CONTINUOUS-TIME META-LEARNING WITH FORWARD MODE DIFFERENTIATION

Anonymous authors
Paper under double-blind review

ABSTRACT

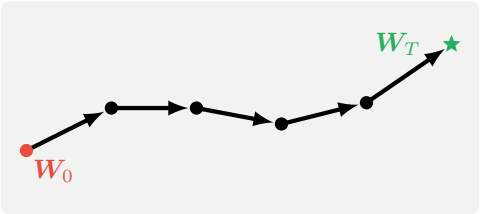
Drawing inspiration from gradient-based meta-learning methods with infinitely small gradient steps, we introduce Continuous-Time Meta-Learning (COMLN), a meta-learning algorithm where adaptation follows the dynamics of a gradient vector field. Specifically, representations of the inputs are meta-learned such that a task-specific linear classifier is obtained as a solution of an ordinary differential equation (ODE). Treating the learning process as an ODE offers the notable advantage that the length of the trajectory is now continuous, as opposed to a fixed and discrete number of gradient steps. As a consequence, we can optimize the amount of adaptation necessary to solve a new task using stochastic gradient descent, in addition to learning the initial conditions as is standard practice in gradient-based meta-learning. Importantly, in order to compute the exact meta-gradients required for the outer-loop updates, we devise an efficient algorithm based on forward mode differentiation, whose memory requirements do not scale with the length of the learning trajectory, thus allowing longer adaptation in constant memory. We provide analytical guarantees for the stability of COMLN, we show empirically its efficiency in terms of runtime and memory usage, and we illustrate its effectiveness on a range of few-shot image classification problems.

1 INTRODUCTION

Among the existing meta-learning algorithms, gradient-based methods as popularized by Model-Agnostic Meta-Learning (MAML, Finn et al., 2017) have received a lot of attention over the past few years. They formulate the problem of learning a new task as an inner optimization problem, typically based on a few steps of gradient descent. An outer *meta*-optimization problem is then responsible for updating the meta-parameters of this learning process, such as the initialization of the gradient descent procedure. However since the updates at the outer level typically require backpropagating through the learning process, this class of methods has often been limited to only a few gradient steps of adaptation, due to memory constraints. Although solutions have been proposed to alleviate the memory requirements of these algorithms, including checkpointing (Baranchuk, 2019), using implicit differentiation (Rajeswaran et al., 2019), or reformulating the meta-learning objective (Flennerhag et al., 2018), they are generally either more computationally demanding, or only approximate the gradients of the meta-learning objective (Nichol et al., 2018; Flennerhag et al., 2020).

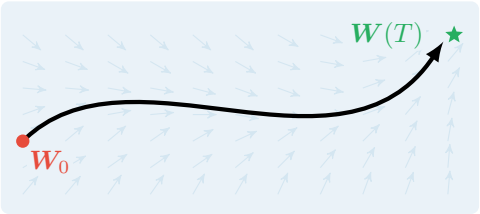
In this work, we propose a continuous-time formulation of gradient-based meta-learning, called *Continuous-Time Meta-Learning* (COMLN), where the adaptation is the solution of a differential equation (see Figure 1). Moving to continuous time allows us to devise a novel algorithm, based on forward mode differentiation, to efficiently compute the exact gradients for meta-optimization, no matter how long the adaptation to a new task might be. We show that using forward mode differentiation leads to a stable algorithm, unlike the counterpart of backpropagation in continuous time called the adjoint method (frequently used in the Neural ODE literature) which tends to be unstable with gradient vector fields. Moreover as the length of the adaptation trajectory is a continuous quantity, as opposed to a discrete number of gradient steps fixed ahead of time, we can treat the amount of adaptation in COMLN as a meta-parameter—on par with the initialization—which we can meta-optimize using stochastic gradient descent. We verify empirically that our method is both computationally and memory efficient, and we show that COMLN outperforms other standard meta-learning algorithms on few-shot image classification datasets.

(a) Gradient-based Meta-Learning



$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \alpha \nabla \mathcal{L}(\boldsymbol{W}_t)$$

(b) COMLN



$$\frac{d\boldsymbol{W}}{dt} = -\nabla \mathcal{L}\big(\boldsymbol{W}(t)\big)$$

Figure 1: Illustration of the adaptation process in (a) a gradient-based meta-learning algorithm, such as ANIL (Raghu et al., 2019), where the adapted parameters $\boldsymbol{W}_T$ are given after T steps of gradient descent, and in (b) Continuous-Time Meta-Learning (COMLN), where the adapted parameters $\boldsymbol{W}(T)$ are the result of following the dynamics of a gradient vector field up to time T.

2 Background

In this work, we consider the problem of few-shot classification, that is the problem of learning a classification model with only a small number of training examples. More precisely for a classification task τ, we assume that we have access to a (small) training dataset $\mathcal{D}_\tau^{\text{train}} = \{(\boldsymbol{x}_m, \boldsymbol{y}_m)\}_{m=1}^M$ to fit a model on task τ, and a distinct test dataset $\mathcal{D}_\tau^{\text{test}}$ to evaluate how well this adapted model generalizes on that task. In the few-shot learning literature, it is standard to consider the problem of k-shot N-way classification, meaning that the model has to classify among N possible classes, and there are only k examples of each class in $\mathcal{D}_\tau^{\text{train}}$, so that overall the number of training examples is $M = kN$. We use the convention that the target labels $\boldsymbol{y}_m \in \{0,1\}^N$ are one-hot vectors.

2.1 Gradient-based meta-learning

Gradient-based meta-learning methods aim to learn an initialization such that the model is able to adapt to a new task via gradient descent. Such methods are often cast as a bi-level optimization process: adapting the task-specific parameters $\boldsymbol{\theta}$ in the inner loop, and training the (task-agnostic) meta-parameters Φ and initialization $\boldsymbol{\theta}_0$ in the outer loop. The meta-learning objective is:

$$\min_{\boldsymbol{\theta}_0, \Phi} \; \mathbb{E}_\tau \big[\mathcal{L}(\boldsymbol{\theta}_T^\tau, \Phi; \mathcal{D}_\tau^{\text{test}}) \big] \tag{1}$$

$$\text{s.t.} \;\; \boldsymbol{\theta}_{t+1}^\tau = \boldsymbol{\theta}_t^\tau - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t^\tau, \Phi; \mathcal{D}_\tau^{\text{train}}) \qquad \boldsymbol{\theta}_0^\tau = \boldsymbol{\theta}_0 \qquad \forall \tau \sim p(\tau), \tag{2}$$

where T is the number of inner loop updates. For example, in the case of MAML (Finn et al., 2017), there is no additional meta-parameter other than the initialization ($\Phi \equiv \emptyset$); in ANIL (Raghu et al., 2019), $\boldsymbol{\theta}$ are the parameters of the last layer, and Φ are the parameters of the shared embedding network; in CAVIA (Zintgraf et al., 2019), $\boldsymbol{\theta}$ are referred to as context parameters.

During meta-training, the model is trained over many tasks τ. The task-specific parameters $\boldsymbol{\theta}$ are learned via gradient descent on $\mathcal{D}_\tau^{\text{train}}$. The meta-parameters are then updated by evaluating the error of the trained model on the test dataset $\mathcal{D}_\tau^{\text{test}}$. At meta-testing time, the meta-trained model is adapted on $\mathcal{D}_\tau^{\text{train}}$—i.e. applying Equation 2 with the learned meta-parameters $\boldsymbol{\theta}_0$ and Φ.

2.2 Local sensitivity analysis of Ordinary Differential Equations

Consider the following (autonomous) Ordinary Differential Equation (ODE):

$$\frac{d\boldsymbol{z}}{dt} = g\big(\boldsymbol{z}(t); \boldsymbol{\theta}\big) \qquad \boldsymbol{z}(0) = \boldsymbol{z}_0(\boldsymbol{\theta}), \tag{3}$$

where the dynamics g and initial value $\boldsymbol{z}_0$ may depend on some external parameters $\boldsymbol{\theta}$, and integration is carried out from 0 to some time T. *Local sensitivity analysis* is the study of how the solution of this dynamical system responds to local changes in $\boldsymbol{\theta}$; this effectively corresponds to calculating the derivative $d\boldsymbol{z}(t)/d\boldsymbol{\theta}$. We present here two methods to compute this derivative, with a special focus on their memory efficiency.

Adjoint sensitivity method Based on the adjoint state (Pontryagin, 2018), and taking its root in control theory (Lions & Magenes, 2012), the *adjoint sensitivity method* (Bryson & Ho, 1969; Chavent et al., 1974) provides an efficient approach for evaluating derivatives of $\mathcal{L}\big(\boldsymbol{z}(T);\boldsymbol{\theta}\big)$, a function of $\boldsymbol{z}(T)$ the solution of the ODE in Equation 3. This method, popularized lately by the literature on Neural ODEs (Chen et al., 2018), requires the integration of the adjoint equation

$$\frac{d\boldsymbol{a}}{dt} = -\boldsymbol{a}(t)\frac{\partial g\big(\boldsymbol{z}(t);\boldsymbol{\theta}\big)}{\partial \boldsymbol{z}(t)} \qquad\qquad \boldsymbol{a}(T) = \frac{d\mathcal{L}\big(\boldsymbol{z}(T);\boldsymbol{\theta}\big)}{d\boldsymbol{z}(T)}, \tag{4}$$

backward in time. The adjoint sensitivity method can be viewed as a continuous-time counterpart to backpropagation, where the forward pass would correspond to integrating Equation 3 forward in time from 0 to T, and the backward pass to integrating Equation 4 backward in time from T to 0.

One possible implementation, reminiscent of backpropagation through time (BPTT), is to store the intermediate values of $\boldsymbol{z}(t)$ during the forward pass, and reuse them to compute the adjoint state during the backward pass. While several sophisticated checkpointing schemes have been proposed (Serban & Hindmarsh, 2003; Gholami et al., 2019), with different compute/memory trade-offs, the memory requirements of this approach typically grow with T; this is similar to the memory limitations standard gradient-based meta-learning methods suffer from as the number of gradient steps increases. An alternative is to augment the adjoint state $\boldsymbol{a}(t)$ with the original state $\boldsymbol{z}(t)$, and to solve this augmented dynamical system backward in time (Chen et al., 2018). This has the notable advantage that the memory requirements are now independent of T, since $\boldsymbol{z}(t)$ are no longer stored during the forward pass, but they are recomputed on the fly during the backward pass.

Forward sensitivity method While the adjoint method is related to reverse-mode automatic differentiation (backpropagation), the *forward sensitivity method* (Feehery et al., 1997; Leis & Kramer, 1988; Maly & Petzold, 1996; Caracotsios & Stewart, 1985), on the other hand, can be viewed as the continuous-time counterpart to forward (tangent-linear) mode differentiation (Griewank & Walther, 2008). This method is based on the fact that the derivative $\mathcal{S}(t) \triangleq d\boldsymbol{z}(t)/d\boldsymbol{\theta}$ is the solution of the so-called *forward sensitivity equation*

$$\frac{d\mathcal{S}}{dt} = \frac{\partial g\big(\boldsymbol{z}(t);\boldsymbol{\theta}\big)}{\partial \boldsymbol{z}(t)}\mathcal{S}(t) + \frac{\partial g\big(\boldsymbol{z}(t);\boldsymbol{\theta}\big)}{\partial \boldsymbol{\theta}} \qquad\qquad \mathcal{S}(0) = \frac{\partial \boldsymbol{z}_0}{\partial \boldsymbol{\theta}}. \tag{5}$$

This equation can be found throughout the literature in optimal control and system identification (Betts, 2010; Biegler, 2010). Unlike the adjoint method, which requires an explicit forward *and* backward pass, the forward sensitivity method only requires the integration forward in time of the original ODE in Equation 3, augmented by the sensitivity state $\mathcal{S}(t)$ with the dynamics above. The memory requirements of the forward sensitivity method do not scale with T either, but it now requires storing $\mathcal{S}(t)$, which may be very large; we will come back to this problem in Section 3.2. We will simply note here that in discrete-time, this is the same issue afflicting forward-mode training of RNNs with real-time recurrent learning (RTRL; Williams & Zipser, 1989), or other meta-learning algorithms (Sutton, 1992; Franceschi et al., 2017; Xu et al., 2018).

3 Continuous-time adaptation

In the limit of infinitely small steps, some optimization algorithms can be viewed as the solution trajectory of a differential equation. This point of view has often been taken to analyze their behavior (Platt & Barr, 1988; Wilson et al., 2016; Su et al., 2014; Orvieto & Lucchi, 2019). In fact, some optimization algorithms such as gradient descent with momentum have even been introduced initially from the perspective of dynamical systems (Polyak, 1964). As the simplest example, gradient descent with a constant step size $\alpha \to 0^+$ (i.e. α tends to 0 by positive values) corresponds to following the dynamics of an autonomous ODE called a *gradient vector field*

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t - \alpha\nabla f(\boldsymbol{z}_t) \qquad \underset{\alpha\to 0^+}{\longrightarrow} \qquad \frac{d\boldsymbol{z}}{dt} = -\nabla f\big(\boldsymbol{z}(t)\big), \tag{6}$$

where the iterate $\boldsymbol{z}(t)$ is now a continuous function of time t. The solution of this dynamical system is uniquely defined by the choice of the initial condition $\boldsymbol{z}(0) = \boldsymbol{z}_0$.

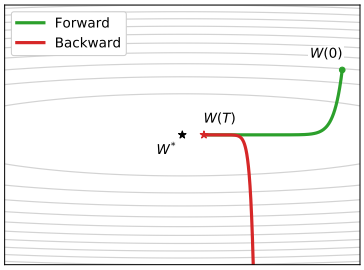


Figure 2: Numerical instability of the adjoint method applied to the gradient vector field of a quadratic loss function. The trajectory in green starting at $\boldsymbol{W}(0)$ corresponds to the integration of the dynamical system in Equation 8 forward in time up to T, and the trajectory in red starting at $\boldsymbol{W}(T)$ corresponds to its integration backward in time. Note that T was chosen so that $\boldsymbol{W}(T)$ does not reach the equilibrium/minimum of the loss $\boldsymbol{W}^\star$.

3.1 Continuous-time meta-learning

In gradient-based meta-learning, the task-specific adaptation with gradient descent may also be replaced by a gradient vector field in the limit of infinitely small steps. Inspired by prior work in meta-learning (Raghu et al., 2019; Javed & White, 2019), we assume that an embedding network f_Φ with meta-parameters Φ is shared across tasks, and only the parameters $\boldsymbol{W}$ of a task-specific linear classifier are adapted, starting at some initialization $\boldsymbol{W}_0$. Instead of being the result of a few steps of gradient descent though, the final parameters $\boldsymbol{W}(T)$ now correspond to integrating an ODE similar to Equation 6 up to a certain horizon T, with the initial conditions $\boldsymbol{W}(0) = \boldsymbol{W}_0$. We call this new meta-learning algorithm Continuous-Time Meta-Learning[1] (COMLN).

Treating the learning algorithm as a continuous-time process has the notable advantage that the adapted parameter $\boldsymbol{W}(T)$ is now differentiable wrt. the time horizon T (Wiggins, 2003, Chap. 7), in addition to being differentiable wrt. the initial conditions $\boldsymbol{W}_0$—which plays a central role in gradient-based meta-learning, as described in Section 2.1. This allows us to view T as a meta-parameter on par with Φ and $\boldsymbol{W}_0$, and to effectively optimize the amount of adaptation using stochastic gradient descent (SGD). The meta-learning objective of COMLN can be written as

$$\min_{\Phi, \boldsymbol{W}_0, T} \; \mathbb{E}_\tau \big[\mathcal{L}\big(\boldsymbol{W}_\tau(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big) \big] \tag{7}$$

$$\text{s.t.} \;\; \frac{d\boldsymbol{W}_\tau}{dt} = -\nabla \mathcal{L}\big(\boldsymbol{W}_\tau(t); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big) \qquad \boldsymbol{W}_\tau(0) = \boldsymbol{W}_0 \qquad \forall \tau \sim p(\tau), \tag{8}$$

where $f_\Phi(\mathcal{D}_\tau^{\text{train}}) = \{(f_\Phi(\boldsymbol{x}_m), \boldsymbol{y}_m) \mid (\boldsymbol{x}_m, \boldsymbol{y}_m) \in \mathcal{D}_\tau^{\text{train}}\}$ is the embedded training dataset, and $f_\Phi(\mathcal{D}_\tau^{\text{test}})$ is defined similarly for $\mathcal{D}_\tau^{\text{test}}$. In practice, adaptation is implemented using a numerical integration scheme based on an iterative discretization of the problem, such as Runge-Kutta methods. Although a complete discussion of numerical solvers is outside of the scope of this paper, we recommend (Butcher, 2008) for a comprehensive overview of numerical methods for solving ODEs.

