META CONTINUAL LEARNING VIA DYNAMIC PROGRAMMING

Anonymous authors
Paper under double-blind review

ABSTRACT

Meta continual learning algorithms seek to train a model when faced with similar tasks observed in a sequential manner. Despite promising methodological advancements, there is a lack of theoretical frameworks that enable analysis of learning challenges such as generalization and catastrophic forgetting. To that end, we develop a new theoretical approach for meta continual learning (MCL) where we mathematically model the learning dynamics using dynamic programming, and we establish conditions of optimality for the MCL problem. Moreover, using the theoretical framework, we derive a new dynamic-programming-based MCL method that adopts stochastic-gradient-driven alternating optimization to balance generalization and catastrophic forgetting. We show that, on MCL benchmark data sets, our theoretically grounded method achieves accuracy better than or comparable to that of existing state-of-the-art methods.

1 INTRODUCTION

The central theme of meta continual learning (MCL) is to learn on similar tasks revealed sequentially. In this process, two fundamental challenges must be addressed: catastrophic forgetting of the previous tasks and generalization to new tasks (Javed and White, 2019). In order to address these challenges, several approaches (Javed and White, 2019; Beaulieu et al., 2020; Riemer et al., 2018) have been proposed in the literature that build on the second-order, derivative-driven approach introduced in Finn et al. (2017).

Despite the promising prior methodological advancements, existing MCL methods suffer from three key issues: (1) there is a lack of a theoretical framework to systematically design and analyze MCL methods; (2) data samples representing the complete task distribution must be known in advance (Finn et al., 2017; Javed and White, 2019; Beaulieu et al., 2020), often, an impractical requirement in real-world environments as the tasks are observed sequentially; and (3) the use of fixed representations (Javed and White, 2019; Beaulieu et al., 2020) limits the ability to handle significant changes in the input data distribution, as demonstrated in Caccia et al. (2020). We focus on a supervised learning paradigm within MCL, and our key contributions are (1) a dynamic-programming-based theoretical framework for MCL and (2) a theoretically grounded MCL approach with convergence properties that compare favorably with the existing MCL methods.

Dynamic-programming-based theoretical framework for MCL: In our approach, the problem is first posed as the minimization of a cost function that is integrated over the lifetime of the model. Nevertheless, at any time t, the future tasks are not available, and the integral calculation becomes intractable. Therefore, we use the Bellman's principle of optimality (Bellman, 2015) to recast the MCL problem to minimize the sum of catastrophic forgetting cost on the previous tasks and generalization cost on the new task. Next, we theoretically analyze the impact of these costs on the MCL problem using tools from the optimal control literature (Lewis et al., 2012). Furthermore, we demonstrate that the MCL approaches proposed in (Finn et al., 2017; Beaulieu et al., 2020; Javed and White, 2019) can be derived from the proposed framework. Our theoretical framework unifies different MCL approaches. In this paper, we discuss the connection between the theoretical framework and other MCL methods in the literature. Specifically, we show how these methods can be derived from our framework.

Theoretically grounded MCL approach: We derive a theoretically grounded dynamic programming-based meta continual learning (DPMCL) approach. The generalization cost is com-

puted by training and evaluating the model on given new task data. The catastrophic forgetting cost is computed by evaluating the model on the task memory (previous tasks) after the model is trained on the new task. We alternately minimize the generalization and catastrophic forgetting costs for a predefined number of iterations to achieve a balance between the two. (See Fig. 2 in the appendix for an overview). We analyze the performance of the DPMCL approach experimentally on classification and regression benchmark data sets.

2 Problem Formulation

We focus on the supervised MCL setting. We let $\mathbb{R}$ denote the set of real numbers and use boldface to denote vectors and matrices. We use $\|.\|$ to denote the Euclidean norm for vectors and the Frobenius norm for matrices. The lifetime of the model is given by $[0, \Gamma] : \Gamma \in \mathbb{R}$, where Γ is the maximum lifetime of the model. We let $p(\boldsymbol{\mathcal{T}})$ be the distribution over all the tasks in the interval $[0, \Gamma]$. Based on underlying processes that generate the tasks, the task arrival can be continuous time (CT) or discrete time (DT). For example, consider the system identification problem in a stochastic process modeled by ordinary differential equations (ODEs) or partial differential equations (PDE)–where the tasks, represented by the states of the process $x(t)$, are generated in CT through the ODE. On the other hand, in the typical supervised learning setting, consider an image classification problem where each task comprises a set of images sampled from a discrete process and the tasks arrive in DT. As in many previous MCL works, we focus on DT MCL. However, we develop our theory for the CT MCL setting first because a CT MCL approach is broadly applicable to many domains. In Section 3.3 we provide a DPMCL approach for DT MCL setting by discretizing our theory. A task $\boldsymbol{\mathcal{T}}(t)$ is a tuple of input-output pairs $\{\boldsymbol{\mathcal{X}}(t), \boldsymbol{\mathcal{Y}}(t)\}$ provided in the interval $[t, t + \Delta t] \forall t \in [0, \Gamma), \Delta t \in \mathbb{R}$. We denote $(\boldsymbol{x}(t), \boldsymbol{y}(t)) \in \{\boldsymbol{\mathcal{X}}(t), \boldsymbol{\mathcal{Y}}(t)\}$. We define a parametric model $g(.)$ with parameters $\hat{\boldsymbol{\theta}}$ such that $\hat{\boldsymbol{y}}(t) = g(\boldsymbol{x}(t); \hat{\boldsymbol{\theta}}(t))$. Although we will use neural networks, any parametric model can be utilized with our framework. Our MCL problem is comprised of two challenges, catastrophic forgetting on previous tasks and generalization to new tasks.

