Simplex picks a few time steps (two or three) and sets the

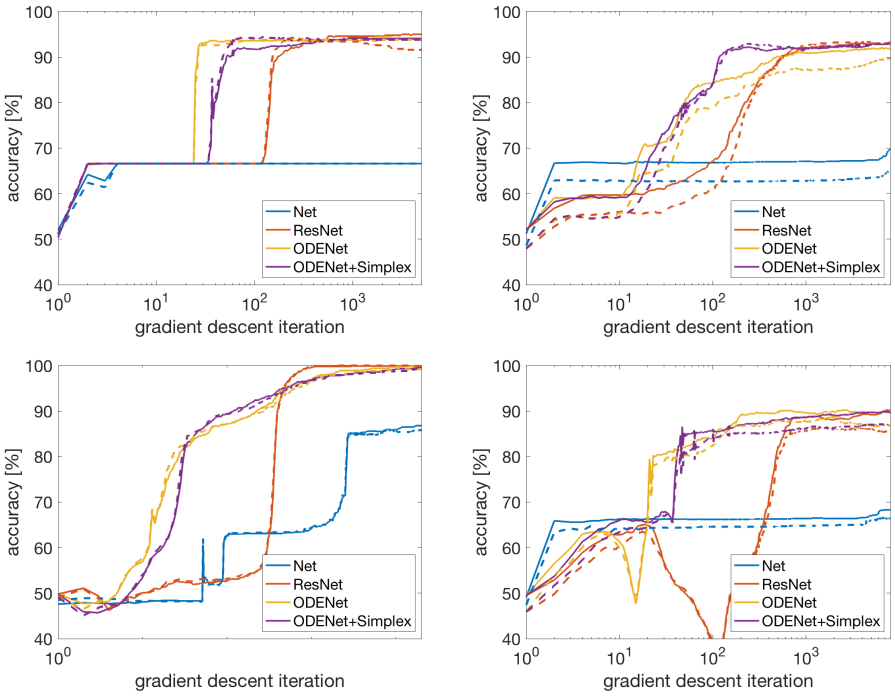

FIGURE 7. Classification accuracy over the course of the gradient descent iterations for data sets `donut1d`, `donut2d`, `spiral`, `squares` (left to right and top to bottom). The solid line represents training and the dashed line test data.

rest to zero. Sparse time steps have the advantage that less memory is needed to store this network and that less computation is needed for classification at test time.

Although it might seem unnatural to allow for negative time steps in this setting, a benefit is that this adds to the flexibility of the approach. It should also be noted that negative steps are rather common in the design of e.g. splitting and composition methods from the ODE literature, [5].

5.3. **Comparing different explicit Runge–Kutta architectures.** We are here showing results for 4 different explicit Runge–Kutta schemes of orders 1–4, their Butcher tableaux are given in Table 1.

The first two methods are the Euler and Improved Euler methods over orders one and two respectively. The other two are due to Kutta [25] and have convergence orders three and four. The presented results are obtained with the data sets `donut1d`, `donut2d`, `spiral`, and `squares`. In the results reported here we have taken the number of

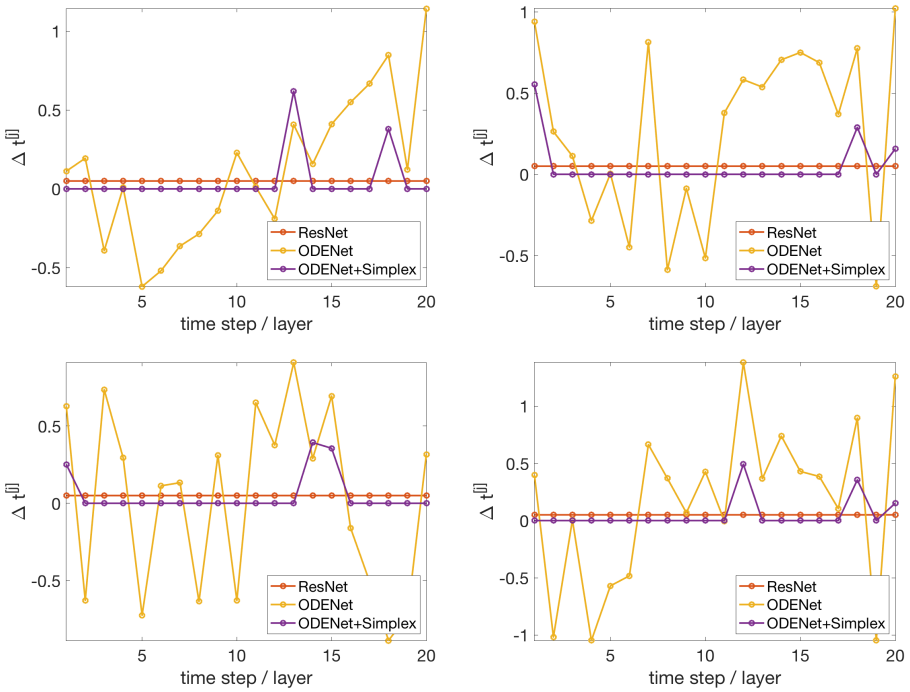

FIGURE 8. Estimated time steps by ResNet/Euler, ODENet and ODENet+simplex for for data sets `donut1d`, `donut2d`, `spiral`, `squares` (left to right and top to bottom). ODENet+simplex consistently picks two to three time steps and set the rest to zero.

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array} \qquad \begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \qquad \begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \qquad \begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

TABLE 1. Four explicit Runge–Kutta methods: ResNet/Euler, Improved Euler, Kutta(3) and Kutta(4).

layers to be 15. In Figure 9 we illustrate the initial and final configurations of the data points for the learned parameters. The blue and red background colours can be thought of as test data results in the upper row of plots. For instance, any point which was originally in a red area will be classified as red with high probability. Similarly, the

background colours in the bottom row of plots show the classification of points which have been transformed to a given location. In the transition between red and blue the classification will have less certainty.