3.2 The challenges of optimizing the meta-learning objective

In order to minimize the meta-learning objective of COMLN, it is common practice to use (stochastic) gradient methods; that requires computing its derivatives wrt. the meta-parameters, which we call *meta-gradients*. Our primary goal is to devise an algorithm whose memory requirements do not scale with the amount of adaptation T; this would contrast with standard gradient-based meta-learning methods that backpropagate through a sequence of gradient steps (similar to BPTT), where the intermediate parameters are stored during adaptation (i.e. $\boldsymbol{\theta}_t^\tau$ for all t in Equation 2). Since this objective involves the solution $\boldsymbol{W}(T)$ of an ODE, we can use either the adjoint method, or the forward sensitivity method, in order to compute the derivatives wrt. Φ and $\boldsymbol{W}_0$ (see Section 2.2).

Although the adjoint method has proven to be an effective strategy for learning Neural ODEs, in practice computing the state $\boldsymbol{W}(t)$ backward in time is numerically unstable when applied to a gradient vector field like the one in Equation 8, even for convex loss functions. Figure 2 shows an example where the trajectory of $\boldsymbol{W}(t)$ recomputed backward in time (in red) diverges significantly from the original trajectory (in green) on a quadratic loss function, even though the two should match exactly in theory since they follow the same dynamics. Intuitively, recomputing $\boldsymbol{W}(t)$ backward in time for a gradient vector field requires doing gradient *ascent* on the loss function, which is prone to compounding numerical errors; this is closely related to the loss of entropy observed by Maclaurin

[1] COMLN is pronounced *chameleon*

et al. (2015). This divergence makes the backward trajectory of $\boldsymbol{W}(t)$ unreliable to find the adjoint state, ruling out the adjoint sensitivity method for computing the meta-gradients in COMLN.

The forward sensitivity method addresses this shortcoming by avoiding the backward pass altogether. However, it can also be particularly expensive here in terms of memory requirements, since the sensitivity state $\mathcal{S}(t)$ in Section 2.2 now corresponds to Jacobian matrices, such as $d\boldsymbol{W}(t)/d\boldsymbol{W}_0$. As the size d of the feature vectors returned by f_Φ may be very large, this $Nd \times Nd$ Jacobian matrix would be almost impossible to store in practice; for example in our experiments, it can be as large as $d = 16{,}000$ for a ResNet-12 backbone. In Section 4.1, we will show how to apply forward sensitivity in a memory-efficient way, by leveraging the structure of the loss function. This is achieved by carefully decomposing the Jacobian matrices into smaller pieces that follow specific dynamics. We show in Appendix D that unlike the adjoint method, this process is stable.

3.3 Connection with Almost No Inner-Loop (ANIL)

Similarly to ANIL (Raghu et al., 2019), COMLN only adapts the parameters $\boldsymbol{W}$ of the last linear layer of the neural network. There is a deeper connection between both algorithms though: while our description of the adaptation in COMLN (Eq. 8) was independent of the choice of the ODE solver used to find the solution $\boldsymbol{W}(T)$ in practice, if we choose an explicit Euler scheme (Euler, 1913, roughly speaking, discretizing Equation 6 from right to left), then the adaptation of COMLN becomes functionally equivalent to ANIL. However, this equivalence can greatly benefit from the memory-efficient algorithm to compute the meta-gradients described in Section 4, based on the forward sensitivity method. This means that using the methods devised here for COMLN, we can effectively compute the meta-gradients of ANIL with a constant memory cost wrt. the number of gradient steps of adaptation, instead of relying on backpropagation (see also Section 4.2).

4 Memory-efficient meta-gradients

For some fixed task τ and $(\boldsymbol{x}_m, \boldsymbol{y}_m) \in \mathcal{D}_\tau^{\text{train}}$, let $\boldsymbol{\phi}_m = f_\Phi(\boldsymbol{x}_m) \in \mathbb{R}^d$ be the embedding of input $\boldsymbol{x}_m$ through the feature extractor f_Φ. Since we are confronted with a classification problem, the loss function of choice $\mathcal{L}(\boldsymbol{W})$ is typically the cross-entropy loss. Böhning (1992) showed that the gradient of the cross-entropy loss wrt. $\boldsymbol{W}$ can be written as

$$\nabla \mathcal{L}\big(\boldsymbol{W}; f_\Phi(\mathcal{D}_\tau^{\text{train}})\big) = \frac{1}{M} \sum_{m=1}^{M} (\boldsymbol{p}_m - \boldsymbol{y}_m)\boldsymbol{\phi}_m^\top, \tag{9}$$

where $\boldsymbol{p}_m = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{\phi}_m)$ is the vector of probabilities returned by the neural network. The key observation here is that the gradient can be decomposed as a sum of M rank-one matrices, where the feature vectors $\boldsymbol{\phi}_m$ are independent of $\boldsymbol{W}$. Therefore we can fully characterize the gradient of the cross-entropy loss with M vectors $\boldsymbol{p}_m - \boldsymbol{y}_m \in \mathbb{R}^N$, as opposed to the full $N \times d$ matrix. This is particularly useful in the context of few-shot classification, where the number of training examples M is small, and typically significantly smaller than the embedding size d.

4.1 Decomposition of the meta-gradients

We saw in Section 3.2 that the forward sensitivity method was the only stable option to compute the meta-gradients of COMLN. However, naively applying the forward sensitivity equation would involve quantities that typically scale with d^2, which can be too expensive in practice. Using the structure of Equation 9, the Jacobian matrices appearing in the computation of the meta-gradients for COMLN can be decomposed in such a way that only small quantities will depend on time.

Meta-gradients wrt. $\boldsymbol{W}_0$ By the chain rule of derivatives, it is sufficient to compute the Jacobian matrix $d\boldsymbol{W}(T)/d\boldsymbol{W}_0$ in order to obtain the meta-gradient wrt. $\boldsymbol{W}_0$. We show in Appendix B.2 that the sensitivity state $d\boldsymbol{W}(t)/d\boldsymbol{W}_0$ can be decomposed as:

$$\frac{d\boldsymbol{W}(t)}{d\boldsymbol{W}_0} = \boldsymbol{I} - \sum_{i=1}^{M}\sum_{j=1}^{M} \boldsymbol{B}_t[i,j] \otimes \boldsymbol{\phi}_i \boldsymbol{\phi}_j^\top, \tag{10}$$

Table 1: Memory required to compute meta-gradients for different algorithms. Exact: the method returns the exact meta-gradients. Full net.: the whole network is adapted, with a number of meta-parameters $|\boldsymbol{\theta}|$. The requirements for checkpointing are taken from (Shaban et al., 2019). Note that typically $M \ll d$ in few-shot learning.

Model	**Exact**	**Full net.**	**Memory**
MAML (Finn et al., 2017)	✓	✓	$\mathcal{O}(\lvert\boldsymbol{\theta}\rvert \cdot T)$
ANIL (Raghu et al., 2019)	✓	✗	$\mathcal{O}(Nd \cdot T)$
Checkpointing (every $\sqrt{T}$ steps)	✓	✓	$\mathcal{O}(\lvert\boldsymbol{\theta}\rvert \cdot \sqrt{T})$
iMAML (Rajeswaran et al., 2019)	✗	✓	$\mathcal{O}(\lvert\boldsymbol{\theta}\rvert)$
Forward sensitivity (naive)	✓	✗	$\mathcal{O}(N^2d^2 + MNd^2)$
COMLN	✓	✗	$\mathcal{O}(M^2N^2 + M^3N)$

where $\otimes$ is the Kronecker product, and each $\boldsymbol{B}_t[i,j]$ is an $N \times N$ matrix, solution of the following system of ODEs[2]

$$\frac{d\boldsymbol{B}_t[i,j]}{dt} = \mathbb{1}(i=j)\boldsymbol{A}_i(t) - \boldsymbol{A}_i(t)\sum_{m=1}^{M}(\boldsymbol{\phi}_i^\top \boldsymbol{\phi}_m)\boldsymbol{B}_t[m,j] \qquad \boldsymbol{B}_0[i,j] = \boldsymbol{0}, \tag{11}$$

and $\boldsymbol{A}_i(t)$ is also an $N \times N$ matrix, that only depends on $\boldsymbol{W}(t)$ and $\boldsymbol{\phi}_i$. The main consequence of this decomposition is that we can simply integrate the augmented ODE in $\big\{\boldsymbol{W}(t), \boldsymbol{B}_t[i,j]\big\}$ up to T to obtain the desired Jacobian matrix, along with the adapted parameters $\boldsymbol{W}(T)$. Furthermore, in contrast to naively applying the forward sensitivity method (see Section 3.2), the M^2 matrices $\boldsymbol{B}_t[i,j]$ are significantly smaller than the full Jacobian matrix. In fact, we show in Appendix C that we can compute vector-Jacobian products—required for the chain rule—using only these smaller matrices, and without having to explicitly construct the $Nd \times Nd$ matrix $d\boldsymbol{W}(t)/d\boldsymbol{W}_0$ with Eq. 10.

Meta-gradients wrt. Φ To backpropagate the error through the embedding network f_Φ, we need to first compute the gradients of the outer-loss wrt. the feature vectors $\boldsymbol{\phi}_m$. Again, by the chain rule, we can get these gradients with the Jacobian matrices $d\boldsymbol{W}(T)/d\boldsymbol{\phi}_m$. Similar to Equation 10, we can show that these Jacobian matrices can be decomposed as:

$$\frac{d\boldsymbol{W}(t)}{d\boldsymbol{\phi}_m} = -\Bigg[\boldsymbol{s}_m(t) \otimes \boldsymbol{I} + \sum_{i=1}^{M}\boldsymbol{B}_t[i,m]\boldsymbol{W}_0 \otimes \boldsymbol{\phi}_i + \sum_{i=1}^{M}\sum_{j=1}^{M}\boldsymbol{z}_t[i,j,m]\boldsymbol{\phi}_j^\top \otimes \boldsymbol{\phi}_i\Bigg], \tag{12}$$

where $\boldsymbol{s}_m(t)$ and $\boldsymbol{z}_t[i,j,m]$ are vectors of size N, that follow some dynamics; the exact form of this system of ODEs, as well as the proof of this decomposition, are given in Appendix B.3. Crucially, the only quantities that depend on time are small objects independent of the embedding size d. Following the same strategy as above, we can incorporate these vectors in the augmented ODE, and integrate it to get the necessary Jacobians. Once all the $d\boldsymbol{W}(t)/d\boldsymbol{\phi}_m$ are known, for all the training datapoints, we can apply standard backpropagation through f_Φ to obtain the meta-gradients wrt. Φ.

Meta-gradient wrt. T One of the major novelties of COMLN is the capacity to meta-learn the amount of adaptation using stochastic gradient descent. To compute the meta-gradient wrt. the time horizon T, we can directly borrow the results derived by Chen et al. (2018) in the context of Neural ODEs, and apply it to our gradient vector field in Equation 8 responsible for adaptation:

$$\frac{d\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{dT} = -\left[\frac{\partial\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{\partial\boldsymbol{W}(T)}\right]^\top \frac{\partial\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big)}{\partial\boldsymbol{W}(T)}. \tag{13}$$

The proof is available in Appendix B.4. Interestingly, we find that this involves the alignment between the vectors of partial derivatives of the inner-loss and the outer-loss at $\boldsymbol{W}(T)$, which appeared in different contexts in the meta-learning and the multi-task learning literature (Li et al., 2018; Rothfuss et al., 2019; Yu et al., 2020).

[2]Here we used the notation $\boldsymbol{B}_t[i,j]$ to make the dependence on t explicit, without overloading the notation. A more precise notation would be $\boldsymbol{B}[i,j](t)$.

Table 2: Few-shot classification on *mini*ImageNet & *tiered*ImageNet. The average accuracy (%) on 1,000 held-out meta-test tasks is reported with 95% confidence interval. ✓ denotes gradient-based meta-learning algorithms. $\star$ denotes baseline results we executed using the official implementations.

Model		**Backbone**	***mini*ImageNet 5-way**		***tiered*ImageNet 5-way**	
			1-shot	**5-shot**	**1-shot**	**5-shot**
MAML (Finn et al., 2017)	✓	Conv-4	48.70 ± 1.84	63.11 ± 0.92	51.67 ± 1.81	70.30 ± 1.75
ANIL (Raghu et al., 2019)	✓	Conv-4	46.30 ± 0.40	61.00 ± 0.60	49.35 ± 0.26	65.82 ± 0.12
Meta-SGD (Li et al., 2017)	✓	Conv-4	50.47 ± 1.87	64.03 ± 0.94	52.80 ± 0.44	62.35 ± 0.26
CAVIA (Zintgraf et al., 2019)	✓	Conv-4	51.82 ± 0.65	65.85 ± 0.55	$52.41 \pm 2.64^\star$	$67.55 \pm 2.05^\star$
iMAML (Rajeswaran et al., 2019)	✓	Conv-4	49.30 ± 1.88	$59.77 \pm 0.73^\star$	$38.54 \pm 1.37^\star$	$60.24 \pm 0.76^\star$
MetaOptNet-RR (Lee et al., 2019)		Conv-4	$\mathbf{53.23 \pm 0.59}$	69.51 ± 0.48	54.63 ± 0.67	$\mathbf{72.11 \pm 0.59}$
MetaOptNet-SVM (Lee et al., 2019)		Conv-4	52.87 ± 0.57	68.76 ± 0.48	$\mathbf{54.71 \pm 0.67}$	71.79 ± 0.59
COMLN (Ours)	✓	Conv-4	53.01 ± 0.62	$\mathbf{70.54 \pm 0.54}$	54.30 ± 0.69	71.35 ± 0.57
MAML (Finn et al., 2017)	✓	ResNet-12	49.92 ± 0.65	63.93 ± 0.59	55.37 ± 0.74	72.93 ± 0.60
ANIL (Raghu et al., 2019)	✓	ResNet-12	49.65 ± 0.65	59.51 ± 0.56	54.77 ± 0.76	69.28 ± 0.67
MetaOptNet-RR (Lee et al., 2019)		ResNet-12	61.41 ± 0.61	77.88 ± 0.46	65.36 ± 0.71	81.34 ± 0.52
MetaOptNet-SVM (Lee et al., 2019)		ResNet-12	$\mathbf{62.64 \pm 0.61}$	$\mathbf{78.63 \pm 0.46}$	$\mathbf{65.99 \pm 0.72}$	$\mathbf{81.56 \pm 0.53}$
COMLN (Ours)	✓	ResNet-12	59.26 ± 0.65	77.26 ± 0.49	62.93 ± 0.71	81.13 ± 0.53

4.2 Memory efficiency

Although naively applying the forward sensitivity method would be memory intensive, we have shown in Section 4.1 that the Jacobians can be carefully decomposed into smaller pieces. It turns out that even the parameters $\boldsymbol{W}(t)$ can be expressed using the vectors $\boldsymbol{s}_m(t)$ from the decomposition in Equation 12; see Appendix B.1 for details. As a consequence, to compute the adapted parameters $\boldsymbol{W}(T)$ as well as all the necessary meta-gradients, it is sufficient to integrate a dynamical system in $\big\{\boldsymbol{B}_t[i,j], \boldsymbol{s}_m(t), \boldsymbol{z}_t[i,j,m]\big\}$ (see Algorithms 1 & 2 in App. A.1), involving exclusively quantities that are independent of the embedding size d. Instead, the size of that system scales with M the total number of training examples, which is typically much smaller than d for few-shot classification.

Table 1 shows a comparison of the memory cost for different algorithms. It is important to note that contrary to other standard gradient-based meta-learning methods, the memory requirements of COMLN do not scale with the amount of adaptation T (i.e. the number of gradient steps in MAML & ANIL), while still returning the exact meta-gradients—unlike iMAML (Rajeswaran et al., 2019), which only returns an approximation of the meta-gradients. We verified empirically this efficiency, both in terms of memory and computation costs, in Section 5.2.

5 Experiments

For our embedding network f_Φ, we consider two commonly used architectures in meta-learning: Conv-4, a convolutional neural network with 4 convolutional blocks, and ResNet-12, a 12-layer residual network (He et al., 2016). Note that following (Lee et al., 2019), ResNet-12 does not include a global pooling layer at the end of the network, leading to feature vectors with embedding dimension $d = 16,000$. Additional details about these architectures are given in Appendix E. To compute the adapted parameters and the meta-gradients in COMLN, we integrate the dynamical system described in Section 4.2 with a 4th order Runge-Kutta method with a Dormand Prince adaptive step size (Runge, 1895; Dormand & Prince, 1980); we will come back to the choice of this numerical solver in Section 5.2. Furthermore to ensure that $T > 0$, we parametrized it with an exponential activation.