The catastrophic forgetting cost measures the error of the model on all the previous tasks; the generalization cost measures the error of the model on the new task. The goal in MCL is to minimize both the catastrophic forgetting cost and generalization cost for every $t \in [0, \Gamma)$. Let us split the interval $[0, \Gamma)$ as $[0, t] \cup (t, \Gamma)$, where the intervals $[0, t]$ and (t, Γ) comprise previous tasks and new task respectively (i.e., the collection of all the task that can be observed in the respective intervals). To take all the previous tasks into account, we define the instantaneous catastrophic forgetting cost $J(t; \hat{\boldsymbol{\theta}}(t))$ to be the integral of the loss function $\ell(\tau)$ for any $t \in [0, \Gamma)$ as

$$J(t; \hat{\boldsymbol{\theta}}(t)) = \int_{\tau=0}^{t} \gamma(\tau)\ell(\tau)d\tau, \tag{1}$$

where $\ell(\tau)$ is computed on task $\boldsymbol{\mathcal{T}}(\tau)$ with $\gamma(\tau)$ being a parameter describing the contribution of this task to the integral. The value of $\gamma(\tau)$ is critical for the integral to be bounded (details are provided in Lemma 1). Given a new task, the goal is to perform well on the new task as well as maintain the performance on the previous tasks. To this end, we write $V(t; \hat{\boldsymbol{\theta}}(t))$, as the cumulative cost (combination of catastrophic cost and generalization cost) that is integrated over $[t, \Gamma)$. We therefore seek to minimize $V(t; \hat{\boldsymbol{\theta}}(t))$ and obtain the optimal value, $V^*(t)$, by solving the problem

$$V^*(t) = min_{\hat{\boldsymbol{\theta}}(\tau) \in \Omega : t \leq \tau \leq \Gamma} \int_{\tau=t}^{\Gamma} J(\tau; \hat{\boldsymbol{\theta}}(\tau))d\,\tau, \tag{2}$$

where Ω is the compact set required to initialize the parameters of the model. In a non-convex optimization problem, the compact set describes the boundaries of the solution space. It is also assumed that there is at least one minima of the optimization space in this set. If the parameters are initialized from within the compact set, the optimization may converge to the local solution within the set.

MCL is a sequential decision making problem where the goal is to obtain a sequence of parameters in the interval (t, Γ) as described in Eq. (2). In MCL context, whenever a new tasks is observed at $\tau \in (t, \Gamma)$, we seek to make a decision (find a parameter set, $\hat{\boldsymbol{\theta}}(\tau) \in \Omega$) such that $\hat{\boldsymbol{\theta}}(\tau)$ is optimal for all the previous tasks and the new task in the interval $[0, \tau]$. Consequentially, for each new task,

we obtain a new parameter set and the solution to the MCL problem is provided by a sequence of parameters (decisions). For making each of these decision, a task is counted exactly one and the contribution of cost is determined by the choice of γ.

The optimization problem in Eq. (2) in its current form is intractable for two reasons. First, note that $V^*(t)$ is the optimal cost value over the complete lifetime of the model $[0, \Gamma)$. Since we have only the data corresponding to all the tasks in the interval $[0, t]$, solving Eq. (2) in its current form is intractable. Second, it is not feasible to maintain a parameter set for each $\tau \in (t, \Gamma)$. To circumvent this issue, we take a dynamic programming view of the MCL problem. We introduce a new theoretical framework where we model the learning process as a dynamical system and use Bellman's principle of optimality to simplify the MCL problem. Furthermore, we derive conditions under which the learning process is stable and optimal, using tools from the optimal control literature (Lewis et al., 2012).

3 Theoretical Meta Continual Learning Framework

We will recast the problem defined in Eq. (2) using ideas from dynamic programming, specifically Bellman's principle of optimality (Lewis et al., 2012). We treat the MCL problem as a dynamical system and describe the system using the following PDE:

$$-\frac{\partial V^*(t)}{\partial t} = min_{\hat{\boldsymbol{\theta}}(t)\in\Omega}\big[J(t;\hat{\boldsymbol{\theta}}(t)) + J_N(t;\hat{\boldsymbol{\theta}}(t)) + \big(V^*_{\hat{\boldsymbol{\theta}}(t)}\big)^T\Delta\hat{\boldsymbol{\theta}}\big] + \big(V^*_{\boldsymbol{x}(t)}\big)^T\Delta\boldsymbol{x}(t), \tag{3}$$

where $V^*(t)$ describes the optimal cost (the left-hand side of Eq. (2)) and $(.)^T$ refers to the transpose operator. The notation $A_{(.)}$ denotes the partial derivative of A with respect to $(.)$, for instance, $V^*_{\hat{\boldsymbol{\theta}}(t)} = \frac{\partial V^*(t;\hat{\boldsymbol{\theta}}(t))}{\partial\hat{\boldsymbol{\theta}}(t)}$ *The full derivation for Eq.* (3) *from Eq.* (2) *is provided in Appendix A.1.*. Note that since $\boldsymbol{y}(t)$ is a function of $\boldsymbol{x}(t)$, the changes in the optimal cost due to $\boldsymbol{y}(t)$ are captured by $\big(V^*_{\boldsymbol{x}(t)}\big)^T\Delta\boldsymbol{x}(t)$. Eq. (3) is also known as the Hamilton-Jacobi Bellman equation in optimal control (Lewis et al., 2012) with the key difference that there is an extra term to quantify the changes due to the new task.