In Figures 10–12, more details of the transition are shown. The leftmost and rightmost plot show the initial and final states respectively, whereas the two in the middle show the transformation in layers 5 and 10. The background colours always show the same and correspond to the final state.

Finally, in Figure 13 we show the progress of the gradient descent method over 10,000 iterations for each of the four data sets.

5.4. **Digit classification with minimal data.** We test four network architectures—three of which are ODE-inspired—on digit classification. The training data is selected from the MNIST data base [28] where we restrict ourselves to classifying 0s and 8s. To make this classification more challenging, we train only on 100 images and take another 500 as test data. We refer to this data as `MNIST100`.

There are a couple of observations which can be made from the results shown in Figures 14 and 15. First, as can be seen in Figure 14, the results are consistent with the observations made from the toy data in Figure 7: the three ODE-inspired methods seem to perform very well, both on training and test data. Also the trained step sizes show similar profiles as in Figure 8, with ODENet learning negative step sizes and ODENet+Simplex learning very sparse time steps. Second, in Figure 15, we show the transformed test data before the classification. Interestingly, all four methods learn what looks to the human eye as adding noise. Only the ODE-inspired networks retain some of the structure of the input features.

6. **Conclusions and outlook.** In this paper we have investigated the interpretation of deep learning as an optimal control problem. In particular, we have proposed a first-optimise-then-discretise approach for the derivation of ODE-inspired deep neural networks using symplectic partitioned Runge–Kutta methods. The latter discretisation guarantees that also after discretisation the first-order optimality conditions for the optimal control problem are fulfilled. This is in particular interesting under the assumption that the learned ODE discretisation follows some underlying continuous transformation that it approximated. Using partitioned Runge–Kutta methods, we derive several new deep learning algorithms which we compare for their convergence behaviour and the transformation dynamics of the so-learned discrete ODE iterations. Interestingly, while the convergence behaviour for the solution

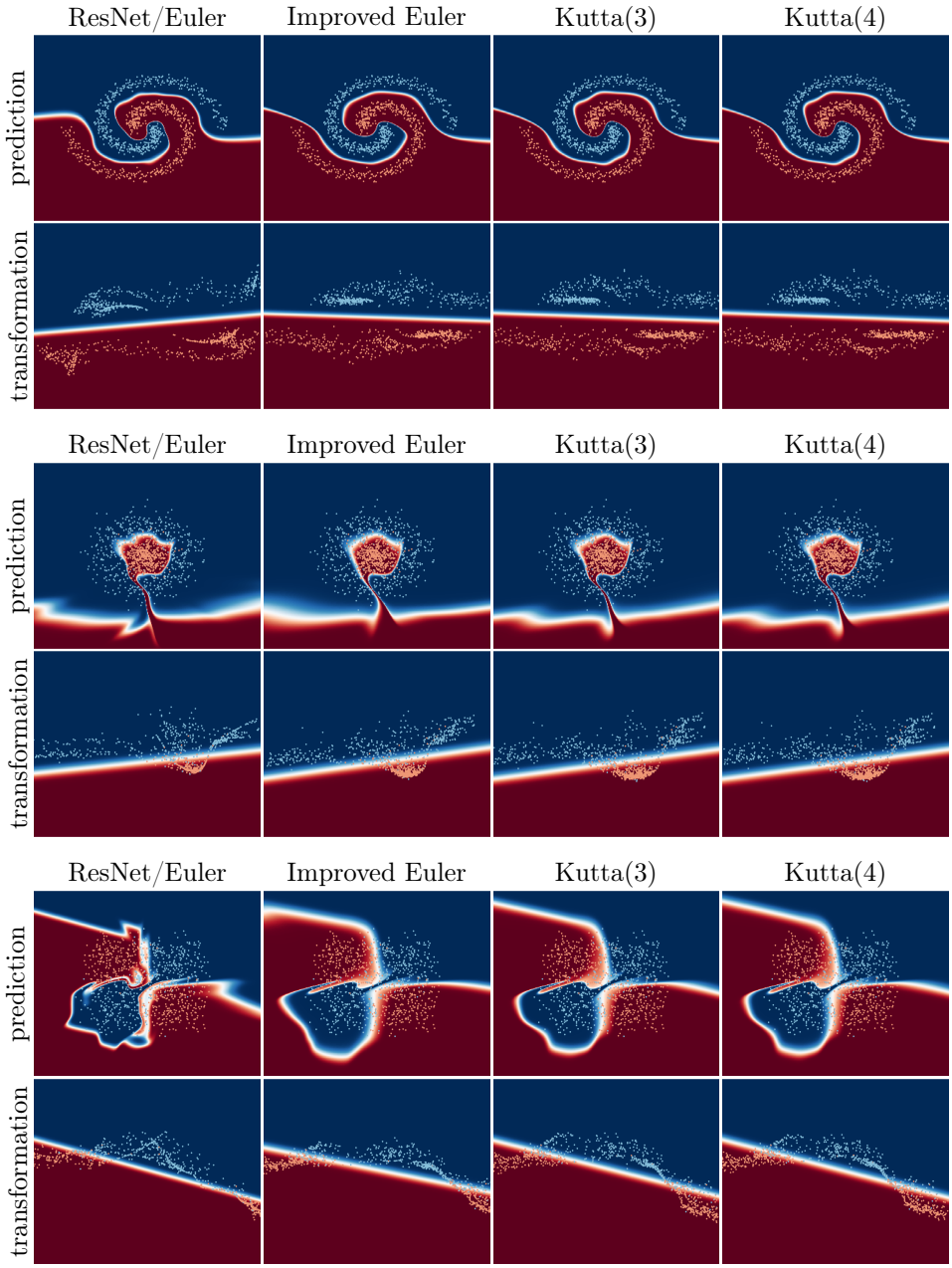

FIGURE 9. Learned prediction and transformation for different Runge–Kutta methods and data sets `spiral` (top), `donut2d` (centre) and `squares` (bottom). All results are for 15 layers.