5.1 Few-shot image classification

We evaluate COMLN on two standard few-shot image classification benchmarks: the *mini*ImageNet (Vinyals et al., 2016) and the *tiered*ImageNet datasets (Ren et al., 2018), both datasets being derived from ILSVRC-2012 (Russakovsky et al., 2015). The process for creating tasks follows the standard procedure from the few-shot classification literature (Santoro et al., 2016), with distinct classes

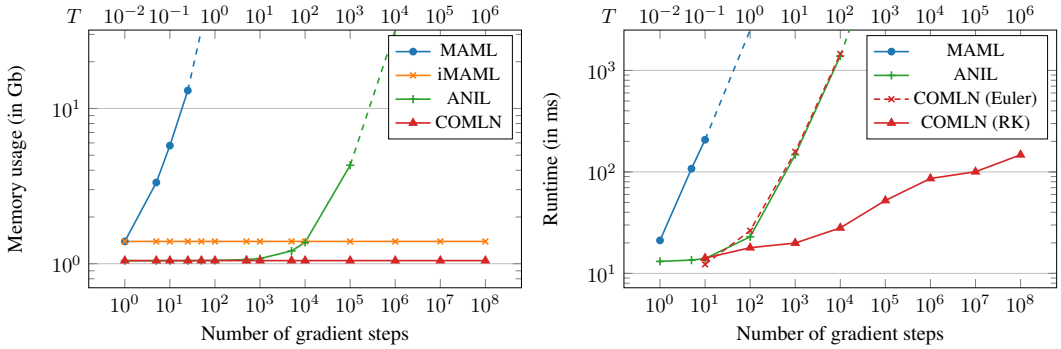


Figure 3: Empirical efficiency of COMLN on a single 5-shot 5-way task, with a Conv-4 backbone. (Left) Memory usage for computing the meta-gradients as a function of the number of inner-gradient steps. The extrapolated dashed lines correspond to the method reaching the memory capacity of a Tesla V100 GPU with 32Gb of memory. (Right) Average time taken (in ms) to compute the exact meta-gradients. The extrapolated dashed lines correspond to the method taking over 2 seconds.

between the different splits. *mini*Imagenet consists of 100 classes, split into 64 training classes, 16 validation classes, and 20 test classes. *tiered*ImageNet consists of 608 classes grouped into 34 high-level categories from ILSVRC-2012, split into 20 training, 6 validation, and 8 testing categories—corresponding to 351/97/160 classes respectively; Ren et al. (2018) argue that separating data according to high-level categories results in a more difficult and more realistic regime.

Table 2 shows the average accuracies of COMLN compared to various meta-learning methods, be it gradient-based or not. For both backbones, COMLN decisively outperforms all other gradient-based meta-learning methods. Compared to methods that explicitly backpropagate through the learning process, such as MAML or ANIL, the performance gain shown by COMLN could be credited to the longer adaptation T it learns, as opposed to a small number of gradient steps—usually about 10 steps; this was fully enabled by our memory-efficient method to compute meta-gradients, which does not scale with the length of adaptation anymore (see Section 4.2). We analyse the evolution of T during meta-training for these different settings in Appendix E.3. In almost all settings, COMLN is even closing the gap with a strong non-gradient-based method like MetaOptNet; the remainder may be explained in part by the training choices made by Lee et al. (2019) (see Appendix E for details).

5.2 Empirical efficiency of COMLN

In Section 4.2, we showed that our algorithm to compute the meta-gradients, based on forward differentiation, had a memory cost independent of the length of adaptation T. We verify this empirically in Figure 3, where we compare the memory required by COMLN and other methods to compute the meta-gradients on a single task, with a Conv-4 backbone (Figure 4 in Appendix E.2 shows similar results for ResNet-12). To ensure an aligned comparison between discrete and continuous time, we use a conversion corresponding to a learning rate $\alpha = 0.01$ in Equation 2; see Appendix E.2 for a justification. As expected, the memory cost increases for both MAML and ANIL as the number of gradient steps increases, while it remains constant for iMAML and COMLN. Interestingly, we observe that the cost of COMLN is equivalent to the cost of running ANIL for a small number of steps, showing that the additional cost of integrating the augmented ODE in Section 4.2 to compute the meta-gradients is minimal.

Increasing the length of adaptation also has an impact on the time it takes to compute the adapted parameters, and the meta-gradients. Figure 3 (right) shows how the runtime increases with the amount of adaptation for different algorithms. We see that the efficiency of COMLN depends on the numerical solver used. When we use a simple explicit-Euler scheme, the time taken to compute the meta-gradients matches the one of ANIL; this behavior empirically confirms our observation in Section 3.3. When we use an adaptive numerical solver, such as Runge-Kutta (RK) with a Dormand Prince step size, this computation can be significantly accelerated, thanks to the smaller number of function evaluations. In practice, we show in Appendix E.1 that the choice of the ODE solver has a very minimal impact on the accuracy.

6 RELATED WORK

We are interested in meta-learning (Bengio et al., 1991; Schmidhuber, 1987; Thrun & Pratt, 2012), and in particular we focus on gradient-based meta-learning methods (Finn, 2018), where the learning rule is based on gradient descent. While in MAML (Finn et al., 2017) the whole network was updated during this process, follow-up works have shown that it is generally sufficient to share most parts of the neural network, and to only adapt a few layers (Raghu et al., 2019; Chen et al., 2020b; Tian et al., 2020). Even though this hypothesis has been challenged recently (Arnold & Sha, 2021), COMLN also updates only the last layer of a neural network, and therefore can be viewed as a continuous-time extension of ANIL (Raghu et al., 2019); see also Section 3.3. With its shared embedding network across tasks, COMLN is also connected to metric-based meta-learning methods (Vinyals et al., 2016; Snell et al., 2017; Sung et al., 2018; Bertinetto et al., 2018; Lee et al., 2019).

Closely related to our work, Zhang et al. (2021) also introduce a formulation where the adaptation of prototypes follows a gradient vector field, but they finally opt for modeling it as a Neural ODE (Chen et al., 2018), possibly due to the challenges of applying the adjoint method we identified in Section 3.2. Zhou et al. (2021) also uses a gradient vector field to motivate a novel method with a closed-form adaptation, based on the NTK theory; COMLN still explicitly updates the parameters following the gradient vector field, since there is no closed-form solution of Eq. 8. As mentioned in Section 3, treating optimization as a continuous-time process has been used to analyze the convergence of different optimization algorithms, including the meta-optimization of MAML (Xu et al., 2021), or to introduce new meta-optimizers based on different integration schemes (Im et al., 2019). Guo et al. (2021) also uses meta-learning to learn new integration schemes for ODEs. Although this is a growing literature at the intersection of meta-learning and dynamical systems, our work is the first algorithm that uses a gradient vector field for adaptation in meta-learning.

Beyond the memory efficiency of our method, one of the main benefits of the continuous-time perspective is that COMLN is capable of learning when to stop the adaptation, as opposed to taking a number of gradient steps fixed ahead of time. However unlike (Chen et al., 2020a), where the number of gradient steps are optimized (up to a maximal number) with variational methods, we incorporate the amount of adaptation as a (continuous) meta-parameter that can be learned using SGD. To compute the meta-gradients, which is known to be challenging for long sequences in gradient-based meta-learning, we use forward-mode differentiation as an alternative to backpropagation through the learning process, similar to prior work in meta-learning (Franceschi et al., 2017; Jiwoong Im et al., 2021). This yields the exact meta-gradients in constant memory, without any assumption on the optimality of the inner optimization problem, which is necessary when using the normal equations (Bertinetto et al., 2018), or to apply implicit differentiation (Rajeswaran et al., 2019).

7 CONCLUSION AND FUTURE WORK

In this paper, we have introduced *Continuous-Time Meta-Learning* (COMLN), a novel algorithm that treats the adaptation in meta-learning as a continuous-time process, by following the dynamics of a gradient vector field up to a certain time horizon T. One of the major novelties of treating adaptation in continuous time is that the amount of adaptation T is now a continuous quantity, that can be viewed as a meta-parameter and can be learned using SGD, alongside the initial conditions and the parameters of the embedding network. In order to learn these meta-parameters, we have also introduced a novel practical algorithm based on forward mode automatic differentiation, capable of efficiently computing the exact meta-gradients using an augmented dynamical system. We have verified empirically that this algorithm was able to compute the meta-gradients in constant memory, making it the first gradient-based meta-learning approach capable of computing the exact meta-gradients with long sequences of adaptation using gradient methods. In practice, we have shown that COMLN significantly outperforms other standard gradient-based meta-learning algorithms.

In addition to having a single meta-parameter T that drives the adaptation of all possible tasks, the fact that the time horizon can be learned with SGD opens up new possibilities for gradient-based methods. For example, we could imagine treating T not as a shared meta-parameters, but as a task-specific parameter. This would allow the learning process to be more *adaptive*, possibly with different behaviors depending on the difficulty of the task. This is left as a future direction of research.

REPRODUCIBILITY STATEMENT

We provide in Appendix A.1 a full description in pseudo-code of the meta-training procedure (Algorithm 1), along with the exact dynamics of the ODE (Algorithm 2) and the projection operations (Algorithms 3 & 4) to avoid explicitly building the Jacobian matrices to compute Jacobian-vector products (see Section 4.1).

We also provide in Appendix A.2 a snippet of code in JAX (Bradbury et al., 2018) to compute the adapted parameters $\boldsymbol{W}(T)$, as well as all the necessary objects $\left\{\boldsymbol{B}_t[i,j], \boldsymbol{s}_m(t), \boldsymbol{z}_t[i,j,m]\right\}$ to compute all the meta-gradients (see Section 4.2). We also give in Code Snippet 2 the code to compute the meta-gradients wrt. the initialization $\boldsymbol{W}_0$ and the integration time T. Computing the meta-gradients wrt. Φ involves non-minimal dependencies on Haiku (Hennigan et al., 2020), and therefore is omitted here. The full code is available in the *Supplementary Materials*.

Data generation & hyperparameters We used the *mini*ImageNet and *tiered*ImageNet datasets provided by (Lee et al., 2019) in order to create the 1-shot 5-way and 5-shot 5-way tasks for both datasets. During evaluation, for each setting, a fixed set of $1,000$ tasks were sampled; this means that both architectures for COMLN have been evaluated using exactly the same data, to ensure direct comparison across backbones. A full description of all the hyperparameters used in COMLN is given in Appendix E.

Reproducibility of baseline results To the best of our ability, we have tried to report baseline results from existing work, to limit as much as possible the bias induced by running our own baseline experiments. The references of those works are given in Table 3. We still had to run CAVIA and iMAML on the remaining settings, since these results have not been reported in the literature. For both methods, we used the data generation described above.

- *CAVIA*: We used the official implementation[3]. We used the hyperparameters reported in (Zintgraf et al., 2019) for *mini*ImageNet, and an architecture with 64 filters.
- *iMAML*: We used the official implementation[4]. We used the hyperparameters reported in (Rajeswaran et al., 2019) for *mini*ImageNet 1-shot 5-way.

Table 3: References for the results provided in Table 2: (Liu et al., 2019), (Oh et al., 2021), (Aimen et al., 2021), (Arnold et al., 2021), and are reported in their respective references (under *Model*). Recall that $\star$ denotes baseline results we executed using the official implementations.

Model		**Backbone**	***mini*ImageNet 5-way**		***tiered*ImageNet 5-way**	
			1-shot	**5-shot**	**1-shot**	**5-shot**
MAML (Finn et al., 2017)	✓	Conv-4	48.70 ± 1.84	63.11 ± 0.92	51.67 ± 1.81	70.30 ± 1.75
ANIL (Raghu et al., 2019)	✓	Conv-4	46.30 ± 0.40	61.00 ± 0.60	49.35 ± 0.26	65.82 ± 0.12
Meta-SGD (Li et al., 2017)	✓	Conv-4	50.47 ± 1.87	64.03 ± 0.94	52.80 ± 0.44	62.35 ± 0.26
CAVIA (Zintgraf et al., 2019)	✓	Conv-4	51.82 ± 0.65	65.85 ± 0.55	$52.41 \pm 2.64^{\star}$	$67.55 \pm 2.05^{\star}$
iMAML (Rajeswaran et al., 2019)	✓	Conv-4	49.30 ± 1.88	$59.77 \pm 0.73^{\star}$	$38.54 \pm 1.37^{\star}$	$60.24 \pm 0.76^{\star}$
MetaOptNet-RR (Lee et al., 2019)		Conv-4	$\mathbf{53.23 \pm 0.59}$	69.51 ± 0.48	54.63 ± 0.67	$\mathbf{72.11 \pm 0.59}$
MetaOptNet-SVM (Lee et al., 2019)		Conv-4	52.87 ± 0.57	68.76 ± 0.48	$\mathbf{54.71 \pm 0.67}$	71.79 ± 0.59
COMLN (Ours)	✓	Conv-4	53.01 ± 0.62	$\mathbf{70.54 \pm 0.54}$	54.30 ± 0.69	71.35 ± 0.57
MAML (Finn et al., 2017)	✓	ResNet-12	49.92 ± 0.65	63.93 ± 0.59	55.37 ± 0.74	72.93 ± 0.60
ANIL (Raghu et al., 2019)	✓	ResNet-12	49.65 ± 0.65	59.51 ± 0.56	54.77 ± 0.76	69.28 ± 0.67
MetaOptNet-RR (Lee et al., 2019)		ResNet-12	61.41 ± 0.61	77.88 ± 0.46	65.36 ± 0.71	81.34 ± 0.52
MetaOptNet-SVM (Lee et al., 2019)		ResNet-12	$\mathbf{62.64 \pm 0.61}$	$\mathbf{78.63 \pm 0.46}$	$\mathbf{65.99 \pm 0.72}$	$\mathbf{81.56 \pm 0.53}$
COMLN (Ours)	✓	ResNet-12	59.26 ± 0.65	77.26 ± 0.49	62.93 ± 0.71	81.13 ± 0.53

[3] https://github.com/lmzintgraf/cavia/
[4] https://github.com/aravindr93/imaml_dev

REFERENCES

Aroof Aimen, Sahil Sidheekh, and Narayanan C Krishnan. Task Attended Meta-Learning for Few-Shot Learning. *arXiv preprint*, 2021.

Sébastien MR Arnold and Fei Sha. Embedding Adaptation is Still Needed for Few-Shot Learning. *arXiv preprint*, 2021.

Sébastien MR Arnold, Guneet S Dhillon, Avinash Ravichandran, and Stefano Soatto. Uniform Sampling over Episode Difficulty. *arXiv preprint*, 2021.

Dmitry Baranchuk. Memory Efficient MAML, 2019. URL `https://github.com/dbaranchuk/memory-efficient-maml`.

Yoshua Bengio, Samy Bengio, Jocelyn Cloutier, and Jan Gecsei. Learning a Synaptic Learning Rule. *International Joint Conference on Neural Networks*, 1991.

Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with Differentiable Closed-Form Solvers. *arXiv preprint*, 2018.

John T Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. SIAM, 2010.

Lorenz T. Biegler. *Nonlinear Programming*. Society for Industrial and Applied Mathematics, January 2010.

Dankmar Böhning. Multinomial Logistic Regression Algorithm. *Annals of the Institute of Statistical Mathematics*, 1992.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

A. E. Bryson and Y. C. Ho. *Applied Optimal Control*. Blaisdell, New York, 1969.

John Charles Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2008.

Makis Caracotsios and Warren E Stewart. Sensitivity analysis of initial value problems with mixed odes and algebraic equations. *Computers & Chemical Engineering*, 9(4):359–365, 1985.

G Chavent, RE Goodson, and M Polis. Identification of parameter distributed systems. *Identification of function parameters in partial differential equations*, pp. 31–48, 1974.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. *Advances in Neural Information Processing Systems*, 2018.

Xinshi Chen, Hanjun Dai, Yu Li, Xin Gao, and Le Song. Learning To Stop While Learning To Predict. In *International Conference on Machine Learning*, 2020a.

Yutian Chen, Abram L Friesen, Feryal Behbahani, Arnaud Doucet, David Budden, Matthew W Hoffman, and Nando de Freitas. Modular Meta-Learning with Shrinkage. *Neural Information Processing Systems*, 2020b.

John R Dormand and Peter J Prince. A family of embedded Runge-Kutta formulae. *Journal of computational and applied mathematics*, 1980.

Leonhard Euler. De integratione aequationum differentialium per approximationem. *Opera Omnia*, 1913.

William F Feehery, John E Tolsma, and Paul I Barton. Efficient sensitivity analysis of large-scale differential-algebraic systems. *Applied Numerical Mathematics*, 25(1):41–54, 1997.

Chelsea Finn. *Learning to Learn with Gradients*. PhD thesis, UC Berkeley, 2018.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *International Conference on Machine Learning (ICML)*, 2017.

Sebastian Flennerhag, Pablo G Moreno, Neil D Lawrence, and Andreas Damianou. Transferring knowledge across learning processes. *arXiv preprint*, 2018.

Sebastian Flennerhag, Andrei A Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-Learning with Warped Gradient Descent. *International Conference on Learning Representations*, 2020.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, 2017.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In *Neural Information Processing Systems*, 2018.

Amir Gholami, Kurt Keutzer, and George Biros. ANODE: unconditionally accurate memory-efficient gradients for neural odes. *CoRR*, abs/1902.10298, 2019. URL `http://arxiv.org/abs/1902.10298`.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives*. Society for Industrial and Applied Mathematics, January 2008.