Intuitively, the PDE completely describes the dynamics of learning for the MCL problem in the period $[0, \Gamma]$. Specifically, the left hand side of Eq. (3), $\frac{\partial V^*(t)}{\partial t}$ describes the change in the global solution of the MCL problem, $V^*(t)$, with respect to time t. The right hand side describes what are the different components of the MCL problem that impact this global solution. Note from the right hand side of Eq. (3), this impact is quantified by the four terms: the cost contribution from all the previous tasks $J(t;\hat{\boldsymbol{\theta}}(t))$; the cost due to the new task $J_N(t;\hat{\boldsymbol{\theta}}(t))$; the change in the optimal cost due to the change in the parameters $\big(V^*_{\hat{\boldsymbol{\theta}}(t)}\big)^T\Delta\hat{\boldsymbol{\theta}}$; and the change in the optimal cost due to change in the input (introduction of new task) $\big(V^*_{\boldsymbol{x}(t)}\big)^T\Delta\boldsymbol{x}(t)$. Since, the PDE completely describes our problem, the solution to the MCL problem can be obtained by obtaining the parameter $\hat{\boldsymbol{\theta}}(t)$ that minimizes the right-hand side of Eq. (3). Specifically, we seek to solve the following optimization problem

$$min_{\hat{\boldsymbol{\theta}}(t)\in\Omega}\big[J(t;\hat{\boldsymbol{\theta}}(t)) + J_N(t;\hat{\boldsymbol{\theta}}(t)) + \big(V^*_{\hat{\boldsymbol{\theta}}(t)}\big)^T\Delta\hat{\boldsymbol{\theta}}\big], \quad \text{s. t.} \quad \frac{\partial V^*(t)}{\partial t} + \big(V^*_{\boldsymbol{x}(t)}\big)^T\Delta\boldsymbol{x}(t) = 0. \tag{4}$$

Solving the problem in Eq. (4) is equivalent to minimizing the impact of introducing a new task on the optimal cost. Observe that the intractable problem in Eq. (2) has been posed as a PDE constrained optimization Eq. (4). Note that in Eq. (2), the global solution is achieved when a series of parameters are obtained. On the other hand, in the optimization problem in Eq. (4) we obtain a parameter at time t to achieve the global solution under certain assumption on the parameters and data. Although, this solution is under assumptions on the data, these assumptions can be satisfied in practice. More details about the convergence of this optimization and assumptions involved is provided in Theorem. 1 in the next section.

3.1 ANALYSIS

Our formalism has two critical elements: $\gamma(t)$, which quantifies the contribution of each task to catastrophic forgetting cost, and the impact of the change in the input data distribution $\Delta \boldsymbol{x}$ on learning, specifically while adapting to new tasks. We will analyze them next.

Impact of $\gamma(t)$ on catastrophic forgetting cost: Since, the cost in Eq. (1) is an integral of cost contributions from all the tasks, it is critical that this integral has a converging point. In other words, we seek to understand, when all the tasks provide non-zero values to the overall loss, is the cost is bounded and can the optimization problem be solved? The existence of the convergence point depends directly on the contribution of each task determined through $\gamma(t)$. We therefore present **Lemma 1 and Corollary 1** *(the complete statement and proofs can be found in Appendix A.2)*

In Lemma 1, we demonstrate that when all tasks in a MCL problem provide a nonzero cost, it is not possible to maintain equivalent performance on all the tasks. Specifically, we show that when $\gamma(\tau) = 1, \forall \tau$ that is the contribution of each task to the cost is greater than zero and equal, then the integral diverges when the number of tasks tend to ∞. This phenomenon has been observed empirically (Lin, 1992). However, by choosing $\gamma(t)$ appropriately, we can control the catastrophic forgetting by selecting which tasks to forget. One reasonable solution is to give older tasks less priority and new tasks more priority, this is shown in Corollary 1 to provide a cost function that is both bounded and convergent. An example of such γ is $\gamma(\tau) = e^{-\tau}, \tau \in [0, \Gamma]$. However, any choice of $\gamma(t)$ that will keep $J(t; \hat{\boldsymbol{\theta}}(t))$ bounded is reasonable.

The second most important component of our approach is the impact of change in the input (Δx) on learning. To substantiate this, we present **Theorem 1** *(the complete statement and proof can be found in Appendix A.2).* Theorem 1 shows the *convergence of a gradient-based solution to the MCL learning* under assumptions on the input, the gradient and the learning rate. Specifically, we show through Lyapunov principles that $J(t; \hat{\boldsymbol{\theta}}(t))$ (the cost on all the previous tasks) decreases as $t \to \infty$ and ultimately achieves a value less than β. In Theorem 1, there are four main assumptions. First is a consequence of Corollary 1 where the contributions of each task to the cost must be chosen such that the cost is bounded and convergent. Second is the assumption of a compact set Ω. This assumption implies that if a weight value initialized from within the compact set Ω, there will be a convergence to local minima. Third, we consider the condition $||J_{\hat{\boldsymbol{\theta}}(t)}|| = 0$ which is well known in the literature as the vanishing gradient problem (Pascanu et al., 2013).

Impact of $\Delta \boldsymbol{x}(t)$ on learning: The fourth assumption, that is $\|J_x\| > 0$ is important to the proof and directly explains how $\Delta \boldsymbol{x}(t)$ can impact the validity of Theorem 1 (thus the convergence of the approach). Note that if $\|J_x\| = 0$, the value of the cost J will not change due to change in the input $\Delta \boldsymbol{x}(t) > 0$. Due to this, Theorem 1 will not hold and the learning will stagnate. To give an example, consider the MNIST dataset with a total of 10 classes and the solution of the MCL problem is to predict efficiently on all the 10 classes. Consider now the case when each task is created randomly to include exactly one class and each task is being shown to the model sequentially. If the model only experiences classes 1 through 5 and does not experience classes 5 through 10 (by virtue of improper sampling), the information provided to the model is not informative enough to perform well on all the 10 classes. Thus, although the model is perfect in predicting classes 1 through 5 such that $\|J_x\| = 0$, the MCL problem has not reached the global solution. Consequently, the learning process has stagnated. In control theory, this condition is known as persistence of excitation (Lewis et al., 1998).