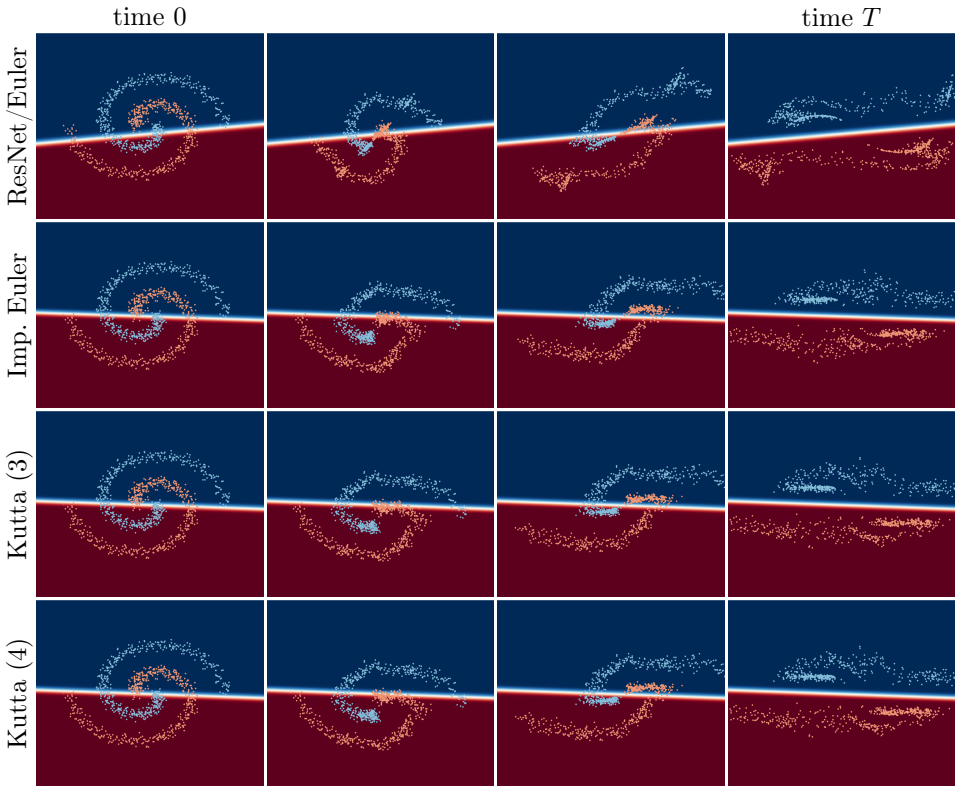

FIGURE 10. Snap shots of the transition from initial to final state through the network with the data set `spiral`.

of the optimal control problem shows differences when trained with different partitioned Runge–Kutta methods, the learned transformation given by the discretised ODE with optimised parameters shows similar characteristics. It is probably too strong of a statement to suggest that our experiments therefore support our hypothesis of an underlying continuous optimal transformation as the similar behaviour could be a consequence of other causes. However, the experiments encourage our hypothesis.

The optimal control formulation naturally lends itself to learning more parameters such as the time discretisation which can be constrained to lie in a simplex. As we have seen in Figure 8, the simplex constraint lead to sparse time steps such that the effectively only very few layers were needed to represent the dynamics, thus these networks have faster online classification performance and lower memory footprint. Another advantage of this approach is that one does not need to know precisely in advance how many layers to choose since the training procedure selects this automatically.

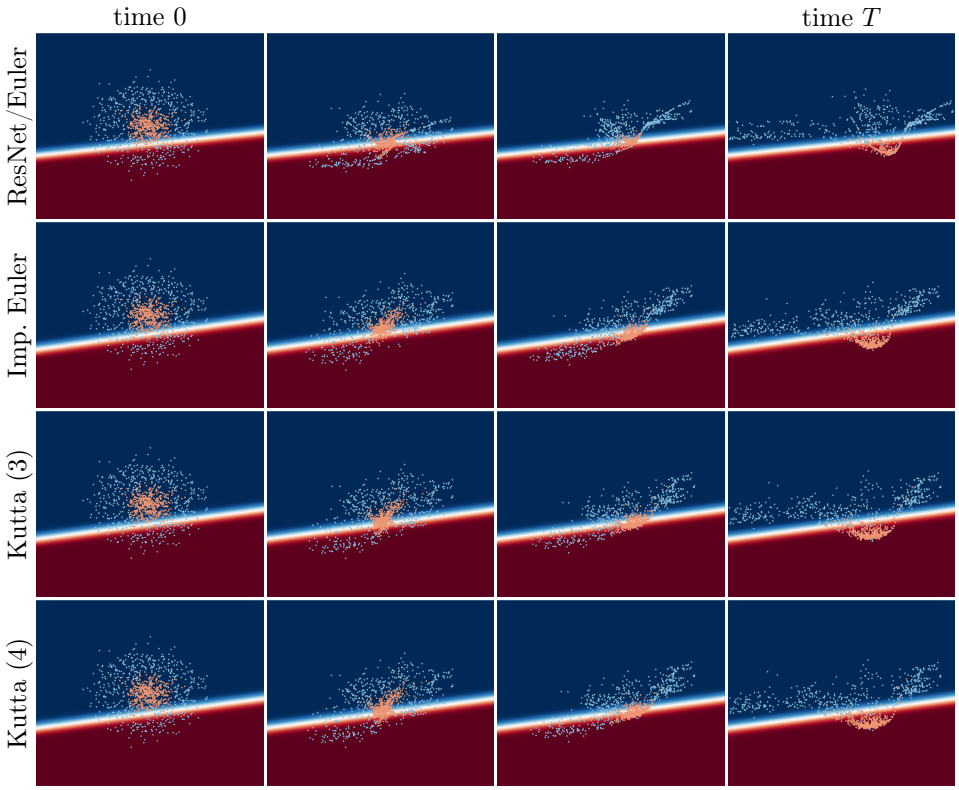

FIGURE 11. Snap shots of the transition from initial to final state through the network with the data set `donut2d`.