Yue Guo, Felix Dietrich, Tom Bertalan, Danimir T Doncevic, Manuel Dahmen, Ioannis G Kevrekidis, and Qianxiao Li. Personalized Algorithm Generation: A Case Study in Meta-Learning ODE Integrators. *arXiv preprint*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020. URL `http://github.com/deepmind/dm-haiku`.

Daniel Jiwoong Im, Yibo Jiang, and Nakul Verma. Model-Agnostic Meta-Learning using Runge-Kutta Methods. *arXiv preprint*, 2019.

Khurram Javed and Martha White. Meta-Learning Representations for Continual Learning. In *Advances in Neural Information Processing Systems*, 2019.

Daniel Jiwoong Im, Cristina Savin, and Kyunghyun Cho. Online hyperparameter optimization by Real-Time Recurrent Learning. *arXiv preprint*, 2021.

Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with Differentiable Convex Optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Jorge R Leis and Mark A Kramer. The simultaneous solution and sensitivity analysis of systems described by ordinary differential equations. *ACM Transactions on Mathematical Software (TOMS)*, 14(1):45–60, 1988.

Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. Learning to Generalize: Meta-Learning for Domain Generalization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-SGD: Learning to learn quickly for few-shot learning. *arXiv preprint*, 2017.

Jacques Louis Lions and Enrico Magenes. *Non-homogeneous boundary value problems and applications: Vol. 1*, volume 181. Springer Science & Business Media, 2012.

Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. Learning to Propagate Labels: Transductive Propagation Network for Few-Shot Learning. *International Conference on Learning Representations*, 2019.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pp. 2113–2122. PMLR, 2015.

Timothy Maly and Linda R Petzold. Numerical methods and software for sensitivity analysis of differential-algebraic systems. *Applied Numerical Mathematics*, 20(1-2):57–79, 1996.

Alex Nichol, Joshua Achiam, and John Schulman. On First-Order Meta-Learning Algorithms. *arXiv preprint*, 2018.

Jaehoon Oh, Hyungjun Yoo, ChangHwan Kim, and Se-Young Yun. BOIL: Towards Representation Change for Few-Shot Learning. *International Conference on Learning Representations*, 2021.

Antonio Orvieto and Aurelien Lucchi. Shadowing Properties of Optimization Algorithms. In *Advances in Neural Information Processing Systems*, 2019.

John Platt and Alan Barr. Constrained Differential Optimization. In *Neural Information Processing Systems*, 1988.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR computational mathematics and mathematical physics*, 1964.

Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. *arXiv preprint*, 2019.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-Learning with Implicit Gradients. In *Advances in Neural Information Processing Systems*, 2019.

Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. In *International Conference on Learning Representations*, 2018.

Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. ProMP: Proximal Meta-Policy Search. *International Conference on Learning Representations*, 2019.

Carl Runge. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 1895.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with Latent Embedding Optimization. *arXiv preprint*, 2018.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint*, 2016.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Radu Serban and Alan C Hindmarsh. Cvodes: An ode solver with sensitivity analysis capabilities. Technical report, Technical Report UCRL-JP-200039, Lawrence Livermore National Laboratory, 2003.

Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated Back-propagation for Bilevel Optimization. In *International Conference on Artificial Intelligence and Statistics*, 2019.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical Networks for Few-shot Learning. In *Advances in Neural Information Processing Systems*, 2017.

Weijie Su, Stephen Boyd, and Emmanuel Candes. A Differential Equation for Modeling Nesterov's Accelerated Gradient Method: Theory and Insights. *Advances in Neural Information Processing Systems*, 2014.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1199–1208, 2018.

Richard S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In William R. Swartout (ed.), *Proceedings of the 10th National Conference on Artificial Intelligence, San Jose, CA, USA, July 12-16, 1992*, pp. 171–176. AAAI Press / The MIT Press, 1992.

Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. Rethinking Few-Shot Image Classification: a Good Embedding Is All You Need? 2020.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Neural Information Processing Systems*, 29:3630–3638, 2016.

Stephen Wiggins. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*, volume 2. Springer, 2003.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, June 1989.

Ashia C Wilson, Benjamin Recht, and Michael I Jordan. A Lyapunov Analysis of Momentum Methods in Optimization. *arXiv preprint*, 2016.

Ruitu Xu, Lin Chen, and Amin Karbasi. Meta Learning in the Continuous Time Limit . In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. PMLR, 2021.

Zhongwen Xu, Hado van Hasselt, and David Silver. Meta-gradient reinforcement learning. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 2402–2413, 2018.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient Surgery for Multi-Task Learning. *Neural Information Processing Systems*, 2020.

Baoquan Zhang, Xutao Li, Yunming Ye, Shanshan Feng, and Rui Ye. MetaNODE: Prototype Optimization as a Neural ODE for Few-Shot Learning. *arXiv preprint*, 2021.

Yufan Zhou, Zhenyi Wang, Jiayi Xian, Changyou Chen, and Jinhui Xu. Meta-Learning with Neural Tangent Kernels. *International Conference on Learning Representations*, 2021.

Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pp. 7693–7702. PMLR, 2019.

Appendix

In these appendices, we provide all the details omitted in the main text due to space constraints. They are organized as follows: in Appendix A we give the pseudo-code for meta-training and meta-testing COMLN, along with minimal code in JAX (Bradbury et al., 2018). In Appendix B we prove the decomposition of the Jacobians introduced in Section 4, and we give the exact dynamics of $\boldsymbol{s}_m(t)$ and $\boldsymbol{z}_t[i,j,m]$ in the decomposition of $d\boldsymbol{W}(t)/d\boldsymbol{\phi}_m$, omitted from the main text for concision. In Appendix C we show how the total derivatives may be computed from the Jacobian matrices without ever having to form them explicitly, hence maintaining the memory-efficiency described in Section 4.2. In Appendix D, we show that unlike the adjoint method described in Section 2.2, the algorithm used in COMLN to compute the meta-gradients based on the forward sensitivity method is stable and guaranteed not to diverge. Finally in Appendix E we give additional details regarding the experiments reported in Section 5, together with additional analyses and results on the dataset introduced in (Rusu et al., 2018).

A Algorithmic details

A.1 Meta-training pseudo-code

We give in Algorithm 1 the pseudo-code for meta-training COMLN, based on a distribution of tasks $p(\tau)$, with references to the relevant propositions developed in Appendices B and C. Note that to simplify the presentation in Algorithm 1, we introduce the notations ∇_τ and $\partial\mathcal{L}_\tau$ to denote the total derivative of the outer-loss, and the partial derivative of the loss $\mathcal{L}$ respectively, both computed at the adapted parameters $\boldsymbol{W}_\tau(T)$. For example:

$$\nabla_\tau \boldsymbol{W}_0 = \frac{d\mathcal{L}\big(\boldsymbol{W}_\tau(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{d\boldsymbol{W}_0} \qquad \partial_{\boldsymbol{W}(T)}\mathcal{L}_\tau^{\text{train}} = \frac{\partial\mathcal{L}\big(\boldsymbol{W}_\tau(T); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big)}{\partial\boldsymbol{W}(T)}$$

Algorithm 1 COMLN – Meta-training

Require: A task distribution $p(\tau)$
  Initialize randomly Φ and $\boldsymbol{W}_0$. Initialize T to a small value $\varepsilon > 0$.
  loop
    Sample a batch of tasks $\mathcal{B} \sim p(\tau)$
    for all $\tau \in \mathcal{B}$ **do**
      Embed the training set: $f_\Phi(\mathcal{D}_\tau^{\text{train}}) = \{(\boldsymbol{\phi}_m, \boldsymbol{y}_m)\}_{m=1}^M$
      $\boldsymbol{s}(T), \boldsymbol{B}_T, \boldsymbol{z}_T \leftarrow$ ODESolve$([\mathbf{0},\mathbf{0},\mathbf{0}], \textsc{Dynamics}, 0, T)$
      $\boldsymbol{W}_\tau(T) \leftarrow \boldsymbol{W}_0 - \sum_m \boldsymbol{s}_m(T)\boldsymbol{\phi}_m^\top$ ▷ Proposition 1

      Embed the test set: $f_\Phi(\mathcal{D}_\tau^{\text{test}})$
      Compute the partial derivatives: $\partial_{\boldsymbol{W}(T)}\mathcal{L}_\tau^{\text{train}}$ & $\partial_{\boldsymbol{W}(T)}\mathcal{L}_\tau^{\text{test}}$
      $\nabla_\tau \boldsymbol{W}_0 \leftarrow \textsc{Project}_{\boldsymbol{W}_0}\big(\partial_{\boldsymbol{W}(T)}\mathcal{L}_\tau^{\text{test}}, \boldsymbol{B}_T, \boldsymbol{\phi}\big)$ ▷ Proposition 5
      $\nabla_\tau \boldsymbol{\phi}_m \leftarrow \textsc{Project}_{\boldsymbol{\phi}_m}\big(\partial_{\boldsymbol{W}(T)}\mathcal{L}_\tau^{\text{test}}, \boldsymbol{s}(T), \boldsymbol{B}_T, \boldsymbol{z}_T, \boldsymbol{\phi}\big) + \partial_{\boldsymbol{\phi}_m}\mathcal{L}_\tau^{\text{test}}$ ▷ Proposition 6
      $\nabla_\tau \Phi \leftarrow$ Backpropagation through f_Φ, starting with $\nabla_\tau \boldsymbol{\phi}_m$ for all m
      $\nabla_\tau T \leftarrow \big(\partial_{\boldsymbol{W}(T)}\mathcal{L}_\tau^{\text{test}}\big)^\top \big(\partial_{\boldsymbol{W}(T)}\mathcal{L}_\tau^{\text{train}}\big)$ ▷ Proposition 4
    end for
    Update the meta-parameters:
      $\boldsymbol{W}_0 \leftarrow \boldsymbol{W}_0 - \frac{\alpha}{|\mathcal{B}|}\sum_{\tau\in\mathcal{B}} \nabla_\tau \boldsymbol{W}_0$
      $\Phi \leftarrow \Phi - \frac{\alpha}{|\mathcal{B}|}\sum_{\tau\in\mathcal{B}} \nabla_\tau \Phi$
      $T \leftarrow T - \frac{\alpha}{|\mathcal{B}|}\sum_{\tau\in\mathcal{B}} \nabla_\tau T$
  end loop

The dynamical system followed during adaptation is given in Propositions 1 to 3 and is summarized in Algorithm 2. The solution of this dynamical system is used not only to find the adapted parameters

$\boldsymbol{W}_\tau(T)$, but also to get the quantities necessary to compute the meta-gradients for the updates of the meta-parameters $\boldsymbol{W}_0$ & Φ. The integration of this dynamical system, named ODESOLVE in Algorithm 1, is done in practice using a numerical solver such as Runge-Kutta methods.

Algorithm 2 Dynamical system

function DYNAMICS($\boldsymbol{W}_0, f_\Phi(\mathcal{D}_\tau^{\text{train}}), \boldsymbol{s}(t)$)

$\boldsymbol{W}_\tau(t) \leftarrow \boldsymbol{W}_0 - \sum_m \boldsymbol{s}_m(t)\boldsymbol{\phi}_m^\top$ ▷ Proposition 1

$\boldsymbol{p}_m(t) \leftarrow \text{softmax}(\boldsymbol{W}_\tau(t)\boldsymbol{\phi}_m)$

$\boldsymbol{A}_m(t) \leftarrow \big(\text{diag}(\boldsymbol{p}_m(t)) - \boldsymbol{p}_m(t)\boldsymbol{p}_m(t)^\top\big)/M$

$\dfrac{d\boldsymbol{s}_m}{dt} \leftarrow \dfrac{1}{M}\big(\boldsymbol{p}_m(t) - \boldsymbol{y}_m\big)$ ▷ Proposition 1

$\dfrac{d\boldsymbol{B}_t[i,j]}{dt} \leftarrow \mathbb{1}(i=j)\boldsymbol{A}_i(t) - \boldsymbol{A}_i(t)\displaystyle\sum_{m=1}^{M}\big(\boldsymbol{\phi}_i^\top\boldsymbol{\phi}_m\big)\boldsymbol{B}_t[m,j]$ ▷ Propositions 2 & 3 ($\hookleftarrow$)

$\dfrac{d\boldsymbol{z}_t[i,j,m]}{dt} \leftarrow -\boldsymbol{A}_i(t)\Big[\mathbb{1}(i=j)\boldsymbol{s}_m(t) + \mathbb{1}(i=m)\boldsymbol{s}_j(t) + \displaystyle\sum_{k=1}^{M}\big(\boldsymbol{\phi}_i^\top\boldsymbol{\phi}_k\big)\boldsymbol{z}_t[k,j,m]\Big]$

return $d\boldsymbol{s}/dt, d\boldsymbol{B}_t/dt, d\boldsymbol{z}_t/dt$

end function

In Algorithms 3 & 4, we give the procedures responsible for the projection of the partial derivatives $\partial_{\boldsymbol{W}(T)}\mathcal{L}_\tau^{\text{test}}$ onto the Jacobian matrices $d\boldsymbol{W}(T)/d\boldsymbol{W}_0$ and $d\boldsymbol{W}(T)/d\boldsymbol{\phi}_m$ respectively, based on Propositions 5 & 6. Note that Algorithm 4 also depends on the initial conditions $\boldsymbol{W}_0$, which is implicitly assumed here for clarity of presentation. It is interesting to see that both functions use the same matrix $\boldsymbol{C}$, and therefore this can be computed only once and reused for both projections.

Algorithm 3 Projection onto $d\boldsymbol{W}(T)/d\boldsymbol{W}_0$

function PROJECT$_{\boldsymbol{W}_0}$($\boldsymbol{V}(T), \boldsymbol{B}_T, \boldsymbol{\phi}$)

$\boldsymbol{C}_j \leftarrow \displaystyle\sum_{i=1}^{M}\boldsymbol{\phi}_i^\top\boldsymbol{V}(T)^\top\boldsymbol{B}_T[i,j]$

return $\boldsymbol{V}(T) - \boldsymbol{C}^\top\boldsymbol{\phi}$

end function

Algorithm 4 Projection onto $d\boldsymbol{W}(T)/d\boldsymbol{\phi}_m$

function PROJECT$_{\boldsymbol{\phi}_m}$($\boldsymbol{V}(T), \boldsymbol{s}(T), \boldsymbol{B}_T, \boldsymbol{z}_T, \boldsymbol{\phi}$)

$\boldsymbol{C}_j \leftarrow \displaystyle\sum_{i=1}^{M}\boldsymbol{\phi}_i^\top\boldsymbol{V}(T)^\top\boldsymbol{B}_T[i,j]$

$\boldsymbol{D}_{m,j} \leftarrow \displaystyle\sum_{i=1}^{M}\boldsymbol{z}_T[i,j,m]^\top\boldsymbol{V}(T)\boldsymbol{\phi}_i$

return $-\big[\boldsymbol{s}_m(T)^\top\boldsymbol{V}(T) + \boldsymbol{C}_m\boldsymbol{W}_0 + \boldsymbol{D}_m\boldsymbol{\phi}\big]$

end function

Finally in Algorithms 5 & 6, we show how to use COMLN at meta-test time on a novel task, based on the learned meta-parameters $\boldsymbol{W}_0$, Φ and T. This simply corresponds to integrating a dynamical system in $\boldsymbol{W}(t)$ (equivalently, in $\boldsymbol{s}_m(t)$, see Proposition 1). Note that for adaptation during meta-testing, there is no need to compute either $\boldsymbol{B}_t[i,j]$ or $\boldsymbol{z}_t[i,j,m]$, since these are only necessary to compute the meta-gradients during meta-training.