On the other hand, a large change in the input data distribution presents issues in the learning process as well. Note that for Theorem 1 to hold, $\alpha(t) > 0$, therefore, $\|J_x\| \|\Delta x(t)\| < 1$. Let $\|\Delta \boldsymbol{x}(t)\| \leq b_x$, where b_x is the upper bound on the change in the input data distribution. If b_x is large (going from predicting on images to understanding texts), the condition $\|J_x\| \|\Delta x(t)\| < 1$ will be violated, and our approach will be unstable. We can, however, adapt our model to the change in the input data-distribution $\Delta x(t)$ exactly if we can explicitly track the change in the input. This type of adaptation can be done easily when the process generating $\boldsymbol{x}(t)$ can be described by using an ODE or PDE. In traditional supervised learning settings, however, such a description is not possible. The issue highlights the need for strong representation learning methodologies where a good representation over all the tasks can minimize the impact of changes in $\Delta x(t)$ on the performance of the MCL problem (Javed and White, 2019; Beaulieu et al., 2020). Currently, in the literature, it is common to control the magnitude of $\Delta x(t)$ through normalization procedures under the assumption that all

tasks are sampled from the same distribution. Therefore, for all practical purposes, we can choose $0 \leq \alpha(t) \leq (\beta \, ||J_{\hat{\boldsymbol{\theta}}(t)}||)^{-1}$.

Connection to MAML, FTML, and their variants: The optimization problem in MAML (Finn et al., 2017), FTML (Finn et al., 2019) and other variants can be obtained from Eq. (3) by setting the third and the fourth terms to zero, which provides $-\frac{\partial V^*(t)}{\partial t} = min_{\hat{\boldsymbol{\theta}}(t) \in \Omega}[J(t; \hat{\boldsymbol{\theta}}(t)) + J_N(t; \hat{\boldsymbol{\theta}}(t)]$. The MCL problem is split into three phases, meta training, meta testing and testing. In the meta training, we learn to generalize to new task. In meta testing, we learn from all the previous tasks, which aims at learning over $p(\boldsymbol{\mathcal{T}})$ (which is similar to minimize catastrophic forgetting). Finally, in the testing phase, the network predicts on a set of held out tasks that are representative of the complete task distribution. To learn common features across all the tasks, we must optimize for the first term on the right-hand side in the equation above (where a second order derivative can be utilized). When the goal is to generalize to new tasks, one must optimize the second term on the right hand side of the equation above. With the choice of different architectures for the neural network, all of the approaches that build on MAML and FTML such as (Javed and White, 2019) and (Beaulieu et al., 2020) can be directly derived.

For instance, to obtain the methodology in (Javed and White, 2019), we may do the following. First, the model architecture is described as a combination of representation learning network (RLN) and prediction learning network (PLN) such that $\hat{\boldsymbol{\theta}} = [\hat{\boldsymbol{\theta}}_1 \hat{\boldsymbol{\theta}}_2]$ and the vector space for $\hat{\boldsymbol{\theta}}_1$ and $\hat{\boldsymbol{\theta}}_2$ can be considered to denote PLN and RLN respectively. In the learning process, we will first pre-train a RLN as an encoder by optimizing the first term in Eq. (3.1) while a new PLN is randomly initialized for each new task. Next, we may freeze the RLN and update PLN to minimize the second term in Eq. (3.1) when new tasks are observed sequentially.

Similar to Javed and White (2019), $\hat{\boldsymbol{\theta}} = [\hat{\boldsymbol{\theta}}_1 \hat{\boldsymbol{\theta}}_2]$ in Beaulieu et al. (2020) described as a combination of neuro-modulatory network (NLM) and the prediction network (PLN). The methodology in Beaulieu et al. (2020) can be obtained by following a two step learning procedure. First, NLM and PLN are pre-trained to minimize the right hand side of (3.1) with data from all the available tasks. Second, we train the prediction network on unseen tasks by optimizing the second term in the right hand side of (3.1) and fix the NLM. All these methods do not adopt the PDE formalism; the third and fourth terms in Eq. (3) are not explicitly included in MAML and FTML. On the other hand, the works in (Javed and White, 2019) and (Beaulieu et al., 2020) learn to represent $p(\boldsymbol{\mathcal{T}})$, (the distribution over all the tasks) and require a pre-training phase.

To the best of our knowledge, the only work where the third and fourth term are implicitly addressed is meta experience replay (MER) (Riemer et al., 2018). MER models the interference (forward transfer and backward transfer) due to the introduction of new tasks as a gradient alignment problem. Observe, from Eq. (3), the third term models the change in the global solution (backward transfer) with respect to change in the weights. Furthermore, the fourth term models the change in the global solution (forward transfer) with respect to change in the input. MER can be directly derived from Eq. (3) by optimizing the third and fourth term with samples from the experience replay memory, to minimize interference (both forward and backward). This is very similar to DPMCL with the key difference that, instead of approximating the optimal cost directly, MER approximates the angle between the gradients.Furthermore, this approximation is performed using Reptile (Nichol et al., 2018). Reptile is essentially similar to constraining the change in the weight such that the third term in Eq. (3) is zero. Although this regularizes the learning in the presence of new tasks such that forgetting is not large, the network can unlearn experiences due to parameter drift especially when the new tasks are very similar to order tasks (Narendra and Annaswamy, 1987).

DPMCL is a new approach derived from the presented theoretical framework. In our DPMCL approach, we provide a clear and methodical procedure from Eq. (3) for deriving the weight update rule, providing us with a much more methodical process of addressing the impact of these terms and, by extension, the impact of key challenges in the MCL setting.