An interesting direction for further investigation is to use the optimal control characterisation of deep learning for studying the stability of the problem under perturbations of Y_0. Since the optimal control problem is equivalent to a Hamiltonian boundary value problem, we can study the stability of the first by analysing the second. One can derive conditions on f and $\mathcal{J}$ that ensure existence and stability of the optimal solutions with respect to perturbations on the initial data, or after increasing the number of data points. For the existence of solutions of the optimal control problem and the Pontryagin maximum principle see [6, 14, 2, 42].

The stability of the problem can be analysed in different ways. The first is to investigate how the parameters $u(t) := (K(t), \beta(t))$ change under change (or perturbation) of the initial data and the cost function. The equation for the momenta of the Hamiltonian boundary value problem (adjoint equation) can be used to compute sensitivities of the optimal control problem under perturbation on the initial data [39]. In

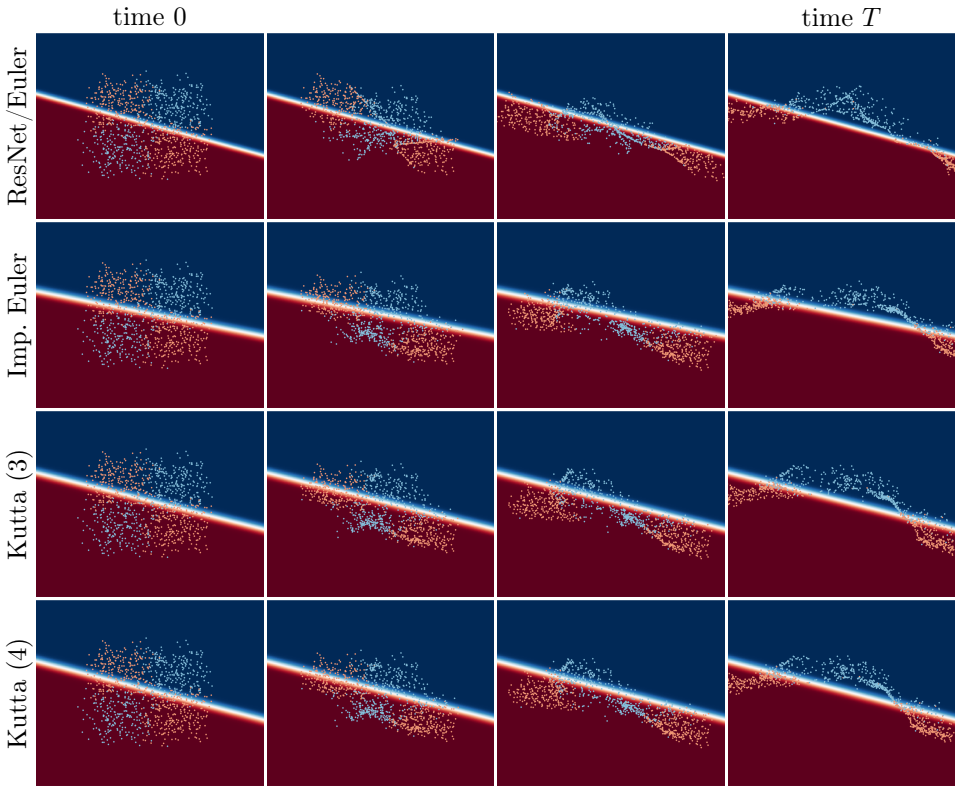

FIGURE 12. Snap shots of the transition from initial to final state through the network with the data set `squares`.

particular, the answer to this is linked to the Hessian of the Hamiltonian with respect to the parameters u. If $H_{u,u}$ is invertible, see Section 2.3, and remains invertible under such perturbations, then $u = \varphi(y, p)$ can be solved for in terms of the state y and the co-state p.

The second is to ask how generalisable the learned parameters u are. The parameters u, i.e. K and β determine the deformation of the data in such a way that the data becomes classifiable with the Euclidean norm at final time T. It would be interesting to show that φ does not change much under perturbation, and neither do the deformations determined by u.

Another interesting direction for future research is the generalisation of the optimal control problem to feature an inverse scale-space ODE as a constraint, where we do not consider the time derivative of the state variable, but of a subgradient of a corresponding convex functional with the state variable as its argument, see for example [40, 8, 7]. Normally these flows are discretised with an explicit or implicit Euler

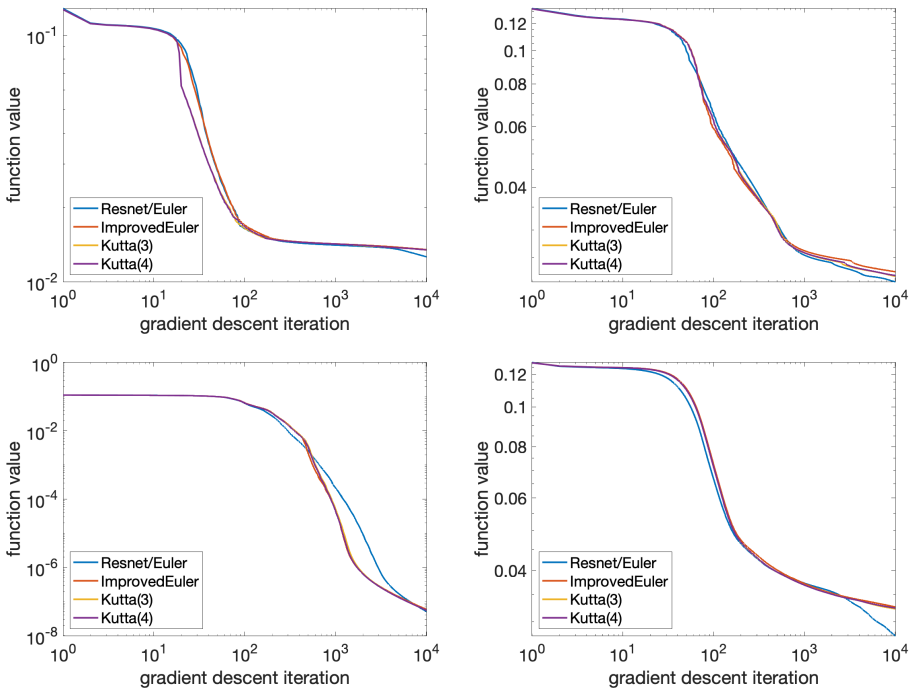

FIGURE 13. Function values over the course of the gradient descent iterations for data sets `donut1d`, `donut2d`, `spiral`, `squares` (left to right and top to bottom).