Algorithm 5 COMLN – Meta-test

Require: A task τ with a dataset $\mathcal{D}_\tau^{\text{train}}$

Require: Meta-parameters Φ, $\boldsymbol{W}_0$ & T

Embed the training dataset: $f_\Phi(\mathcal{D}_\tau^{\text{train}})$

$\boldsymbol{s}(T) \leftarrow \text{ODESolve}(\boldsymbol{0}, \text{DYNAMICS}, 0, T)$

$\boldsymbol{W}_\tau(T) \leftarrow \boldsymbol{W}_0 - \sum_m \boldsymbol{s}_m(T)\boldsymbol{\phi}_m^\top$

return $\boldsymbol{W}_\tau(T) \circ f_\Phi$

Algorithm 6 Dynamical system for adaptation

function DYNAMICS($\boldsymbol{W}_0, f_\Phi(\mathcal{D}_\tau^{\text{train}}), \boldsymbol{s}(t)$)

$\boldsymbol{W}_\tau(t) \leftarrow \boldsymbol{W}_0 - \sum_m \boldsymbol{s}_m(t)\boldsymbol{\phi}_m^\top$

$\boldsymbol{p}_m(t) \leftarrow \text{softmax}(\boldsymbol{W}_\tau(t)\boldsymbol{\phi}_m)$

$d\boldsymbol{s}_m/dt \leftarrow \big(\boldsymbol{p}_m(t) - \boldsymbol{y}_m\big)/M$

return $d\boldsymbol{s}/dt$

end function

A.2 SOURCE CODE

We provide a snippet of code written in JAX (Bradbury et al., 2018) in order to compute the adapted parameters, based on Algorithms 1 & 2. The `dynamics` function not only computes the vectors $\boldsymbol{s}_m(t)$ necessary to compute $\boldsymbol{W}(t)$ (see Appendix B.1), but also $\boldsymbol{B}_t[i,j]$ and $\boldsymbol{z}_t[i,j,m]$ jointly in order to compute the meta-gradients. This snippet shows that various necessary quantities can be precomputed ahead of adaptation, such as the Gram matrix `gram`.

```
import jax.numpy as jnp

from jax import vmap, grad, nn, ops, tree_util
from jax.experimental import ode
from collections import namedtuple

State = namedtuple('State', ['s', 'B', 'z'])

M, N = train_inputs.shape[0], train_labels.shape[1]
gram = jnp.matmul(train_inputs, train_inputs.T)
logits_0 = jnp.matmul(train_inputs, W_0.T)
diag = jnp.diag_indices(M)

def dynamics(state, _):
    preds = nn.softmax(logits_0 - jnp.matmul(gram, state.s), axis=1)
    A = (vmap(jnp.diag)(preds) - vmap(jnp.outer)(preds, preds)) / M

    # Update of s
    ds = (preds - train_labels) / M

    # Update of B
    cross_prod = jnp.einsum('ikn,im,mjnl->ijkl', A, gram, state.B)
    dB = ops.index_add(-cross_prod, diag, A)

    # Update of z
    cross_prod = jnp.einsum('iln,ik,jmn->ijml', A, gram, state.z)
    A_s = jnp.einsum('ikl,jl->ijk', A, state.s)
    dz = ops.index_add(cross_prod, diag, A_s)
    dz = ops.index_add(dz, (diag[0], None, diag[1]), A_s)

    return State(s=ds, B=dB, z=-dz)

state_0 = State(
    s=jnp.zeros((M, N)),
    B=jnp.zeros((M, M, N, N)),
    z=jnp.zeros((M, M, M, N))
)
solution = ode.odeint(dynamics, state_0, jnp.array([0., T]))
state_T = tree_util.tree_map(lambda x: x[-1], solution)
W_T = W_0 - jnp.matmul(state_T.s.T, train_inputs)
```

Code Snippet 1: Snippet of code in JAX to compute the adapted parameter `W_T` for a given task specified by the embedded training set `train_inputs` & `train_labels` (note that here `train_inputs` are the embedding vectors returned by f_Φ) and an initialization `W_0`.

We want to emphasize that all the information required to compute the meta-gradients are available in `state_T` found after integration of the `dynamics` function forward in time, and does not require any additional backward pass (apart from backpropagation through the embedding network f_Φ). In particular, the meta-gradients wrt. the initial conditions $\boldsymbol{W}_0$ and the wrt. the time horizon T can be computed using the snippet of code available in Code Snippet 2.

```
grads_test = grad(cross_entropy_loss)(W_T, test_inputs, test_labels)
grads_train = grad(cross_entropy_loss)(W_T, train_inputs, train_labels)

C = jnp.einsum('in,kn,ijkl->jl', train_inputs, grads_test, state_T.B)

grads_W_0 = grads_test - jnp.matmul(C.T, train_inputs)
grads_T = -jnp.vdot(grads_train, grads_test)
```

Code Snippet 2: Snippet of code in JAX to compute the meta-gradients wrt. the meta-parameters `W_0` and `T`, based on `state_T` found above. See Appendix B.4 & Appendix C for details.

The code to compute the meta-gradients wrt. the meta-parameters of the embedding network Φ is not included here for clarity, as it involves non-minimal dependencies on Haiku (Hennigan et al., 2020) in order to backpropagate the error through the backbone network.

B PROOFS OF MEMORY EFFICIENT META-GRADIENTS

As mentioned in Section 4, the core of computing the meta-gradient efficiently is based on the decomposition of the gradient (as well as the Hessian matrix) of the cross-entropy loss (Böhning, 1992). We recall this decomposition as the following lemma:

Lemma 1 (Böhning (1992))**.** *Let $f_\Phi(\mathcal{D}_\tau^{\text{train}}) = \{(\boldsymbol{\phi}_m, \boldsymbol{y}_m)\}_{m=1}^M$ be the embedded training set through the embedding network f_Φ, and $\mathcal{L}$ the cross-entropy loss defined as*

$$\mathcal{L}\big(\boldsymbol{W}; f_\Phi(\mathcal{D}_\tau^{\text{train}})\big) = -\frac{1}{M}\sum_{m=1}^{M} \boldsymbol{y}_m^\top \log \boldsymbol{p}_m \qquad \textit{where} \quad \boldsymbol{p}_m = \operatorname{softmax}\big(\boldsymbol{W}\boldsymbol{\phi}_m\big).$$

The gradient $\nabla\mathcal{L}$ and the Hessian matrix $\nabla^2\mathcal{L}$ of the cross-entropy loss can be written as

$$\nabla\mathcal{L}\big(\boldsymbol{W}; f_\Phi(\mathcal{D}_\tau^{\text{train}})\big) = \frac{1}{M}\sum_{m=1}^{M} \big(\boldsymbol{p}_m - \boldsymbol{y}_m\big)\boldsymbol{\phi}_m^\top$$

$$\nabla^2\mathcal{L}\big(\boldsymbol{W}; f_\Phi(\mathcal{D}_\tau^{\text{train}})\big) = \sum_{m=1}^{M} \boldsymbol{A}_m \otimes \boldsymbol{\phi}_m\boldsymbol{\phi}_m^\top,$$

where $\boldsymbol{A}_m = \big(\operatorname{diag}(\boldsymbol{p}_m) - \boldsymbol{p}_m\boldsymbol{p}_m^\top\big)/M$ are $N \times N$ matrices, and $\otimes$ is the Kronecker product.

This lemma is particularly useful in the context of few-shot learning since it reduces the characterization of the gradient and the Hessian from quantities of size $N \times d$ and $Nd \times Nd$ respectively (where d is the dimension of the embedding vectors $\boldsymbol{\phi}$) to M objects whose size is independent of d—the embedding vectors $\boldsymbol{\phi}$ being independent of $\boldsymbol{W}$. Typically in few-shot learning, M is much smaller than d.

In order to avoid higher-order tensors when defining the different Jacobians and Hessians, we always consider them as matrices, with possibly a flattening operation. For example here even though $\boldsymbol{W}$ is a $N \times d$ matrix, we treat the Hessian $\nabla^2\mathcal{L}$ as a $Nd \times Nd$ matrix, as opposed to a 4D tensor. When the context is required, we will make this transformation from a higher-order tensor to a matrix explicit with an encoding of indices. Moreover throughout this section, we will only consider the computation of the meta-gradients for a single task τ, and therefore we will often drop the explicit dependence of the different objects on τ (e.g. we will write $\boldsymbol{W}(t)$ instead of $\boldsymbol{W}_\tau(t)$); the meta-gradients are eventually averaged over a batch of tasks for the update of the outer-loop, see Appendix A for details in the pseudo-code. Finally, since we only consider adaptation of the last layer of the neural network, the presentation here is always made in the context of an embedded training set $f_\Phi(\mathcal{D}_\tau^{\text{train}}) = \{(\boldsymbol{\phi}_m, \boldsymbol{y}_m)\}_{m=1}^M$ that went through the backbone f_Φ.

B.1 DECOMPOSITION OF $\boldsymbol{W}(t)$ FOR PARAMETER ADAPTATION

As a direct application of Lemma 1, we first decompose the parameters $\boldsymbol{W}(t)$ into smaller quantities $\boldsymbol{s}_m(t)$ that follow the dynamics defined in Proposition 1. Although this decomposition is equivalent

to $\boldsymbol{W}(t)$, in practice solving a smaller dynamical system in $\boldsymbol{s}_m(t)$ improves the efficiency of our method.

Proposition 1. *Let $f_\Phi(\mathcal{D}_\tau^{\text{train}}) = \{(\boldsymbol{\phi}_m, \boldsymbol{y}_m)\}_{m=1}^M$ be the embedded training set through the embedding network f_Φ, and $\boldsymbol{W}(t)$ be the solution of the following dynamical system*

$$\frac{d\boldsymbol{W}}{dt} = -\nabla\mathcal{L}\big(\boldsymbol{W}(t); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big) \qquad \boldsymbol{W}(0) = \boldsymbol{W}_0,$$

where the loss function is the cross-entropy loss. The solution $\boldsymbol{W}(t)$ of this dynamical system can be written as

$$\boldsymbol{W}(t) = \boldsymbol{W}_0 - \sum_{m=1}^M \boldsymbol{s}_m(t)\boldsymbol{\phi}_m^\top,$$

where for all m, $\boldsymbol{s}_m(t)$ is the solution of the following dynamical system:

$$\frac{d\boldsymbol{s}_m}{dt} = \frac{1}{M}\big(\boldsymbol{p}_m(t) - \boldsymbol{y}_m\big) \qquad \boldsymbol{s}_m(0) = \boldsymbol{0},$$

and $\boldsymbol{p}_m(t) = \mathrm{softmax}(\boldsymbol{W}(t)\boldsymbol{\phi}_m)$ is the vector of predictions returned by the network at time t.

Proof. Recall from Lemma 1 that the gradient of the cross-entropy loss can be written as

$$\nabla\mathcal{L}\big(\boldsymbol{W}(t); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big) = \frac{1}{M}\sum_{m=1}^M \big(\boldsymbol{p}_m(t) - \boldsymbol{y}_m\big)\boldsymbol{\phi}_m^\top = \sum_{m=1}^M \frac{d\boldsymbol{s}_m(t)}{dt}\boldsymbol{\phi}_m^\top.$$

Using this form, we can integrate $d\boldsymbol{W}/dt$ to obtain

$$\begin{aligned}\boldsymbol{W}(t) &= \boldsymbol{W}_0 - \int_0^t \nabla\mathcal{L}\big(\boldsymbol{W}(t'); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big)\,dt' = \boldsymbol{W}_0 - \sum_{m=1}^M \left[\int_0^t \frac{d\boldsymbol{s}_m(t')}{dt'}\,dt'\right]\boldsymbol{\phi}_m^\top \\ &= \boldsymbol{W}_0 - \sum_{m=1}^M \big[\boldsymbol{s}_m(t) - \underbrace{\boldsymbol{s}_m(0)}_{=\boldsymbol{0}}\big]\boldsymbol{\phi}_m^\top = \boldsymbol{W}_0 - \sum_{m=1}^M \boldsymbol{s}_m(t)\boldsymbol{\phi}_m^\top\end{aligned}$$

□

B.2 Decomposition of the Jacobian matrix $d\boldsymbol{W}(t)/d\boldsymbol{W}_0$

The core objective of gradient-based meta-learning methods is the capacity to compute the meta-gradients wrt. the initial conditions $\boldsymbol{W}_0$. In order to compute this meta-gradient using forward-mode automatic differentiation, we want to first compute the Jacobian matrix $d\boldsymbol{W}(t)/d\boldsymbol{W}_0$. We show that this Jacobian can be decomposed into smaller quantities that follow the dynamics in Proposition 2.

Proposition 2. *Let $f_\Phi(\mathcal{D}_\tau^{\text{train}}) = \{(\boldsymbol{\phi}_m, \boldsymbol{y}_m)\}_{m=1}^M$ be the embedded training set through the embedding network f_Φ. The Jacobian matrix $d\boldsymbol{W}(t)/d\boldsymbol{W}_0$ can be written as*

$$\frac{d\boldsymbol{W}(t)}{d\boldsymbol{W}_0} = \boldsymbol{I} - \sum_{i=1}^M\sum_{j=1}^M \boldsymbol{B}_t[i,j] \otimes \boldsymbol{\phi}_i\boldsymbol{\phi}_j^\top,$$

where $\otimes$ is the Kronecker product, and for all i, j, $\boldsymbol{B}_t[i,j]$ is a $N \times N$ matrix which the solution of the following dynamical system

$$\frac{d\boldsymbol{B}_t[i,j]}{dt} = \mathbb{1}(i=j)\boldsymbol{A}_i(t) - \boldsymbol{A}_i(t)\sum_{m=1}^M \big(\boldsymbol{\phi}_i^\top\boldsymbol{\phi}_m\big)\boldsymbol{B}_t[m,j] \qquad \boldsymbol{B}_0[i,j] = \boldsymbol{0},$$

and $\boldsymbol{A}_i(t) = \big(\mathrm{diag}(\boldsymbol{p}_i(t)) - \boldsymbol{p}_i(t)\boldsymbol{p}_i(t)^\top\big)/M$ are defined using the vectors of predictions at time t $\boldsymbol{p}_i(t) = \mathrm{softmax}\big(\boldsymbol{W}(t)\boldsymbol{\phi}_i\big)$.

Proof. We will use the forward sensitivity equation from Section 2.2 in order to derive this new dynamical system over $\boldsymbol{B}_t[i,j]$. Recall that to simplify the notations we can write the gradient vector field followed during adaptation as

$$\frac{d\boldsymbol{W}}{dt} = g\big(\boldsymbol{W}(t); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big) \triangleq -\nabla\mathcal{L}\big(\boldsymbol{W}(t); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big).$$

Introducing the matrix-valued function $\mathcal{S}(t) = d\boldsymbol{W}(t)/d\boldsymbol{W}_0$ as a sensitivity state, we can use the forward sensitivity equations and see that $\mathcal{S}(t)$ satisfies the following equation

$$\begin{aligned}\frac{d\mathcal{S}}{dt} &= \frac{\partial g\big(\boldsymbol{W}(t)\big)}{\partial \boldsymbol{W}(t)}\mathcal{S}(t) + \cancel{\frac{\partial g\big(\boldsymbol{W}(t)\big)}{\partial \boldsymbol{W}_0}} \qquad\qquad \mathcal{S}(0) = \boldsymbol{I}\\ &= -\nabla^2\mathcal{L}\big(\boldsymbol{W}(t); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big)\mathcal{S}(t),\end{aligned}$$

the partial derivative wrt. $\boldsymbol{W}_0$ being simplified since the function g does not depend directly on the initial conditions $\boldsymbol{W}_0$. The rest of the proof is based on the unicity of the solution of a given autonomous[5] differential equation given a particular choice of initial conditions (Wiggins, 2003, Prop. 7.4.2). In other words, if we can find another function $\widetilde{\mathcal{S}}(t)$ that also satisfies the above differential equation with the initial conditions $\widetilde{\mathcal{S}}(0) = \boldsymbol{I}$, then it means that for all t we have $\widetilde{\mathcal{S}}(t) = \mathcal{S}(t)$. Suppose that this function can be written as

$$\widetilde{\mathcal{S}}(t) = \boldsymbol{I} - \sum_{i=1}^{M}\sum_{j=1}^{M} \boldsymbol{B}_t[i,j] \otimes \boldsymbol{\phi}_i\boldsymbol{\phi}_j^\top,$$

where $\boldsymbol{B}_t[i,j]$ satisfies the dynamical system defined in the statement of Proposition 2. Then we have, using Lemma 1:

$$\begin{aligned}-\nabla^2\mathcal{L}\big(\boldsymbol{W}(t); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big)\widetilde{\mathcal{S}}(t) &= -\Big[\sum_{i=1}^{M}\boldsymbol{A}_i(t)\otimes\boldsymbol{\phi}_i\boldsymbol{\phi}_i^\top\Big]\Big[\boldsymbol{I} - \sum_{m,j}\boldsymbol{B}_t[m,j]\otimes\boldsymbol{\phi}_m\boldsymbol{\phi}_j^\top\Big]\\ &= -\sum_{i=1}^{M}\boldsymbol{A}_i(t)\otimes\boldsymbol{\phi}_i\boldsymbol{\phi}_i^\top + \sum_{i,j,m}\big(\boldsymbol{A}_i(t)\boldsymbol{B}_t[m,j]\big)\otimes\big(\boldsymbol{\phi}_i\underbrace{\boldsymbol{\phi}_i^\top\boldsymbol{\phi}_m}_{\in\mathbb{R}}\boldsymbol{\phi}_j^\top\big)\\ &= -\sum_{i,j}\Big[\mathbb{1}(i=j)\boldsymbol{A}_i(t) - \boldsymbol{A}_i(t)\sum_{m=1}^{M}\big(\boldsymbol{\phi}_i^\top\boldsymbol{\phi}_m\big)\boldsymbol{B}_t[m,j]\Big]\otimes\boldsymbol{\phi}_i\boldsymbol{\phi}_j^\top\\ &= -\sum_{i,j}\frac{d\boldsymbol{B}_t[i,j]}{dt}\otimes\boldsymbol{\phi}_i\boldsymbol{\phi}_j^\top = \frac{d\widetilde{\mathcal{S}}}{dt}\end{aligned}$$

We have shown that $\widetilde{\mathcal{S}}(t)$ follows the same dynamics as $\mathcal{S}(t)$. Moreover, using the initial conditions $\boldsymbol{B}_0[i,j] = \boldsymbol{0}$, it is clear that $\widetilde{\mathcal{S}}(0) = \boldsymbol{I}$, which are the same initial conditions as the equation satisfied by $\mathcal{S}(t)$. Therefore, we have $\widetilde{\mathcal{S}}(t) = \mathcal{S}(t)$ for all t, showing the expected decomposition of $\mathcal{S}(t) = d\boldsymbol{W}(t)/d\boldsymbol{W}_0$. □

B.3 Decomposition of the Jacobian matrix $d\boldsymbol{W}(t)/d\boldsymbol{\phi}_m$

Similar to Proposition 2, there exists a decomposition of the Jacobian matrix $d\boldsymbol{W}(T)/d\boldsymbol{\phi}_m$. Recall that this Jacobian matrix appears in the computation of the gradient of the outer-loss wrt. the embedding vectors $\boldsymbol{\phi}_m$, which is necessary in order to compute the meta-gradients wrt. the meta-parameters of the embedding network f_Φ using backpropagation from the last layer of f_Φ.