3.2 Dynamic programming-based meta continual learning (DPMCL)

As a consequence of Theorem 1, the update for the parameters is provided by $\alpha(t) V_{\hat{\boldsymbol{\theta}}(t))}$. Since $V(t; \hat{\boldsymbol{\theta}}(t))$ is not completely known, we have to approximate this gradient. To derive this approx-

imation, we first rewrite Eq. (3) as $-\frac{\partial V^*(t;\hat{\boldsymbol{\theta}}(t))}{\partial t} = min_{\hat{\boldsymbol{\theta}}(t)\in\Omega}\big[H(t;\hat{\boldsymbol{\theta}}(t))\big]$, which provides the optimization problem as $\hat{\boldsymbol{\theta}}^*(t) = \arg\min_{\hat{\boldsymbol{\theta}}(t)\in\Omega}\big[H(t;\hat{\boldsymbol{\theta}}(t)))\big]$, where $H(t;\hat{\boldsymbol{\theta}}(t))) = J(t;\hat{\boldsymbol{\theta}}(t)) + J_N(t;\hat{\boldsymbol{\theta}}(t)) + \big(V^*_{\hat{\boldsymbol{\theta}}(t)}\big)^T\Delta\hat{\boldsymbol{\theta}} + \big(V^*_{\boldsymbol{x}(t)}\big)^T\Delta\boldsymbol{x}(t)$ is the CT Hamiltonian. The solution for this optimization problem (the updates for the parameters in the network) is provided when the derivative of the Hamiltonian is set to zero.

First, we discretize the MCL problem setting. Let k be the discrete sampling instant such that $t = k\Delta t$, where Δt is the sampling interval. Let a task $\boldsymbol{\mathcal{T}}_k$ at instant k be sampled from $p(\boldsymbol{\mathcal{T}})$. Let $\boldsymbol{\mathcal{T}}_k = (\boldsymbol{\mathcal{X}}_k, \boldsymbol{\mathcal{Y}}_k)$ be a tuple, where $\boldsymbol{\mathcal{X}}_k \in \mathbb{R}^{n\times p}$ denotes the input data and $\boldsymbol{\mathcal{Y}}_k \in \mathbb{R}^{n\times 1}$ denotes the target labels (output). Let n be the number of samples and p be the number of dimensions. Let the parametric model be given as $\hat{\boldsymbol{y}} = g(h(\boldsymbol{x};\hat{\boldsymbol{\theta}}_1);\hat{\boldsymbol{\theta}}_2)$, where the inner map $h(.)$ is treated as a representation learning network and $g(.)$ is the prediction network. Let $\hat{\boldsymbol{\theta}} = [\hat{\boldsymbol{\theta}}_1 \quad \hat{\boldsymbol{\theta}}_2]$ be the learnable model parameters and the weight updates be given as

$$\hat{\boldsymbol{\theta}}(k+1) = \quad \hat{\boldsymbol{\theta}}(k) - \alpha(k)\frac{\partial}{\partial\hat{\boldsymbol{\theta}}(k)}\big[J_N(k) + J_P(k) + \big(J_{PN}(k) - J_{PN}(k;\hat{\boldsymbol{\theta}}(k+\zeta))\big)\big] \tag{5}$$

Our update rule has three terms (the terms inside the bracket). The first term depends on J_N, which is calculated on the new task; the second term depends on J_P and is calculated on all the previous tasks; the third term comprises J_{PN} and is evaluated on a combination of the previous tasks and the new task. The first term minimizes the generalization cost. Together, the second and the third terms minimize the catastrophic forgetting cost.

The first two terms can be obtained by measuring J_P and J_N directly. To obtain the third term in Eq. (5), we simplify the third and fourth term in Eq. (3). In Eq. 3, the third and the fourth term quantifies the change in the optimal cost due to the parameter updates and the change in the input respectively. Since, the boundary value for the optimal cost is the current performance of the model on a combination of all the previous tasks and the new task (how well the model performs on all the tasks till now), we use the current performance as an approximation of the optimal cost. This fact combined with the finite difference approximation results in the third term in Eq. (5). Updating with the third term in Eq. (5) regularizes the impact of the new task on the global solution. In Eq. (5) $\alpha(k) \leq \frac{1}{\beta\|J_{\boldsymbol{\theta}(\hat{k}))}\|^2+\epsilon}$, where β is a user-defined parameter and $\epsilon > 0$ is a small value to ensure that the denominator does not go to zero. *The derivation of the update rule and the discretization are presented in Appendix A.3*. The value β is the theoretical threshold on the optimal cost and the way we choose beta as the reciprocal of the learning rate.

Equipped with the gradient updates, we now describe the DPMCL algorithm. We define a new task sample, $\boldsymbol{\mathcal{D}}_N(k) = \{\mathcal{X}_k, \mathcal{Y}_k\}$, and a task memory (samples from all the previous tasks) $\boldsymbol{\mathcal{D}}_P(k) \subset \cup_{\tau=0}^{k-1}\boldsymbol{\mathcal{T}}_\tau$. We can approximate the required terms in our update rule Eq. 5 using samples (batches) from $\boldsymbol{\mathcal{D}}_P(k)$ and $\boldsymbol{\mathcal{D}}_N(k)$. The overall algorithm consists of two steps: generalization and catastrophic forgetting (see Algorithm 1 in Appendix B.4). DPMCL comprises representation and prediction neural networks parameterized by $\hat{\boldsymbol{\theta}}_1$ and $\hat{\boldsymbol{\theta}}_2$, respectively. Since, the vector space of $\hat{\boldsymbol{\theta}}$ is written as a combination of $\hat{\boldsymbol{\theta}}_1$ and $\hat{\boldsymbol{\theta}}_2$. The theory developed on $\hat{\boldsymbol{\theta}}$ extends to the vector space defined by the combination of $\hat{\boldsymbol{\theta}}_1$ and $\hat{\boldsymbol{\theta}}_2$. The solution in this space can be done achieved either by coordinate descent or by alternative minimization. We choose to update in an alternative minimization manner.