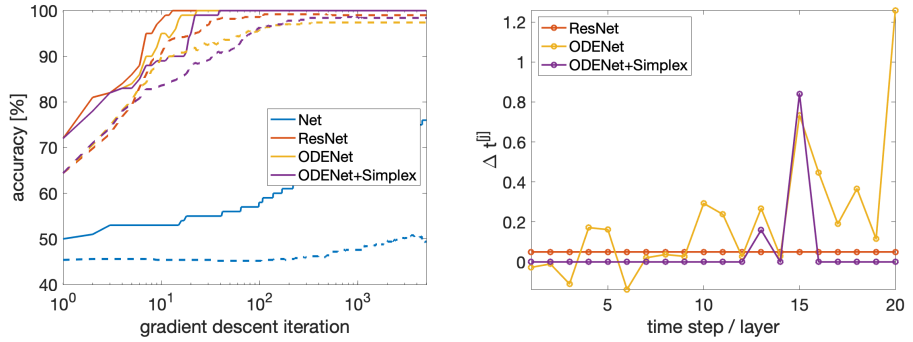

FIGURE 14. Accuracy (left) and time steps (right) for `MNIST100` dataset [28].

scheme. These discretisations can reproduce various neural network architectures [3, Section 9]. Hence, applying the existing knowledge of numerical discretisation methods in a deep learning context may lead to a better and more systematic way of developing new architectures with desirable properties.

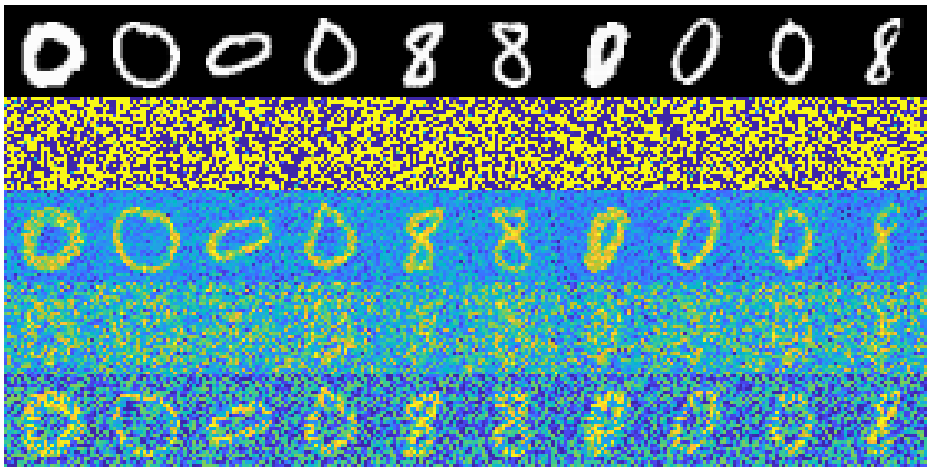

FIGURE 15. Features of testing examples from `MNIST100` dataset [28] and transformed features by four networks under comparison: Net, ResNet, ODENet, ODENet+Simplex (from top to bottom). All networks have 20 layers.

Other questions revolve around the sensitivity in the classification error. How can we estimate the error in the classification once the parameters are learned? Given u obtained solving the optimal control problem, if we change (or update) the set of features, how big is the error in the classification $\mathcal{J}(y^{[N]})$?

Acknowledgments: MB acknowledges support from the Leverhulme Trust Early Career Fellowship ECF-2016-611 'Learning from mistakes: a supervised feedback-loop for imaging applications'. CBS acknowledges support from the Leverhulme Trust project on Breaking the non-convexity barrier, the Philip Leverhulme Prize, the EPSRC grant No. EP/M00483X/1, the EPSRC Centre No. EP/N014588/1, the European Union Horizon 2020 research and innovation programmes under the Marie Skodowska-Curie grant agreement No. 777826 NoMADS and No. 691070 CHiPS, the Cantab Capital Institute for the Mathematics of Information and the Alan Turing Institute. We gratefully acknowledge the support of NVIDIA Corporation with the donation of a Quadro P6000 and a Titan Xp GPU used for this research. 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) Grant number EP/R014604/1 where work on this paper was undertaken. This work was supported by EPSRC grant No. EP/K032208/1.

Appendix A. **Discrete necessary optimality conditions.** We prove Proposition 1 for the general symplectic partitioned Runge–Kutta method.

Proof of Proposition 1. We introduce Lagrangian multipliers $p_i^{[j]}$, $p^{[j+1]}$ and consider the Lagrangian

$$\mathcal{L} = \mathcal{L}\left(\{y^{[j]}\}_{j=1}^{N}, \{y_i^{[j]}\}_{j=1}^{N}, \{u_i^{[j]}\}_{j=0}^{N-1}, \{p^{[j+1]}\}_{j=0}^{N-1}, \{p_i^{[j]}\}_{j=0}^{N-1}\right) \tag{42}$$

$i = 1, \dots, s,$

$$\begin{aligned}\mathcal{L} \;&:=\; \mathcal{J}(y^{[N]}) - \Delta t \sum_{j=0}^{N-1} \langle p^{[j+1]}, \frac{y^{[j+1]} - y^{[j]}}{\Delta t} - \sum_{i=1}^{s} f(y_i^{[j]}, u_i^{[j]})\rangle \\ &- \;\Delta t \sum_{j=0}^{N-1} \Delta t \sum_{i=1}^{s} b_i \langle \ell_i^{[j]}, \frac{y_i^{[j]} - y^{[j]}}{\Delta t} - \sum_{m=1}^{s} a_{i,m} f(y_m^{[j]}, u_m^{[j]})\rangle.\end{aligned}$$