Proposition 3. *Let $f_\Phi(\mathcal{D}_\tau^{\text{train}}) = \{(\boldsymbol{\phi}_m, \boldsymbol{y}_m)\}_{m=1}^{M}$ be the embedded training set through the embedding network f_Φ. For all m, the Jacobian matrix $d\boldsymbol{W}(t)/d\boldsymbol{\phi}_m$ can be written as*

$$\frac{d\boldsymbol{W}(t)}{d\boldsymbol{\phi}_m} = -\Big[\boldsymbol{s}_m(t)\otimes\boldsymbol{I}_d + \sum_{i=1}^{M}\boldsymbol{B}_t[i,m]\boldsymbol{W}_0\otimes\boldsymbol{\phi}_i + \sum_{i=1}^{M}\sum_{j=1}^{M}\boldsymbol{z}_t[i,j,m]\boldsymbol{\phi}_j^\top\otimes\boldsymbol{\phi}_i\Big],$$

[5] While the dynamical system defined here for the sensitivity state $\mathcal{S}(t)$ alone is not exactly autonomous due to the dependence of the Hessian matrix on $\boldsymbol{W}(t)$, we could augment $\mathcal{S}(t)$ with $\boldsymbol{W}(t)$ to obtain an autonomous system on the augmented state. We come back to this distinction in Appendix D. The unicity argument still holds here (the augmented solution would be unique).

where $\boldsymbol{s}_m(t)$ and $\boldsymbol{B}_t[i,j]$ are the solutions of the ODEs defined in Propositions 1 & 2, and for all i,j,m, $\boldsymbol{z}_t[i,j,m]$ is a vector of length N solution of the following dynamical system:

$$\frac{d\boldsymbol{z}_t[i,j,m]}{dt} = -\boldsymbol{A}_i(t)\Big[\mathbb{1}(i=j)\boldsymbol{s}_m(t) + \mathbb{1}(i=m)\boldsymbol{s}_j(t) + \sum_{k=1}^{M}\big(\boldsymbol{\phi}_i^\top\boldsymbol{\phi}_k\big)\boldsymbol{z}_t[k,j,m]\Big],$$

with the initial conditions $\boldsymbol{z}_0[i,j,m] = \mathbf{0}$.

Proof. The outline of this proof follows the proof of Proposition 2: we first use the forward sensitivity equations to get the dynamical system followed by the Jacobian matrix $d\boldsymbol{W}(t)/d\boldsymbol{\phi}_m$, and then use a unicity argument given that the new decomposition satisfies the same ODE. Recall that we use the following notation to write the gradient vector field followed during adaptation:

$$\frac{d\boldsymbol{W}}{dt} = g\big(\boldsymbol{W}(t); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big) \triangleq -\nabla\mathcal{L}\big(\boldsymbol{W}(t); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big).$$

For a fixed m, we introduce the matrix valued function $\mathcal{S}(t) = d\boldsymbol{W}(t)/d\boldsymbol{\phi}_m$ as a sensitivity state. We can use the forward sensitivity equations, and see that $\mathcal{S}(t)$ satisfies the following equation

$$\begin{aligned}\frac{d\mathcal{S}}{dt} &= \frac{\partial g\big(\boldsymbol{W}(t),\boldsymbol{\phi}_m\big)}{\partial \boldsymbol{W}(t)}\mathcal{S}(t) + \frac{\partial g\big(\boldsymbol{W}(t),\boldsymbol{\phi}_m\big)}{\partial \boldsymbol{\phi}_m} \qquad\qquad \mathcal{S}(0) = \frac{d\boldsymbol{W}(0)}{d\boldsymbol{\phi}_m} = \mathbf{0}\\ &= -\nabla^2\mathcal{L}\big(\boldsymbol{W}(t); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big)\mathcal{S}(t) - \boldsymbol{A}_m(t)\boldsymbol{W}_0\otimes\boldsymbol{\phi}_m\\ &\quad + \boldsymbol{A}_m(t)\sum_{i=1}^{M}\boldsymbol{s}_i(t)\boldsymbol{\phi}_i^\top\otimes\boldsymbol{\phi}_m - \frac{1}{M}\big(\boldsymbol{p}_m(t)-\boldsymbol{y}_m\big)\otimes\boldsymbol{I}_d\end{aligned}$$

where we make the direct dependence of g on $\boldsymbol{\phi}_m$ explicit, and we used the decomposition of $\boldsymbol{W}(t)$ from Proposition 1. Suppose that we define the function $\widetilde{\mathcal{S}}(t)$ as

$$\widetilde{\mathcal{S}}(t) = -\Big[\boldsymbol{s}_m(t)\otimes\boldsymbol{I}_d + \sum_{i=1}^{M}\boldsymbol{B}_t[i,m]\boldsymbol{W}_0\otimes\boldsymbol{\phi}_i + \sum_{i,j}\boldsymbol{z}_t[i,j,m]\boldsymbol{\phi}_j^\top\otimes\boldsymbol{\phi}_i\Big],$$

where $\boldsymbol{z}_t[i,j,m]$ satisfies the dynamical system defined in the statement of Proposition 3. Then we have, using Lemma 1:

$$\begin{aligned}&-\nabla^2\mathcal{L}\big(\boldsymbol{W}(t); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big)\widetilde{\mathcal{S}}(t) - \boldsymbol{A}_m(t)\Big[\boldsymbol{W}_0 - \sum_{i=1}^{M}\boldsymbol{s}_i(t)\boldsymbol{\phi}_i^\top\Big]\otimes\boldsymbol{\phi}_m - \frac{1}{M}\big(\boldsymbol{p}_m(t)-\boldsymbol{y}_m\big)\otimes\boldsymbol{I}_d\\ &= \Big[\sum_{i=1}^{M}\boldsymbol{A}_i(t)\otimes\boldsymbol{\phi}_i\boldsymbol{\phi}_i^\top\Big]\Big[\boldsymbol{s}_m(t)\otimes\boldsymbol{I}_d + \sum_{i=1}^{M}\boldsymbol{B}_t[i,m]\boldsymbol{W}_0\otimes\boldsymbol{\phi}_i + \sum_{i,j}\boldsymbol{z}_t[i,j,m]\boldsymbol{\phi}_j^\top\otimes\boldsymbol{\phi}_i\Big]\\ &\qquad - \boldsymbol{A}_m(t)\boldsymbol{W}_0\otimes\boldsymbol{\phi}_m + \boldsymbol{A}_m(t)\sum_{i=1}^{M}\boldsymbol{s}_i(t)\boldsymbol{\phi}_i^\top\otimes\boldsymbol{\phi}_m - \frac{1}{M}\big(\boldsymbol{p}_m(t)-\boldsymbol{y}_m\big)\otimes\boldsymbol{I}_d\\ &= -\underbrace{\frac{1}{M}\big(\boldsymbol{p}_m(t)-\boldsymbol{y}_m\big)}_{=d\boldsymbol{s}_m/dt}\otimes\boldsymbol{I}_d - \sum_{i=1}^{M}\Big[\underbrace{\mathbb{1}(i=m)\boldsymbol{A}_i(t) - \boldsymbol{A}_i(t)\sum_{k=1}^{M}\big(\boldsymbol{\phi}_i^\top\boldsymbol{\phi}_k\big)\boldsymbol{B}_t[k,m]}_{=d\boldsymbol{B}_t[i,m]/dt}\Big]\boldsymbol{W}_0\otimes\boldsymbol{\phi}_i\\ &\qquad + \sum_{i,j}\underbrace{\boldsymbol{A}_i(t)\Big[\mathbb{1}(i=j)\boldsymbol{s}_m(t) + \mathbb{1}(i=m)\boldsymbol{s}_j(t) + \sum_{k=1}^{M}\big(\boldsymbol{\phi}_i^\top\boldsymbol{\phi}_k\big)\boldsymbol{z}_t[k,j,m]\Big]}_{=-d\boldsymbol{z}_t[i,j,m]/dt}\boldsymbol{\phi}_j^\top\otimes\boldsymbol{\phi}_i\\ &= \frac{d\widetilde{\mathcal{S}}}{dt}\end{aligned}$$

We have shown that $\widetilde{\mathcal{S}}(t)$ follows the same dynamics as $\mathcal{S}(t)$. Moreover using the initial conditions $\boldsymbol{s}_m(0) = \mathbf{0}$, $\boldsymbol{B}_0[i,j] = \mathbf{0}$, and $\boldsymbol{z}_0[i,j,m] = \mathbf{0}$, we have $\widetilde{\mathcal{S}}(0) = \mathbf{0}$, which are the same initial conditions as the equation satisfied by $\mathcal{S}(t)$. Therefore we have $\widetilde{\mathcal{S}}(t) = \mathcal{S}(t)$ for all t, showing the expected decomposition of the Jacobian matrix $\mathcal{S}(t) = d\boldsymbol{W}(t)/d\boldsymbol{\phi}_m$. □

B.4 PROOF OF THE META-GRADIENT WRT. T

Proposition 4. *Let $f_\Phi(\mathcal{D}_\tau^{\text{train}})$ and $f_\Phi(\mathcal{D}_\tau^{\text{test}})$ be the embedding through the network f_Φ of the training and test set respectively. The gradient of the outer-loss wrt. the time horizon T is given by:*

$$\frac{d\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{dT} = -\left[\frac{\partial\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{\partial\boldsymbol{W}(T)}\right]^\top \frac{\partial\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big)}{\partial\boldsymbol{W}(T)}.$$

Proof. We can directly apply the result from (Chen et al., 2018, App. B.2), which we recall here for completeness. Given the ODE $d\boldsymbol{z}/dt = g(\boldsymbol{z}(t))$, the gradient of the loss $\mathcal{L}\big(\boldsymbol{z}(T)\big)$ wrt. T is given by

$$\frac{d\mathcal{L}}{dT} = \boldsymbol{a}(T)^\top g(\boldsymbol{z}(T)) \qquad \text{where} \quad \boldsymbol{a}(T) = \frac{\partial\mathcal{L}}{\partial\boldsymbol{z}(T)}$$

is the *initial adjoint state* (see Section 2.2). In our case we have

$$g(\boldsymbol{z}(T)) \equiv -\nabla\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{train}})\big) \qquad \boldsymbol{a}(T) \equiv \frac{\partial\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{\partial\boldsymbol{W}(T)}.$$

□

C PROJECTION ONTO THE JACOBIAN MATRICES

Once we have computed the Jacobian $d\boldsymbol{W}(T)/d\boldsymbol{\theta}$ wrt. some meta-parameter $\boldsymbol{\theta}$ (in our case, either the initial conditions $\boldsymbol{W}_0$ or the embedding vectors $\boldsymbol{\phi}_m$), we only have to project the vector of partial derivatives onto it to obtain the meta-gradients (vector-Jacobian product), using the chain rule:

$$\frac{d\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{d\boldsymbol{\theta}} = \frac{\partial\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{\partial\boldsymbol{W}(T)}\frac{d\boldsymbol{W}(T)}{d\boldsymbol{\theta}} + \frac{\partial\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{\partial\boldsymbol{\theta}}.$$

While we have shown how to decompose the Jacobian matrices in such a way that it only involves quantities that are independent of d the dimension of the embedding vectors $\boldsymbol{\phi}$, this final projection a priori requires us to form the Jacobian explicitly. Even though this operation is done only once at the end of the integration, this may be very expensive since the Jacobian matrices scale quadratically in d, which can be as high as $d = 16$k in our experiments.

Fortunately, we can perform this projection as a function of $\boldsymbol{s}_m(T)$, $\boldsymbol{B}_T[i,j]$, and $\boldsymbol{z}_T[i,j,m]$, without having to explicitly form the full Jacobian matrix, by exchanging the order of operations.

Proposition 5. *Let $f_\Phi(\mathcal{D}_\tau^{\text{train}}) = \{(\boldsymbol{\phi}_m, \boldsymbol{y}_m)\}_{m=1}^M$ and $f_\Phi(\mathcal{D}_\tau^{\text{test}})$ be the embedded training set and test set through the embedding network f_Φ respectively. Let $\boldsymbol{B}_T[i,j]$ be the solution at time T of the differential equation defined in Proposition 2, and $\boldsymbol{V}(T) \in \mathbb{R}^{N\times d}$ the partial derivative of the outer-loss wrt. the adapted parameters $\boldsymbol{W}(T)$:*

$$\boldsymbol{V}(T) = \frac{\partial\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{\partial\boldsymbol{W}(T)} \quad \textit{so that} \quad \frac{d\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{d\boldsymbol{W}_0} = \text{Vec}\big(\boldsymbol{V}(T)\big)^T\frac{d\boldsymbol{W}(T)}{d\boldsymbol{W}_0}.$$

Let $\boldsymbol{\phi} = [\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_M]^\top \in \mathbb{R}^{M\times d}$ be the design matrix. The gradient of the outer-loss wrt. the initial conditions $\boldsymbol{W}_0$ can be computed as

$$\frac{d\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{d\boldsymbol{W}_0} = \boldsymbol{V}(T) - \boldsymbol{C}^\top\boldsymbol{\phi},$$

where the rows of $\boldsymbol{C} \in \mathbb{R}^{M\times N}$ are defined by

$$\boldsymbol{C}_j = \sum_{i=1}^M \boldsymbol{\phi}_i^\top \boldsymbol{V}(T)^\top \boldsymbol{B}_T[i,j].$$

Proof. In this proof, we will make the encoding of the indices for the Jacobian $d\boldsymbol{W}(T)/d\boldsymbol{W}_0$ more explicit, as mentioned in Appendix B. Recall from Proposition 2 that this Jacobian matrix can be decomposed as

$$\frac{d\boldsymbol{W}(T)}{d\boldsymbol{W}_0} = \boldsymbol{I} - \sum_{i,j} \boldsymbol{B}_T[i,j] \otimes \boldsymbol{\phi}_i \boldsymbol{\phi}_j^\top.$$

Introducing the following notations

$$\boldsymbol{F} \triangleq \sum_{i,j} \boldsymbol{B}_T[i,j] \otimes \boldsymbol{\phi}_i \boldsymbol{\phi}_j^\top \in \mathbb{R}^{Nd \times Nd} \qquad \boldsymbol{G} \triangleq \mathrm{Vec}\big(\boldsymbol{V}(T)\big)^\top \boldsymbol{F} \in \mathbb{R}^{Nd},$$

for all $l \in \{0, \dots, N-1\}$ and $y \in \{0, \dots, d-1\}$:

$$\begin{aligned}
\boldsymbol{G}[dl+y] &= \sum_{k=0}^{N-1} \sum_{x=0}^{d-1} \boldsymbol{V}_T[k,x] \boldsymbol{F}[dk+x, dl+y] \\
&= \sum_{i,j} \sum_{k=0}^{N-1} \Bigg[\underbrace{\sum_{x=0}^{d-1} \boldsymbol{V}_T[k,x] \boldsymbol{\phi}_i[x]}_{=[\boldsymbol{V}(T)\boldsymbol{\phi}_i]_k} \Bigg] \boldsymbol{B}_T[i,j,k,l] \boldsymbol{\phi}_j[y] \\
&= \sum_{i,j} \Bigg[\underbrace{\sum_{k=0}^{N-1} [\boldsymbol{V}(T)\boldsymbol{\phi}_i]_k \boldsymbol{B}_T[i,j,k,l]}_{=[\boldsymbol{\phi}_i^\top \boldsymbol{V}(T)^\top \boldsymbol{B}_T[i,j]]_l} \Bigg] \boldsymbol{\phi}_j[y] \\
&= \sum_{j=1}^{M} \Bigg[\underbrace{\sum_{i=1}^{M} \big[\boldsymbol{\phi}_i^\top \boldsymbol{V}(T)^\top \boldsymbol{B}_T[i,j]\big]_l}_{=\boldsymbol{C}_{j,l}} \Bigg] \boldsymbol{\phi}_j[y] = \big[\boldsymbol{C}^\top \boldsymbol{\phi}\big]_{l,y}
\end{aligned}$$

This shows that $\boldsymbol{G}$ is equal to $\boldsymbol{C}^\top \boldsymbol{\phi}$ up to reshaping. Using the full form of the Jacobian, including $\boldsymbol{I}$, concludes the proof. □

An interesting observation is that if the term $\boldsymbol{C}^\top \boldsymbol{\phi}$ is ignored in the computation of the meta-gradient, we recover an equivalent of the first-order approximation introduced in (Finn et al., 2017). Similarly, we can show that we can perform the projection onto the Jacobian matrix $d\boldsymbol{W}(T)/d\boldsymbol{\phi}_m$ without having to form the explicit $Nd \times d$ matrix.