For each batch $b_N \in \boldsymbol{\mathcal{D}}_N(k)$, DPMCL alternatively performs generalization and catastrophic forgetting cost updates ρ times. The generalization cost update consists of computing the cost J_N and using that to update $\hat{\boldsymbol{\theta}}_1$ and $\hat{\boldsymbol{\theta}}_2$; the catastrophic forgetting cost update comprises the following steps. First we create a batch that combines the new task data with samples from the previous tasks $b_{PN} = b_P \cup b_N(k)$, where $b_P \in \boldsymbol{\mathcal{D}}_P(k)$. Second, to approximate the term $\big(J_{PN}(k;\hat{\boldsymbol{\theta}}(k+\zeta))\big)$,, we copy $\hat{\boldsymbol{\theta}}_2$ (prediction network) into a temporary network parameterized by $\hat{\boldsymbol{\theta}}_B$. We then perform ζ updates on $\hat{\boldsymbol{\theta}}_B$ while keeping $\hat{\boldsymbol{\theta}}_1$ fixed. Third, using $\hat{\boldsymbol{\theta}}_B(k+\zeta)$, we compute $J_{PN}(k;\hat{\boldsymbol{\theta}}_B(k+\zeta))$ and update $\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_2$ with $J_P(k) + (J_{PN}(k) - J_{PN}(k;\hat{\boldsymbol{\theta}}_B(k+\zeta)))$. The inner loop with ζ is purely for the purpose of approximating the optimal cost.

The rationale behind repeated updates to approximate $J_{PN}(k; \hat{\boldsymbol{\theta}}(k+\zeta))$, is as follows. At every instant k, $J_{PN}(k)$ (the cost on all the previous tasks and the new task) is the boundary value for the optimal cost as the optimal cost can only be less than $J_{PN}(k)$ (minimum of the cost can only be less than or equal to the current value). Therefore, if we start from $J_{PN}(k)$ and execute a Markov chain (repeated gradient updates) for ζ steps, the end point of this Markov chain is the optimal value of $J_{PN}(k)$ (the cost will reduce with repeated updates using the same batch of data) at instant k, provided ζ is large enough. Furthermore, the difference between the cost at the starting point and the end point of this chain provides us with a value that should be minimized such that the cost $J_{PN}(k)$ will reach the optimal value for the MCL problem at instant k. To execute this Markov chain, we perform repeated updates on a copy of $\hat{\boldsymbol{\theta}}_2$ (prediction network) denoted as $\hat{\boldsymbol{\theta}}_B$.

The goal of the markov chain procedure is purely to approximate the optimal cost, that is, how well, the current state of the model can predict on all the tasks that have been observed till now (previous tasks and the new task). The performance of the model depends on the prediction network and the representation network. Since, we seek to maintain a global solution, the pursuit is well served by learning a robust representation over all the tasks. Therefore, we want to observe, can the current representation allow us to reach the global solution over all the tasks? Therefore, we seek to estimate the difference $J_{PN}(k) - J_{PN}(k; \hat{\boldsymbol{\theta}}_B(k+\zeta))$, corresponding to the current representation and we treat the output of the representation network as input and measure the optimal cost by updating only a copy of $\hat{\boldsymbol{\theta}}_2$. This process allows us to measure the optimal cost as a function of the representation network and train the representation network toward minimizing optimal cost.

We repeat this alternative update process for each data batch in the new task. Once all the data from the new task is exhausted, we move to the next task.

3.3 Related work

Existing MCL methods can be grouped into three groups: (1) dynamic architectures and flexible knowledge representation (Sutton, 1990; Rusu et al., 2016; Yoon et al., 2017); (2) regularization approaches, ((Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi et al., 2018)) and (3) memory/experience replay, (Lin, 1992; Lopez-Paz and Ranzato, 2017; Chaudhry et al., 2019). Flexible knowledge representations seek to maintain a state of the whole dataset and require computationally expensive mechanisms (Yoon et al., 2017). Regularization approaches (Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi et al., 2018) attempt to minimize the impact of new tasks (changes in the input data-distribution) on the parameters of the model involving a significant trial-and-error process. Memory/experience replay-driven approaches (Lin, 1992; Chaudhry et al., 2019; Lopez-Paz and Ranzato, 2017) can address catastrophic forgetting but do not generalize well to new tasks.

In recent literature, the most comprehensive methodology that enables the study of the MCL problem was introduced in Finn et al. (2017). In Finn et al. (2017), the authors presented an approach where an additional term is introduced into the cost function (the gradient of the cost function with respect to the previous tasks). However, this method requires all the data to be known prior to the start of the learning procedure. To obviate this constraint, an online MAML approach was introduced in **?**Caccia et al. (2020). The approach does not explicitly minimize catastrophic forgetting but focuses on fast online learning. In contrast with Finn et al. (2017), our method can learn sequentially as the new tasks are observed. Although sequential learning was possible in Finn et al. (2019), it was highlighted in Caccia et al. (2020), that there is an inherent trade-off between memory requirements and catastrophic forgetting, which could be addressed by learning a representation over all the tasks as in Javed and White (2019); Beaulieu et al. (2020). Similar to Javed and White (2019); Beaulieu et al. (2020), our method allows a representation to be learned over the distribution of all the tasks $p(\boldsymbol{\mathcal{T}})$. However, both the representation and the training model are learned sequentially in DPCML, and we do not observe a pre-training step.