An equivalent formulation of (3) subject to (21)-(23) is

$$\inf_{\substack{\{u_i^{[j]}\}_{j=0}^{N-1},\\ \{y^{[j]}\}_{j=1}^{N}, \{y_i^{[j]}\}_{j=1}^{N},}} \sup_{\{p^{[j]}\}_{j=1}^{N}, \{\ell_i^{[j]}\}_{j=1}^{N}} \mathcal{L}$$

Taking arbitrary and independent variations

$$y^{[j]} + \xi v^{[j]}, \quad y_i^{[j]} + \xi v_i^{[j]}, \quad u_i^{[j]} + \xi w_i^{[j]} \quad p^{[j+1]} + \xi \gamma^{[j+1]}, \quad \ell_i^{[j]} + \xi \gamma_i^{[j]}$$

and imposing $\delta\mathcal{L} = 0$ for all variations, we obtain

$$
\begin{aligned}
0 = \quad \delta\mathcal{L} \quad &= \langle \partial\mathcal{J}(y^{[N]}), v^{[N]} \rangle \\
&- \Delta t \sum_{j=0}^{N-1} \langle \gamma^{[j+1]}, \frac{y^{[j+1]} - y^{[j]}}{\Delta t} - \sum_{i=1}^{s} b_i f(y_i^{[j]}, u_i^{[j]}) \rangle \\
&- \Delta t \sum_{j=0}^{N-1} \langle p^{[j+1]}, \frac{v^{[j+1]} - v^{[j]}}{\Delta t} - \sum_{i=1}^{s} b_i \partial_y f(y_i^{[j]}, u_i^{[j]}) v_i^{[j]} \rangle \\
&+ \Delta t \sum_{j=0}^{N-1} \langle p^{[j+1]}, \sum_{i=1}^{s} b_i \partial_u f(y_i^{[j]}, u_i^{[j]}) w_i^{[j]} \rangle \\
&- \Delta t \sum_{j=0}^{N-1} \Delta t \sum_{i=1}^{s} b_i \langle \gamma_i^{[j+1]}, \frac{y_i^{[j+1]} - y^{[j]}}{\Delta t} - \sum_{m=1}^{s} a_{i,m} f(y_m^{[j]}, u_m^{[j]}) \rangle \\
&- \Delta t \sum_{j=0}^{N-1} \Delta t \sum_{i=1}^{s} b_i \langle \ell_i^{[j]}, \frac{v_i^{[j+1]} - v^{[j]}}{\Delta t} + \sum_{m=1}^{s} a_{i,m} \partial_y f(y_m^{[j]}, u_m^{[j]}) v_m^{[j]} \rangle \\
&- \Delta t \sum_{j=0}^{N-1} \Delta t \sum_{i=1}^{s} b_i \langle \ell_i^{[j]}, \sum_{m=1}^{s} a_{i,m} \partial_u f(y_m^{[j]}, u_m^{[j]}) w_m^{[j]} \rangle .
\end{aligned}
$$

Because the variations $\gamma^{[j]}$, $\gamma_i^{[j]}$ are arbitrary, we must have

$$
\begin{aligned}
\frac{y^{[j+1]} - y^{[j]}}{\Delta t} &= \sum_{i=1}^{s} b_i f(y_i^{[j]}, u_i^{[j]}) \\
\frac{y_i^{[j+1]} - y^{[j]}}{\Delta t} &= \sum_{m=1}^{s} a_{i,m} f(y_m^{[j]}, u_m^{[j]})
\end{aligned}
$$

corresponding to the forward method, (16a), (16b), and we are left with terms depending on $w_i^{[j]}$ and $v_i^{[j]}$ which we can discuss separately. Collecting all the terms containing the variations $w_i^{[j]}$ we get

$$
\begin{aligned}
\sum_{j=0}^{N-1} \Bigg(& \sum_{i=1}^{s} b_i \langle p^{[j+1]}, \partial_u f(y_i^{[j]}, u_i^{[j]}) w_i^{[j]} \rangle \\
& - \Delta t \sum_{i=1}^{s} b_i \sum_{m=1}^{s} a_{i,m} \langle \ell_i^{[j]}, \partial_u f(y_m^{[j]}, u_m^{[j]}) w_m^{[j]} \rangle \Bigg) .
\end{aligned} \tag{43}
$$

In (43), renaming the indexes so that $i \to k$ in the first sum and $m \to k$ and $w_m^{[j]} \to w_k^{[j]}$ in the second sum, we get

$$\sum_{k=1}^{s} (b_k \langle p^{[j+1]}, \partial_u f(y_k^{[j]}, u_k^{[j]}) w_k^{[j]} \rangle - \Delta t \sum_{i=1}^{s} b_i a_{i,k} \langle \ell_i^{[j]}, \partial_u f(y_k^{[j]}, u_k^{[j]}) w_k^{[j]} \rangle),$$

for $j = 0, \ldots, N-1$, and

$$\sum_{k=1}^{s} \left(\langle b_k \partial_u f(y_k^{[j]}, u_k^{[j]})^T p^{[j+1]} - \Delta t \sum_{i=1}^{s} b_i a_{i,k} \partial_u f(y_k^{[j]}, u_k^{[j]})^T \ell_i^{[j]}, w_k^{[j]} \rangle \right).$$

Because each of the variations $w_k^{[j]}$ is arbitrary for $k = 1, \ldots, s$ and $j = 0, \ldots, N-1$ each of the terms must vanish and we get

$$\langle \partial_u f(y_k^{[j]}, u_k^{[j]})^T p^{[j+1]} - \Delta t \sum_{i=1}^{s} \frac{b_i a_{i,k}}{b_k} \partial_u f(y_k^{[j]}, u_k^{[j]})^T \ell_i^{[j]}, w_k^{[j]} \rangle = 0,$$

and finally

$$\partial_u f(y_k^{[j]}, u_k^{[j]})^T \left(p^{[j+1]} - \Delta t \sum_{i=1}^{s} \frac{b_i a_{i,k}}{b_k} \ell_i^{[j]} \right) = 0$$

corresponding to the discretised constraints, and where we recognise that

$$p_k^{[j]} = p^{[j+1]} - \Delta t \sum_{i=1}^{s} \frac{b_i a_{i,k}}{b_k} \ell_i^{[j]}.$$