Proposition 6. *Let $f_\Phi(\mathcal{D}_\tau^{\text{train}}) = \{(\boldsymbol{\phi}_m, \boldsymbol{y}_m)\}_{m=1}^M$ and $f_\Phi(\mathcal{D}_\tau^{\text{test}})$ be the embedded training set and test set through the embedding network f_Φ respectively. Let $\boldsymbol{s}_m(T)$ be the solution at time T of the differential equation defined in Proposition 1, $\boldsymbol{B}_T[i,j]$ the solution of the one defined in Proposition 2, and $\boldsymbol{z}_T[i,j,m]$ the solution of the one defined in Proposition 3. Let $\boldsymbol{V}(T) \in \mathbb{R}^{N \times d}$ be the partial derivative of the outer-loss wrt. the adapted parameters $\boldsymbol{W}(T)$:*

$$\boldsymbol{V}(T) = \frac{\partial \mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)}{\partial \boldsymbol{W}(T)}.$$

Let $\boldsymbol{\phi} = [\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_M]^\top \in \mathbb{R}^{M \times d}$ be the design matrix. The projection of $\boldsymbol{V}(T)$ onto the Jacobian matrix $d\boldsymbol{W}(T)/d\boldsymbol{\phi}_m$ can be computed as

$$\mathrm{Vec}\big(\boldsymbol{V}(T)\big)^\top \frac{d\boldsymbol{W}(T)}{d\boldsymbol{\phi}_m} = -\big[\boldsymbol{s}_m(T)^\top \boldsymbol{V}(T) + \boldsymbol{C}_m \boldsymbol{W}_0 + \boldsymbol{D}_m \boldsymbol{\phi}\big],$$

where $\boldsymbol{C}_m \in \mathbb{R}^N$ and $\boldsymbol{D}_m \in \mathbb{R}^M$ are defined by

$$\boldsymbol{C}_m = \sum_{i=1}^{M} \boldsymbol{\phi}_i^\top \boldsymbol{V}(T)^\top \boldsymbol{B}_T[i,m] \qquad \boldsymbol{D}_{m,j} = \sum_{i=1}^{M} \boldsymbol{z}_T[i,j,m]^\top \boldsymbol{V}(T) \boldsymbol{\phi}_i$$

Note that the definition of $\boldsymbol{C}$ in Proposition 6 matches exactly the definition of $\boldsymbol{C}$ in Proposition 5, and therefore we only need to compute this matrix once to perform the projections for both meta-gradients.

Proof. Recall from Proposition 3 that the Jacobian $d\boldsymbol{W}(T)/d\boldsymbol{\phi}_m$ can be decomposed as

$$\frac{d\boldsymbol{W}(T)}{d\boldsymbol{\phi}_m} = -\Big[\boldsymbol{s}_m(T) \otimes \boldsymbol{I}_d + \sum_{i=1}^{M} \boldsymbol{B}_T[i,m]\boldsymbol{W}_0 \otimes \boldsymbol{\phi}_i + \sum_{i,j} \boldsymbol{z}_T[i,j,m]\boldsymbol{\phi}_j^\top \otimes \boldsymbol{\phi}_i\Big].$$

We will consider each of these 3 terms individually.

- For the first term, let

$$\boldsymbol{F}_1 \triangleq \boldsymbol{s}_m(T) \otimes \boldsymbol{I}_d \in \mathbb{R}^{Nd\times d} \qquad \boldsymbol{G}_1 \triangleq \mathrm{Vec}\big(\boldsymbol{V}(T)\big)^\top \boldsymbol{F}_1 \in \mathbb{R}^d,$$

  and for $y \in \{0,\dots,d-1\}$:

$$\begin{aligned}\boldsymbol{G}_1[y] &= \sum_{k=0}^{N-1}\sum_{x=0}^{d-1} \boldsymbol{V}_T[k,x]\boldsymbol{F}_1[dk+x,y] = \sum_{k=0}^{N-1}\sum_{x=0}^{d-1} \boldsymbol{V}_T[k,x]\mathbb{1}(x=y)\big[\boldsymbol{s}_m(T)\big]_k \\ &= \sum_{k=0}^{N-1} \big[\boldsymbol{s}_m(T)\big]_k \boldsymbol{V}_T[k,y] = \big[\boldsymbol{s}_m(T)^\top \boldsymbol{V}(T)\big]_y\end{aligned}$$

  Therefore, $\boldsymbol{G}_1$ the projection of $\boldsymbol{V}(T)$ onto the first term of the Jacobian matrix is equal to $\boldsymbol{s}_m(T)^\top \boldsymbol{V}(T)$.

- For the second term, let

$$\boldsymbol{F}_2 \triangleq \sum_{i=1}^{M} \boldsymbol{B}_T[i,m]\boldsymbol{W}_0 \otimes \boldsymbol{\phi}_i \in \mathbb{R}^{Nd\times d} \qquad \boldsymbol{G}_2 \triangleq \mathrm{Vec}\big(\boldsymbol{V}(T)\big)^\top \boldsymbol{F}_2 \in \mathbb{R}^d,$$

  and for $y \in \{0,\dots,d-1\}$:

$$\begin{aligned}\boldsymbol{G}_2[y] &= \sum_{k=0}^{N-1}\sum_{x=0}^{d-1} \boldsymbol{V}_T[k,x]\boldsymbol{F}_2[dk+x,y] \\ &= \sum_{i=1}^{M}\sum_{l=0}^{N-1}\sum_{k=0}^{N-1} \Bigg[\underbrace{\sum_{x=0}^{d-1} \boldsymbol{V}_T[k,x]\boldsymbol{\phi}_i[x]}_{=[\boldsymbol{V}(T)\boldsymbol{\phi}_i]_k}\Bigg] \boldsymbol{B}_T[i,m,k,l]\boldsymbol{W}_0[l,y] \\ &= \sum_{i=1}^{M}\sum_{l=0}^{N-1} \Bigg[\underbrace{\sum_{k=0}^{N-1} [\boldsymbol{V}(T)\boldsymbol{\phi}_i]_k \boldsymbol{B}_T[i,m,k,l]}_{=\left[\boldsymbol{\phi}_i^\top \boldsymbol{V}(T)^\top \boldsymbol{B}_T[i,m]\right]_l}\Bigg] \boldsymbol{W}_0[l,y] \\ &= \sum_{l=0}^{N-1} \Bigg[\underbrace{\sum_{i=1}^{M} \big[\boldsymbol{\phi}_i^\top \boldsymbol{V}(T)^\top \boldsymbol{B}_T[i,m]\big]_l}_{=\boldsymbol{C}_{m,l}}\Bigg] \boldsymbol{W}_0[l,y] = \big[\boldsymbol{C}_m \boldsymbol{W}_0\big]_y\end{aligned}$$

  $\boldsymbol{G}_2$ the projection of $\boldsymbol{V}(T)$ onto the second term of the Jacobian matrix is equal to $\boldsymbol{C}_m\boldsymbol{W}_0$.

- Finally for the third term, let

$$\boldsymbol{F}_3 \triangleq \sum_{i,j} \boldsymbol{z}_T[i,j,m]\boldsymbol{\phi}_j^\top \otimes \boldsymbol{\phi}_i \in \mathbb{R}^{Nd\times d} \qquad \boldsymbol{G}_3 \triangleq \mathrm{Vec}\big(\boldsymbol{V}(T)\big)^\top \boldsymbol{F}_3 \in \mathbb{R}^d$$

  and for $y \in \{0,\dots,d-1\}$:

$$\boldsymbol{G}_3[y] = \sum_{k=0}^{N-1}\sum_{x=0}^{d-1} \boldsymbol{V}_T[k,x]\boldsymbol{F}_3[dk+x,y]$$

$$= \sum_{i,j} \sum_{k=0}^{N-1} \Bigg[\underbrace{\sum_{x=0}^{d-1} \boldsymbol{V}_T[k,x]\boldsymbol{\phi}_i[x]}_{=[\boldsymbol{V}(T)\boldsymbol{\phi}_i]_k} \Bigg] \boldsymbol{z}_T[i,j,m,k]\boldsymbol{\phi}_j[y]$$

$$= \sum_{i,j} \Bigg[\underbrace{\sum_{k=0}^{N-1} [\boldsymbol{V}(T)\boldsymbol{\phi}_i]_k \boldsymbol{z}_T[i,j,m,k]}_{=\boldsymbol{z}_T[i,j,m]^\top \boldsymbol{V}(T)\boldsymbol{\phi}_i \in \mathbb{R}} \Bigg] \boldsymbol{\phi}_j[y]$$

$$= \sum_{j=1}^{M} \Bigg[\underbrace{\sum_{i=1}^{M} \boldsymbol{z}_T[i,j,m]^\top \boldsymbol{V}(T)\boldsymbol{\phi}_i}_{=\boldsymbol{D}_{m,j}} \Bigg] \boldsymbol{\phi}_j[y] = \big[\boldsymbol{D}_m\boldsymbol{\phi}\big]_y$$

$\boldsymbol{G}_3$ the projection of $\boldsymbol{V}(T)$ onto the third term of the Jacobian is equal to $\boldsymbol{D}_m\boldsymbol{\phi}$.

□

Note that the projection of $\boldsymbol{V}(T)$ as defined in Proposition 6 is not equal to the meta-gradient itself, as it is missing the term coming from the partial derivative of the outer-loss wrt. the embedding vector $\partial\mathcal{L}\big(\boldsymbol{W}(T); f_\Phi(\mathcal{D}_\tau^{\text{test}})\big)/\partial\boldsymbol{\phi}_m$, which is non-zero unlike the counterpart for $\boldsymbol{W}_0$. However, this projection is the only expensive operation required to compute the total gradient.

D Proof of stability

In this section, we would like to show that unlike the adjoint method (see Sections 2.2 and 3.2), our method to compute the meta-gradients of COMLN is guaranteed to be *stable*. In other words, even under some small perturbations due to the ODE solver, the solution found by numerical integration is going to stay close to the true solution of the dynamical system. To do so, we use the concept of Lyapunov stability of the solution of an ODE, which we recall here for completeness:

Definition 1 (Lyapunov stability (Wiggins, 2003))**.** *The solution $\bar{\boldsymbol{x}}(t)$ of a dynamical system is said to be* Lyapunov stable *if, given $\varepsilon > 0$, there exists δ such that for any other solution $\boldsymbol{x}(t)$ such that $\|\bar{\boldsymbol{x}}(0) - \boldsymbol{x}(0)\| < \delta$, then $\|\bar{\boldsymbol{x}}(t) - \boldsymbol{x}(t)\| < \varepsilon$ for all $t > 0$.*

We would like to understand the conditions needed for a function $f : \mathbb{R}^n \to \mathbb{R}^n$, such that a trajectory $\bar{\boldsymbol{x}}(t)$ satisfying the following autonomous differential equation is (Lyapunov) stable:

$$\frac{d\boldsymbol{x}}{dt} = f\big(\boldsymbol{x}(t)\big) \tag{14}$$

If we assume for a moment that $f(\boldsymbol{x}) = -\nabla\mathcal{L}(\boldsymbol{x})$ where $\mathcal{L}$ is a convex function, then for any two trajectories $\boldsymbol{x}_1(t)$ and $\boldsymbol{x}_2(t)$ of the above dynamical system, we can see that the function $F : \mathbb{R} \to \mathbb{R}$ defined by

$$F(t) = \frac{1}{2}\|\boldsymbol{x}_1(t) - \boldsymbol{x}_2(t)\|^2 \tag{15}$$

satisfies $\dot{F}(t) \leq 0$. In fact, since $\nabla^2\mathcal{L}$ is positive semi-definite and by defining $\boldsymbol{h}(t) = \boldsymbol{x}_2(t) - \boldsymbol{x}_1(t)$, we get

$$\dot{F}(t) = \frac{dF}{dt} = \left[\frac{dF}{d\boldsymbol{h}}\right]^\top \frac{d\boldsymbol{h}}{dt} \tag{16}$$

$$= \big(\boldsymbol{x}_2(t) - \boldsymbol{x}_1(t)\big)^\top \big(\dot{\boldsymbol{x}}_2(t) - \dot{\boldsymbol{x}}_1(t)\big) \tag{17}$$

$$= \big(\boldsymbol{x}_2(t) - \boldsymbol{x}_1(t)\big)^\top \big(f(\boldsymbol{x}_2(t)) - f(\boldsymbol{x}_1(t))\big) \tag{18}$$

$$= \boldsymbol{h}(t)^\top \left[\int_0^1 Df\big(\boldsymbol{x}_1(t) + s\boldsymbol{h}(t)\big)ds\right] \boldsymbol{h}(t) \tag{19}$$

$$= -\int_0^1 \underbrace{\boldsymbol{h}(t)^\top \nabla^2\mathcal{L}\big(\boldsymbol{x}_1(t) + s\boldsymbol{h}(t)\big)\boldsymbol{h}(t)}_{\geq 0} ds \tag{20}$$

where Equation 19 follows from the *fundamental theorem of calculus*. Now since $\dot{F}(t) \leq 0$, the distance between two trajectories decreases as time t increases, hence we have Lyapunov stability for all trajectories or solutions of the above autonomous differential equation.

Now note that we didn't actually need f to be of the specific form $f(\boldsymbol{x}) = -\nabla\mathcal{L}(\boldsymbol{x})$, but having the Jacobian matrix Df negative semi-definite everywhere would have been sufficient. Hence we get the following statement

Proposition 7. *If a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}^n$ is such that the Jacobian matrix Df is negative semi-definite everywhere, then any solution $\bar{\boldsymbol{x}}(t)$ to the autonomous differential equation*

$$\frac{d\boldsymbol{x}}{dt} = f\big(\boldsymbol{x}(t)\big) \tag{21}$$

with initial condition $\bar{\boldsymbol{x}}(0) = \boldsymbol{x}_0$ is (Lyapunov) stable.

We can extend the above result to the non-autonomous case. For this purpose, let us define

$$\frac{d\boldsymbol{x}}{dt} = f\big(t, \boldsymbol{x}(t)\big) \tag{22}$$

which induces the parametrized function $f_t : \boldsymbol{x} \mapsto f(t, \boldsymbol{x})$. With this notation we can state the following result.