Our approach is the first comprehensive theoretical framework based on dynamic programming that is model agnostic and can adapted to different MCL setting in both CT and DT. Although theoretical underpinnings were provided in Finn et al. (2019) and Flennerhag et al. (2019), the focus was to provide structure for parameter updates but not attempt to holistically model the overall learning dynamics as is done in our theoretical framework. The key ideas in this paper have been adapted from dynamic programming and optimal control theory; additional details can be found in Lewis and Vrabie (2009).

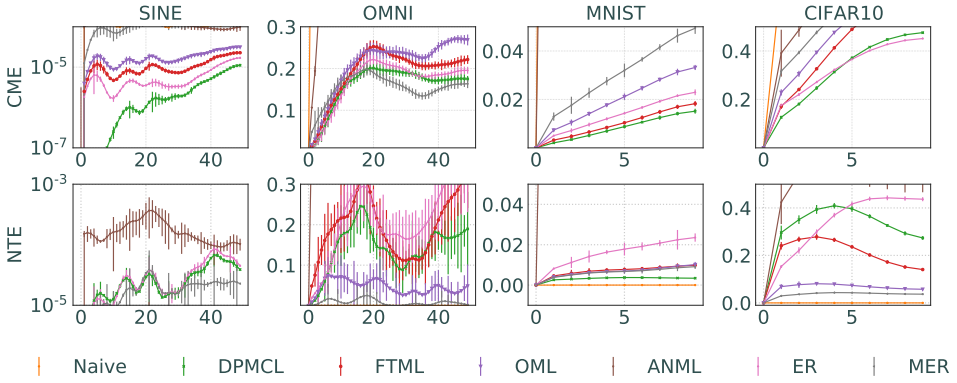

Figure 1: Top row: Cumulative error (CME) trends with respect to tasks; Bottom row: new task error (NTE) trends with respect to tasks. The error bars describe the area between $\mu + \sigma_{error}$ and $\mu - \sigma_{error}$ over 50 repetitions. Gaussian smoothing filter with standard deviation of 2 is applied on each trajectory.

4 EXPERIMENTS

We use four continual learning data sets: incremental sine wave (regression on 50 tasks *(SINE)*); split-Omniglot (classification on 50 tasks, *(OMNI)*); continuous MNIST (classification on 10 tasks and *(MNIST)*); and CIFAR10 (classification on 10 tasks, *(CIFAR10)*). All these data sets have been used in (Finn et al., 2019; 2017; Javed and White, 2019). We compare DPMCL with Naive (training is always performed on the new task without any explicit catastrophic forgetting minimization); Experience-Replay (ER) (Lin, 1992) (training is performed by sampling batches of data from all the tasks (previous and the new)); follow the meta learner (FTML) (Finn et al., 2019), online meta-continual learning (OML) (Javed and White, 2019), neuro-modulated meta learning (ANML) (Beaulieu et al., 2020) and meta experience replay (MER) (Riemer et al., 2018). To keep consistency with our computing environment and the task structure, we implement a sequential online version (Finn et al., 2019) (where each task is exposed to the model sequentially) of all these algorithms in our environment. For ANML and OML, we will run a pre-training phase for each new task. In particular, for each task, we will first train the RLN/NLM with data for the new task. Next, we train the complete network (RLN/NLM and PLN) with the data from the new task. For any particular data set, we set the same model hyper-parameters (number of hidden layers, number of hidden layer units, activation functions, learning rate, etc.) across all implementations. For fair comparisons, for any given task we also fix the total number of gradient updates a method can perform (See Appendix B.1,2,3 for details on data-set and hyper-parameters).

For each task, we split the given data into training (60%), validation (20%), and testing (20%) data sets. Methods such as ANML, FTML, and OML follow the two loop training strategy from FTML: the inner loop and the outer loop. The training data for each task is used for the inner loop, whereas the validation data is used in the outer loop. The testing data is used to report accuracy metrics. We measure generalization and catastrophic forgetting through cumulative error (CME) given by the average error on all the previous tasks and the new task error (NTE) given by the average error on the new task, respectively. For regression problems, they are computed from mean squared error; for classification problems, given as $(1 - \frac{Acc}{100})$, where Acc refers to the classification accuracy. For cost function, we use the mean squared error for regression and categorical cross-entropy for classification. We use a total of 50 runs (repetition) with different random seeds and report the mean μ and standard error of the mean ($\sigma_{error} = \sigma/\sqrt{50}$, where σ is the standard deviation with 50 being the number of repetition). We report only the σ_{error} when it is greater than 10^{-3}; otherwise, we indicate a 0 (See Appendix B.4,5 for implementations details).

4.1 RESULTS

We first analyze the `CME` and `NTE` as each task is incrementally shown to the model. We record the `CME` and `NTE` on the testing data (averaged over 50 repetitions) at each instant when a task is observed. The results are shown in Fig. 1. Unlike the other methods, DPMCL achieves low error

with respect to CME and NTE. ANML and OML perform poorly on CME because of the lack of a learned representation (as we do not have a pre-training phase to learn an encoder). The performance of DPMCL is better than MER, FTML and ER in all data sets except CIFAR-10, where ER is comparable to DPMCL. The poor performance of Naive is expected because it is trained only on the new task data and thus incurs catastrophic forgetting. DPMCL generalizes well to new tasks, and consequently the performance of DPMCL is better than FTML, ER, and ANML in all the data sets except CIFAR10. As expected, Naive has the lowest NTE. In the absence of a well-learned representation, OML exhibits behavior similar to Naive's and is able to quickly generalize to new task. MER also behaves similar to Naive and incurs very low NTE. For CIFAR10, FTML achieves NTE lower than that of DPMCL. ER struggles to generalize to a new task; however, the performance is better than ANML's, which exhibits the poorest performance due to the absence of a well-learned representation.