The remaining terms contain the variations $v_i^{[j]}$ and we have

$$\begin{aligned} &\langle \partial \mathcal{J}(y^{[N]}), v^{[N]} \rangle \\ &\quad - \Delta t \sum_{j=0}^{N-1} \langle p^{[j+1]}, \frac{v^{[j+1]} - v^{[j]}}{\Delta t} - \sum_{i=1}^{s} b_i \partial_y f(y_i^{[j]}, u_i^{[j]}) v_i^{[j]} \rangle \\ &\qquad - \Delta t^2 \sum_{j=0}^{N-1} \sum_{i=1}^{s} b_i \langle \ell_i^{[j]}, \frac{v_i^{[j+1]} - v^{[j]}}{\Delta t} + \sum_{m=1}^{s} a_{i,m} \partial_y f(y_m^{[j]}, u_m^{[j]}) v_m^{[j]} \rangle = 0 \end{aligned}$$

There are only two terms involving v_N, leading to

$$\mathcal{J}(y^{[N]}), v^{[N]} \rangle - \langle p^{[N]}, v_N \rangle = 0$$

corresponding to the condition $p^{[N]} = \mathcal{J}(y^{[N]})$. We consider separately for each j terms involving $v^{[j]}$ and $V_i^{[j]}$ for $i = 1, \ldots, s$ and see that

$$\begin{aligned} & \langle p^{[j+1]}, v^{[j]} + \Delta t \sum_{i=1}^{s} b_i \partial_y f(y_i^{[j]}, u_i^{[j]}) v_i^{[j]} \rangle \\ - \quad & \Delta t \sum_{i=1}^{s} b_i \langle \ell_i^{[j]}, v_i^{[j]} - v^{[j]} + \Delta t \sum_{m=1}^{s} a_{i,m} \partial_y f(y_m^{[j]}, u_m^{[j]}) v_m^{[j]} \\ - \quad & \langle p^{[j]}, v^{[j]} \rangle = 0 \end{aligned}$$

which we rearrange into

$$\begin{aligned} & \langle p^{[j+1]} - p^{[j]} + h \sum_{i=1}^{s} b_i \ell_i^{[j]}, v^{[j]} \rangle \\ & \Delta t \sum_{k=1}^{s} b_k \langle \partial_y f(y_k^{[j]}, u_k^{[k]})^T p^{[j+1]}, v_k^{[j]} \rangle \\ - \quad & \Delta t \sum_{i=1}^{s} b_i \left(\langle \ell_i^{[j]}, v_i^{[j]} \rangle + \Delta t \sum_{m=1}^{s} a_{i,m} \langle \partial_y f(y_m^{[j]}, u_m^{[j]}) \ell_i^{[j]}, v_m^{[j]} \rangle \right) = 0 \end{aligned}$$

This yields

$$p^{[j+1]} = p^{[j]} - \Delta t \sum_{i=1}^{s} b_i \ell_i^{[j]}$$

and

$$\begin{aligned} & \Delta t \sum_{k=1}^{s} b_k \langle \partial_y f(y_k^{[j]}, u_k^{[k]})^T p^{[j+1]}, v_k^{[j]} \rangle \\ - \quad & \Delta t \sum_{i=1}^{s} b_i \left(\langle \ell_i^{[j]}, v_i^{[j]} \rangle + \Delta t \sum_{m=1}^{s} a_{i,m} \langle \partial_y f(u_m^{[j]}, u_m^{[j]}) \ell_i^{[j]}, v_m^{[j]} \rangle \right) = 0. \end{aligned}$$

From the last equation we get

$$0 = \partial_y f(y_k^{[j]}, u_k^{[k]})^T p^{[j+1]} - \ell_k^{[j]} - \Delta t \sum_{i=1}^{s} \frac{b_i a_{i,k}}{b_k} \partial_y f(y_k^{[j]}, u_k^{[j]})^T \ell_i^{[j]}.$$

with

$$\ell_k^{[j]} = \partial_y f(y_k^{[j]}, u_k^{[j]})^T \left[p^{[j+1]} - \Delta t \sum_{i=1}^{s} \frac{b_i a_{i,k}}{b_k} \ell_i^{[j]} \right].$$

□

REFERENCES

[1] H. D. Abarbanel, P. J. Rozdeba and S. Shirman, Machine learning: Deepest learning as statistical data assimilation problems, *Neural computation*, **30** (2018), 2025–2055.

[2] A. A. Agrachev and Y. Sachkov, *Control theory from the geometric viewpoint*, vol. 87, Springer Science & Business Media, 2013.

[3] M. Benning and M. Burger, Modern regularization methods for inverse problems, *Acta Numerica*, **27** (2018), 1–111.

[4] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[5] S. Blanes and F. Casas, On the necessity of negative coefficients for operator splitting schemes of order higher than two, *Applied Numerical Mathematics*, **54** (2005), 23–37, URL http://dx.doi.org/10.1016/j.apnum.2004.10.005.

[6] R. Bucy, Two-point boundary value problems of linear hamiltonian systems, *SIAM Journal on Applied Mathematics*, **15** (1967), 1385–1389.

[7] M. Burger, G. Gilboa, S. Osher, J. Xu et al., Nonlinear inverse scale space methods, *Communications in Mathematical Sciences*, **4** (2006), 179–212.

[8] M. Burger, S. Osher, J. Xu and G. Gilboa, Nonlinear inverse scale space methods for image restoration, in *International Workshop on Variational, Geometric, and Level Set Methods in Computer Vision*, Springer, 2005, 25–36.

[9] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert and E. Holtham, Reversible architectures for arbitrarily deep residual neural networks, in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[10] T. Q. Chen, Y. Rubanova, J. Bettencourt and D. K. Duvenaud, Neural ordinary differential equations, in *Advances in Neural Information Processing Systems*, 2018, 6572–6583.

[11] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli and Y. Bengio, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, in *Advances in neural information processing systems*, 2014, 2933–2941.