Proposition 8. *If the Jacobian matrix Df_t is negative semi-definite everywhere for all $t \geq 0$, then any trajectory $\boldsymbol{x}(t)$, which is a solution to Equation 22 with initial condition $\boldsymbol{x}(0) = \boldsymbol{x}_0$ is (Lyapunov) stable.*

Proof. Let us start with two trajectories $\boldsymbol{x}_1(t)$ and $\boldsymbol{x}_2(t)$, which are solutions to the above Equation 22. By defining $\boldsymbol{h}(t) = \boldsymbol{x}_2(t) - \boldsymbol{x}_1(t)$ and by applying the *fundamental theorem of calculus*, we get

$$f\big(t, \boldsymbol{x}_1(t)\big) - f\big(t, \boldsymbol{x}_2(t)\big) = f_t\big(\boldsymbol{x}_1(t)\big) - f_t\big(\boldsymbol{x}_2(t)\big) \tag{23}$$

$$= \left[\int_0^1 Df_t\big(\boldsymbol{x}_1(t) + s\boldsymbol{h}(t)\big)ds\right] \cdot \boldsymbol{h}(t) \tag{24}$$

Following the same idea as already previously outlined, let us define $F(t) = \frac{1}{2}\|\boldsymbol{x}_1(t) - \boldsymbol{x}_2(t)\|^2$ and show that $\dot{F}(t) \leq 0$:

$$\dot{F}(t) = (\boldsymbol{x}_2(t) - \boldsymbol{x}_1(t))^\top \big(f(t, \boldsymbol{x}_2(t)) - f(t, \boldsymbol{x}_1(t))\big) \tag{25}$$

$$= \boldsymbol{h}(t)^\top \left[\int_0^1 Df_t\big(\boldsymbol{x}_1(t) + s\boldsymbol{h}(t)\big)ds\right] \boldsymbol{h}(t) \tag{26}$$

$$= \int_0^1 \underbrace{\boldsymbol{h}(t)^T Df_t\big(\boldsymbol{x}_1(t) + s\boldsymbol{h}(t)\big)\boldsymbol{h}(t)}_{\leq 0} ds \leq 0 \tag{27}$$

Hence the distance between two trajectories decreases as time t increases, and we have Lyapunov stability for all solutions of the non-autonomous differential equation above. □

D.1 Stability of the forward sensitivity equations

In this subsection let us start by considering the general case of solving the initial value problem to the following autonomous system of differential equations:

$$\begin{cases} \dfrac{d\boldsymbol{W}}{dt} = g(\boldsymbol{W}(t), \boldsymbol{\theta}) & \boldsymbol{W}(0) = \boldsymbol{W}_0 \\ \dfrac{d\mathcal{S}}{dt} = \dfrac{\partial g\big(\boldsymbol{W}(t), \boldsymbol{\theta}\big)}{\partial \boldsymbol{W}(t)} \cdot \mathcal{S}(t) + \dfrac{\partial g\big(\boldsymbol{W}(t), \boldsymbol{\theta}\big)}{\partial \boldsymbol{\theta}} & \mathcal{S}(0) = \dfrac{\partial \boldsymbol{W}(0)}{\partial \boldsymbol{\theta}}. \end{cases} \tag{28}$$

Proposition 9. *There exists a solution of the autonomous system of differential equations in Equation 28, which is also unique.*

Proof. By Theorem 7.1.1 in Wiggins (2003), we know that there exists a solution to the system of differential equations in Equation 28. Then, by Theorem 7.4.1 in Wiggins (2003), we can conclude that this solution is unique upon the choice of initial conditions $\boldsymbol{W}(0) = \boldsymbol{W}_0$ and $\mathcal{S}(0) = \frac{\partial \boldsymbol{W}(0)}{\partial \boldsymbol{\theta}}$.

Alternatively, one could also apply Theorem 7.1.1 and Theorem 7.4.1 in Wiggins (2003), to conclude that the initial value problem $\frac{d\boldsymbol{W}}{dt} = g(\boldsymbol{W}(t), \boldsymbol{\theta})$ with initial condition $\boldsymbol{W}(0) = \boldsymbol{W}_0$ has a unique solution. Then the existence and uniqueness of this solution then gives rise to existence and uniqueness of a solution to the entire system of differential equations via $\mathcal{S}(t) = \frac{d\boldsymbol{W}}{dt}$ by applying Clairnaut's theorem. □

In our case, we have that $g(\boldsymbol{W}(t), \theta) = -\nabla\mathcal{L}\big(\boldsymbol{W}(T), \boldsymbol{\theta}\big)$, where $\mathcal{L}$ is a convex function in $\boldsymbol{W}$. Hence the autonomous system of differential equations rewrites as

$$\begin{cases} \dfrac{d\boldsymbol{W}}{dt} = -\nabla\mathcal{L}\big(\boldsymbol{W}(T), \boldsymbol{\theta}\big) & \boldsymbol{W}(0) = \boldsymbol{W}_0 \\ \dfrac{d\mathcal{S}}{dt} = -\nabla^2\mathcal{L}\big(\boldsymbol{W}(T), \boldsymbol{\theta}\big) \cdot \mathcal{S}(t) + \dfrac{\partial g\big(\boldsymbol{W}(t), \boldsymbol{\theta}\big)}{\partial \boldsymbol{\theta}} & \mathcal{S}(0) = \dfrac{\partial \boldsymbol{W}(0)}{\partial \boldsymbol{\theta}}. \end{cases} \tag{29}$$

Proposition 10. *Let $\boldsymbol{W}(t)$ be the solution to*

$$\frac{d\boldsymbol{W}}{dt} = -\nabla\mathcal{L}\big(\boldsymbol{W}(T), \boldsymbol{\theta}\big) \qquad \boldsymbol{W}(0) = \boldsymbol{W}_0 \tag{30}$$

Then the trajectory $\boldsymbol{W}(t)$ is (Lyapunov) stable.

Proof. Note that $Dg\big(\boldsymbol{W}(t)\big) = -\nabla^2\mathcal{L}$ here. Since $\mathcal{L}$ is convex, $\nabla^2\mathcal{L}$ is positive semi-definite, and hence $Dg(\boldsymbol{W}(t))$ is negative semi-definite. Hence the result directly follows from Proposition 7. □

In addition we can now also conclude that the solution of Equation 29 is also (Lyapunov) stable.

Corollary 1. *The solution $\big[\boldsymbol{W}(t), \mathcal{S}(t)\big]$ of Equation 29 is (Lyapunov) stable.*

This result is general to the application of the forward sensitivity equations on a gradient vector field derived from a convex loss function. We can also show that when the gradient $d\boldsymbol{W}^\star/d\boldsymbol{\theta}$ exists and is finite, then the solution of the system is also bounded, which guarantees us that our solution will not diverge (unlike the adjoint method applied to a gradient vector field).

Proposition 11. *Assuming that $\big\|\frac{d\boldsymbol{W}^\star}{d\boldsymbol{\theta}}\big\|$ is finite, the solution $\big[\boldsymbol{W}(t), \mathcal{S}(t)\big]$ of Eq. 29 is bounded.*

Proof. Let us define

$$V(t) = \frac{1}{2}\|\boldsymbol{W}(t)\|^2 + \frac{1}{2}\|\mathcal{S}(t)\|^2$$

Since $\boldsymbol{W}(t)$ and $\mathcal{S}(t)$ are continuous functions in t, verifying

$$\boldsymbol{W}(t) \underset{t\to\infty}{\longrightarrow} \boldsymbol{W}^\star \qquad \text{and} \qquad \mathcal{S}(t) \underset{t\to\infty}{\longrightarrow} \frac{d\boldsymbol{W}^\star}{d\boldsymbol{\theta}}$$

Hence

$$V(t) \underset{t\to\infty}{\longrightarrow} \frac{1}{2}\|\boldsymbol{W}^\star\|^2 + \frac{1}{2}\Big\|\frac{d\boldsymbol{W}^\star}{d\boldsymbol{\theta}}\Big\|^2 < +\infty$$

where the last point follows from our assumption that $\big\|\frac{d\boldsymbol{W}(t)}{d\boldsymbol{\theta}}\big\| < +\infty$. We also have that

$$V(0) = \frac{1}{2}\|\boldsymbol{W}_0\|^2 + \frac{1}{2}\Big\|\frac{d\boldsymbol{W}(0)}{d\boldsymbol{\theta}}\Big\|^2 < +\infty$$

$V(t)$ being continuous as a composition of continuous functions, we conclude that V is bounded, and hence $\boldsymbol{W}(t)$ and $\mathcal{S}(t)$ are bounded as well. □

Table 4: The effect of the numerical solver on the performance of COMLN. The average accuracy (%) on $1,000$ held-out meta-test tasks is reported with 95% confidence interval. Note that for a given setting, the same $1,000$ tasks are used for evaluation, making both methods directly comparable. RK: 4th-order Runge-Kutta with Dormand Prince adaptive step size. Euler: explicit Euler scheme.

Method	***mini*ImageNet 5-way**	
	1-shot	**1-shot**
COMLN (RK)	53.01 ± 0.62	70.54 ± 0.54
COMLN (Euler)	53.00 ± 0.83	70.50 ± 0.72

E EXPERIMENTAL DETAILS

E.1 CHOICE OF THE NUMERICAL SOLVER

In Section 5.2, we showed that the numerical solver had a significant impact on the time to compute the meta-gradients (but not the memory requirements). In Table 4, we show that using explicit Euler instead of an adaptive scheme like Runge-Kutta leads to very similar results. In all our experiments we therefore chose Runge-Kutta for its faster execution, and by convenience—it is the default numerical solver available in JAX (Bradbury et al., 2018).

E.2 DETAILS FOR MEASURING THE EFFICIENCY

In Figures 3 & 4, we show the efficiency, both in terms of memory and runtime, of COMLN compared to other meta-learning algorithms. The computational efficiency (runtime) was measured as the average time taken to compute the meta-gradients on a single 5-shot 5-way task from the *mini*ImageNet dataset over 100 runs.

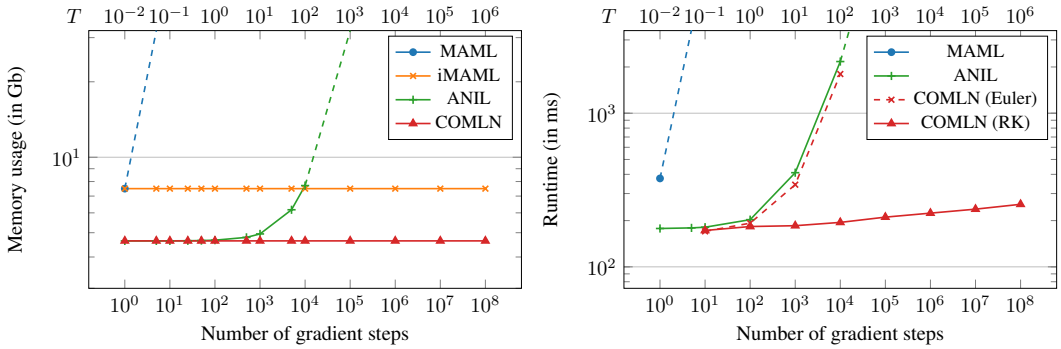


Figure 4: Empirical efficiency of COMLN on a single 5-shot 5-way task, with a ResNet-12 backbone; this figure is similar to Figure 3. (Left) Memory usage for computing the meta-gradients as a function of the number of inner-gradient steps. The extrapolated dashed lines correspond to the method reaching the memory capacity of a Tesla V100 GPU with 32Gb of memory. (Right) Average time taken (in ms) to compute the exact meta-gradients. The extrapolated dashed lines correspond to the method taking over 3 seconds.

In order to ensure fair comparison between methods that rely on a discrete number of gradient steps (MAML, ANIL, and iMAML), and COMLN which relies on a continuous integration time T, we added a conversion between the number of gradient steps and T. This corresponds to taking a learning rate of $\alpha = 0.01$ in Equation 2 (which is standard for MAML and ANIL on miniImageNet) (i.e. the number of gradient steps is $100\times$ larger than T). This can be formally justified by considering an explicit Euler scheme for COMLN (see Section 3.3), with a constant step size $\alpha = 0.01$. The memory requirements is independent of the choice of the numerical solver. For the computation time, this correspondence between T and the number of gradient steps is no longer exact for *COMLN (RK)*—the comparison with *COMLN (Euler)* is still valid though.

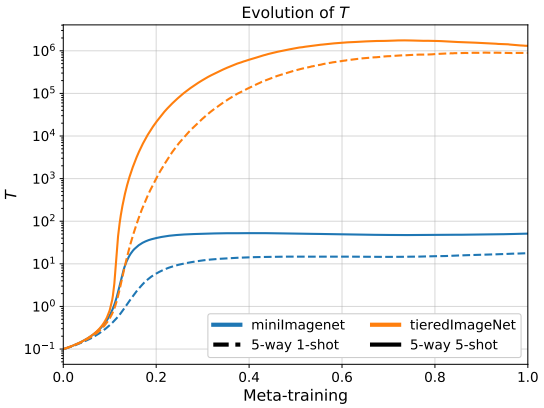


Figure 5: Evolution of the meta-parameter T controlling the amount of adaptation necessary for all tasks during meta-training. Here, the backbone is a ResNet-12. We normalized the duration of meta-training in $[0, 1]$ to account for early stopping; typically the model for *mini*ImageNet requires an order of magnitude fewer iterations.

E.3 Analysis of the learned horizon

Since one of the advantage of COMLN compared to other gradient-based methods is its capacity to learn the amount of adaptation through the time horizon T, Figure 5 shows the evolution of this meta-parameter during meta-training. We can observe that for the more complex dataset *tiered*ImageNet, COMLN appropriately learns to use a longer sequence of adaptation. Similarly within each dataset, it also learns to use shorter sequences of adaptation of 1-shot problems, possibly to allow for better generalization and to reduce overfitting.

E.4 Additional Experiment: Preprocessed *mini*ImageNet dataset

In addition to our experiments on the *mini*ImageNet and *tiered*ImageNet datasets in Section 5, we want to evaluate the performance of continuous-time adaptation in isolation from learning the embedding network. We evaluate this using a preprocessed version of *mini*ImageNet introduced in (Rusu et al., 2018), where the embeddings were trained using a Wide Residual Network (WRN) via supervised classification on the meta-train set. For our purposes, we consider these embeddings $\boldsymbol{\phi} \in \mathbb{R}^{640}$ as fixed, and only meta-learn the initial conditions $\boldsymbol{W}_0$ as well as T.

The classification accuracies for COMLN and other baselines using pretrained embeddings are shown in Table 5. COMLN achieves comparable or better performance with a single linear classifier layer as other meta-learning methods in both the 5-way 1-shot and 5-way 5-shot tasks. The single exception is the 5-way 1-shot result of Meta-SGD from (Rusu et al., 2018), which exceeded the performance of COMLN. However, our implementation of Meta-SGD achieved comparable performance to COMLN. This gap in performance is likely due to the the additional data used in (Rusu et al., 2018) (meta-train and meta-validation splits) during meta-training, as opposed to the only meta-training set in all other baselines. These results show that isolated from representation learning, all these meta-learning algorithms (either gradient-based or not) perform similarly, and COMLN is no exception.

One notable exception though is MetaOptNet (Lee et al., 2019), where the performance is not as high as the other baselines when the backbone network is not learned anymore—despite often being the best performing model in Table 2. Our hypothesis is that this discrepancy is due to the accuracy of the QP solver used in MetaOptNet, since learning individual SVMs on 1,000 held-out meta-test tasks leads to performance matching all other methods (about 50% for 5-way 1-shot and about 70% for 5-way 5-shot).

E.5 Experimental details

For all methods and all datasets, we used SGD with momentum 0.9 and Nesterov acceleration, with a decreasing learning rate starting at 0.1 and decreasing according to the schedule provided by Lee et al. (2019). For meta-training, we followed the standard procedure in gradient-based meta-learning, and meta-trained with a fixed number of shots: for example in *mini*ImageNet 5-shot 5-way, we only used tasks with $k = 5$ training examples for each of the $N = 5$ classes. This contrasts with (Lee et al., 2019), which uses a larger number of shots during meta-training than the one used for evaluation (e.g. meta-training with $k = 15$, and evaluating on $k = 1$). This may explain the gap

Table 5: *mini*ImageNet results using LEO embeddings and a single linear classifier layer. The average accuracy (%) on 1,000 held-out meta-test tasks is reported with 95% confidence interval. * Results reported in (Rusu et al., 2018). ** Note that LEO uses more than a single linear classifier layer, but we add the numbers for completeness.

Model	***mini*ImageNet 5-way**	
	1-shot	**5-shot**
MAML (Finn et al., 2017)	50.35 ± 0.63	65.28 ± 0.54
Meta-SGD* (Li et al., 2017)	54.24 ± 0.03	70.86 ± 0.04
Meta-SGD (Li et al., 2017)	50.57 ± 0.64	69.09 ± 0.53
iMAML (Rajeswaran et al., 2019)	50.26 ± 0.61	69.52 ± 0.51
R2D2 (Bertinetto et al., 2018)	50.33 ± 0.62	70.38 ± 0.52
LRD2 (Bertinetto et al., 2018)	50.41 ± 0.62	70.29 ± 0.52
LEAP (Flennerhag et al., 2018)	50.95 ± 0.62	66.72 ± 0.55
MetaOptNet (Lee et al., 2019)	40.60 ± 0.60	50.94 ± 0.62
LEO** (Rusu et al., 2018)	61.76 ± 0.08	77.59 ± 0.12
COMLN (Ours)	50.39 ± 0.63	70.06 ± 0.52

in performance between COMLN and MetaOptNet, especially on 1-shot settings. We opted to not follow this decision made by Lee et al. (2019) to ensure a fair comparison with other gradient-based methods, which all used the process described above.

Conv-4 backbone We used a standard convolutional neural network with 4 convolutional blocks (Finn et al., 2017). Each block consists of a convolutional layer with a 3×3 kernel and 64 channels, followed by a batch normalization layer, and a max-pooling layer with window size and stride 2×2. The activation function is a ReLU.

ResNet-12 backbone We largely followed the architecture from (Lee et al., 2019), which consists of a 12-layer residual network. The neural network is composed of 4 blocks with residual connections of 3 convolutional layers with a 3×3 kernel. The convolutional layers in the residual block have $k = 64$, 160, 320 and 640 channels respectively. The non-linearity functions are $\text{LEAKYRELU}(0.1)$, and a max-pooling layer with window size and stride 2×2 is applied at the end of each block. No global pooling is performed at the end of the embedding network, meaning that the embedding dimension is $d = 16,000$. The only notable difference with the architecture used by Lee et al. (2019) is the absence of DropBlock (Ghiasi et al., 2018) regularization.