The CME and NTE values for all the data sets and methods are summarized in Table 1 in Appendix B.6. DPMCL achieves CME $[\mu(\sigma_{err})]$ of $1.12 \times 10^{-5}(0)$ for SINE, $0.175(0.007)$ for OMNI, $0.015(0.001)$ for MNIST, and $0.475(0.003)$ for CIFAR10. We note that the CME for DPMCL are the best among all the methods and all the data sets except MER for OMNI and ER for CIFAR10. In CIFAR10, ER demonstrates a 7% improvement in accuracy. For OMNI, MER outperforms DPMCL with an 1% improvement on the CME scale. On the NTE $[\mu\ (\sigma_{err})]$ scale, DPMCL achieves $3.89 \times 10^{-5}(0)$ for SINE, $0.189(0.091)$ for OMNI, $0.003(0)$ for MNIST and $0.273(0.008)$ for CIFAR10. On the NTE scale, the best-performing method is Naive followed by OML and MER; this behavior can be observed in the trends from Fig. 1. DPMCL is better than all the other methods in all the data sets except CIFAR 10, where FTML is better (observed earlier in Fig. 1) by 10%. However, this 10% improvement for FTML comes at the expense of 18% drop in performance on the CME scale. Similarly, although ER exhibits a 3% improvement on CME scale, DPMCL outperforms ER on the NTE scale by a 22.7% improvement.

The method closest to DPMCL in terms of design and performance is MER. MER outperforms DPMCL on the NTE scale in all the datasets except sine and MNIST where the performance is comparable. MER is expected to generalize really well with new tasks as it is designed for this purpose. MER regularizes the change in weight parameters with the introduction of a new task as the update rules are derived from Nichol et al. (2018). On the other hand, on the CME scale, DPMCL outperforms MER in all datasets except OMNI where the performance of MER is better. The reason is parameter drift (Narendra and Annaswamy, 1987) observed due to weight regularization present in MER. The parameter drift can make the network unlearn previous experiences especially when the new tasks are very similar to the older tasks (Narendra and Annaswamy, 1987). In OMNI, the new tasks are distinctly different from the older ones but in the rest of the datasets, the new tasks are similar. Such a parameter drift can be avoided when the weight is updated proportional to the cost (Narendra and Annaswamy, 1987) which is why DPMCL performs better on the CME scale.

Overall, the results show that DPMCL achieves a balance between CME and NTE. The balance can be engineered depending on the choice of ζ and κ (the study is presented in Appendix B.6.). We have chosen these parameters to achieve better performance in the CME scale. As a result, we show that DPMCL outperforms all methods in CME scale without significant drop in the NTE scale. All other methods either perform well on the NTE or CME scale but not on both.

5 Conclusions

We introduced a dynamic-programming-based theoretical framework for meta continual learning.Within this framework, catastrophic forgetting and generalization, the two central challenges of the meta continual learning, can be studied and analyzed methodically. Furthermore, the framework also allowed us to provide theoretical justification for intuitive and empirically proven ideas about generalization and catastrophic forgetting. We then introduced DPMCL which was able to systematically model and compensate for the trade-off between the catastrophic forgetting and generalization. We also provided experimental results in a sequential learning setting that show that the framework is practical with comparable performance to state of the art in meta continual learning.

In the future, we plan to extend this approach to reinforcement and unsupervised learning. Moreover, we plan to study different architectures such as convolutional neural networks and graph neural networks.

REFERENCES

R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

S. Beaulieu, L. Frati, T. Miconi, J. Lehman, K. O. Stanley, J. Clune, and N. Cheney. Learning to continually learn. *arXiv preprint arXiv:2002.09571*, 2020.

R. E. Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 2015.

M. Caccia, P. Rodríguez, O. Ostapenko, F. Normandin, M. Lin, L. Caccia, I. H. Laradji, I. Rish, A. Lacoste, D. Vázquez, and L. Charlin. Online fast adaptation and knowledge accumulation: A new approach to continual learning. *ArXiv*, abs/2003.05856, 2020.

A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, and M. Ranzato. Continual learning with tiny episodic memories. *arXiv preprint arXiv:1902.10486*, 2019.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

C. Finn, A. Rajeswaran, S. Kakade, and S. Levine. Online meta-learning. *arXiv preprint arXiv:1902.08438*, 2019.

S. Flennerhag, A. A. Rusu, R. Pascanu, F. Visin, H. Yin, and R. Hadsell. Meta-learning with warped gradient descent, 2019.

K. Javed and M. White. Meta-learning representations for continual learning. In *Advances in Neural Information Processing Systems*, pages 1818–1828, 2019.

J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

F. Lewis, S. Jagannathan, and A. Yesildirak. *Neural network control of robot manipulators and non-linear systems*. CRC Press, 1998.

F. L. Lewis and D. Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE Circuits and Systems Magazine*, 9(3):32–50, 2009.

F. L. Lewis, D. Vrabie, and V. L. Syrmos. *Optimal control*. John Wiley & Sons, 2012.

L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, 1992.

D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

K. Narendra and A. Annaswamy. A new adaptive law for robust adaptation without persistent excitation. *IEEE Transactions on Automatic Control*, 32(2):134–145, 1987. doi: 10.1109/TAC.1987.1104543.

A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms, 2018.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

M. Riemer, I. Cases, R. Ajemian, M. Liu, I. Rish, Y. Tu, and G. Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

J. Yoon, E. Yang, J. Lee, and S. J. Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR. org, 2017.