[12] J. Duchi, S. Shalev-Shwartz, Y. Singer and T. Chandra, Efficient Projections onto the L1 -ball for Learning in High dimensions, in *Proceedings of the 25th International Conference on Machine Learning*, 2008, 272–279.

[13] W. E, A proposal on machine learning via dynamical systems, *Communications in Mathematics and Statistics*, **5** (2017), 1–11.

[14] F. Gay-Balmaz and T. S. Ratiu, Clebsch optimal control formulation in mechanics, *J. Geom. Mech*, **3** (2011), 41–79.

[15] A. Gholami, K. Keutzer and G. Biros, Anode: Unconditionally accurate memory-efficient gradients for neural odes., 2019.

[16] I. Goodfellow, J. Shlens and C. Szegedy, Explaining and harnessing adversarial examples, 2014.

[17] E. Haber and L. Ruthotto, Stable architectures for deep neural networks, *Inverse Problems*, **34** (2017), 014004.

[18] W. W. Hager, Runge–Kutta methods in optimal control and the transformed adjoint system, *Numerische Mathematik*, **87** (2000), 247–282.

[19] E. Hairer, C. Lubich and G. Wanner, *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, vol. 31, Springer Science & Business Media, 2006.

[20] K. He, X. Zhang, S. Ren and J. Sun, Identity mappings in deep residual networks, in *European Conference on Computer Vision*, 2016, 630–645.
[21] K. He, X. Zhang, S. Ren and J. Sun, Deep Residual Learning for Image Recognition, in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, 770–778.
[22] C. F. Higham and D. J. Higham, Deep Learning: An Introduction for Applied Mathematicians, 2018.
[23] M. Igami, Artificial intelligence as structural estimation: Economic interpretations of Deep Blue, Bonanza, and AlphaGo, 2017.
[24] A. Kurakin, I. Goodfellow and S. Bengio, Adversarial examples in the physical world, 2016.
[25] W. Kutta, Beitrag zur näherungsweisen integration totaler differentialgleichungen, *Z. Math. Phys.*, **46** (1901), 435–453.
[26] D. A. L and W. W. Hager, The euler approximation in state constrained optimal control, *Mathematics of Computation*, **70** (2000), 173–203.
[27] Y. LeCun, Y. Bengio and G. Hinton, Deep learning, *Nature*, **521** (2015), 436–444.
[28] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, Gradient-Based Learning Applied to Document Recognition, in *Proceedings of the IEEE*, vol. 86, 1998, 2278–2324.
[29] Y. LeCun, A theoretical framework for back-propagation, in *Proceedings of the 1988 connectionist models summer school*, vol. 1, CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988, 21–28.
[30] Q. Li, L. Chen, C. Tai and E. Weinan, Maximum principle based algorithms for deep learning, *The Journal of Machine Learning Research*, **18** (2017), 5998–6026.
[31] Q. Li and S. Hao, An Optimal Control Approach to Deep Learning and Applications to Discrete-Weight Neural Networks, 2018.
[32] Y. Lu, A. Zhong, Q. Li and B. Dong, Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations, in *Proceedings of the 35th International Conference on Machine Learning* (eds. J. Dy and A. Krause), vol. 80 of Proceedings of Machine Learning Research, PMLR, Stockholmsmässan, Stockholm Sweden, 2018, 3276–3285, URL http://proceedings.mlr.press/v80/lu18d.html.
[33] S.-M. Moosavi-Dezfooli, A. Fawzi and P. Frossard, Deepfool: a simple and accurate method to fool deep neural networks, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, 2574–2582.
[34] D. L. Phillips, A technique for the numerical solution of certain integral equations of the first kind, *Journal of the ACM (JACM)*, **9** (1962), 84–97.
[35] D. C. Plaut et al., Experiments on learning by back propagation.
[36] L. Pontryagin, *L.S. Pontryagin selected works, The Mathematical theory of optimal processes, Classics of Soviet Mathematics*, vol. 4, CRC Press., 1987.
[37] I. M. Ross, A roadmap for optimal control: the right way to commute, *Annals of the New York Academy of Sciences*, **1065** (2005), 210–231.
[38] F. Santosa and W. W. Symes, Linear inversion of band-limited reflection seismograms, *SIAM Journal on Scientific and Statistical Computing*, **7** (1986), 1307–1330.

[39] J. M. Sanz-Serna, Symplectic Runge-Kutta schemes for adjoint equations automatic differentiation, optimal control and more, *SIAM Review*, **58** (2015), 3–33.
[40] O. Scherzer and C. Groetsch, Inverse scale space theory for inverse problems, in *International Conference on Scale-Space Theories in Computer Vision*, Springer, 2001, 317–325.
[41] S. Sonoda and N. Murata, Double continuum limit of deep neural networks, in *ICML Workshop Principled Approaches to Deep Learning*, 2017.
[42] E. D. Sontag, *Mathematical control theory: deterministic finite dimensional systems*, vol. 6, Springer Science & Business Media, 2013.
[43] I. Sutskever, J. Martens, G. E. Dahl and G. E. Hinton, On the importance of initialization and momentum in deep learning., *ICML (3)*, **28** (2013), 5.
[44] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow and R. Fergus, Intriguing properties of neural networks, in *International Conference on Learning Representations*, 2014.
[45] M. Thorpe and Y. van Gennip, Deep Limits of Residual Neural Networks, 2018.
[46] R. Tibshirani, Regression shrinkage and selection via the lasso, *Journal of the Royal Statistical Society: Series B (Methodological)*, **58** (1996), 267–288.
[47] A. N. Tikhonov, Solution of incorrectly formulated problems and the regularization method, in *Dokl. Akad. Nauk.*, vol. 151, 1963, 1035–1038.
[48] E. Weinan, J. Han and Q. Li, A Mean-Field Optimal Control Formulation of Deep Learning, 2018.

E-mail address: m.benning@qmul.ac.uk
E-mail address: elena.celledoni@ntnu.no
E-mail address: m.ehrhardt@bath.ac.uk
E-mail address: brynjulf.owren@ntnu.no
E-mail address: cbs31@cam.ac.uk