Meta-Continual Learning via Dynamic Programming

R. Krishnan\(^1\) and Prasanna Balaprakash\(^{1,2}\)
\(^1\)Mathematics and Computer Science Division
\(^2\)Leadership Computing Facility
Argonne National Laboratory
kraghavan,pbalapra@anl.gov

Abstract

Meta-continual learning algorithms seek to rapidly train a model when faced with similar tasks sampled sequentially from a task distribution. Although impressive strides have been made in this area, there is no theoretical framework that enables systematic analysis of key learning challenges, such as generalization and catastrophic forgetting. We introduce a new theoretical framework for meta-continual learning using dynamic programming, analyze generalization and catastrophic forgetting, and establish conditions of optimality. We show that existing meta-continual learning methods can be derived from the proposed dynamic programming framework. Moreover, we develop a new dynamic-programming-based meta-continual approach that adopts stochastic-gradient-driven alternating optimization method. We show that, on meta-continual learning benchmark data sets, our theoretically grounded meta-continual learning approach is better than or comparable to the purely empirical strategies adopted by the existing state-of-the-art methods.

1 Introduction

The central theme of meta-continual learning algorithms is to quickly and incrementally train a model on similar tasks revealed in a continual fashion. These algorithms seek to solve two fundamental challenges: catastrophic forgetting and generalization to new tasks\([7,9]\). To address these challenges, several approaches have been introduced in the literature\([3,7–9]\), where quick learning was achieved through the use of a second-order, derivative-driven approach introduced in\([7]\). Despite promising results, the meta-continual learning literature suffers from the following key issues: 1) there is a lack of a theoretical framework to systematically design and analyze meta-continual learning methods; 2) the use of fixed representations\([9]\) or pre-trained embedding\([3]\) limits the ability to compensate for sensitivity in the data distribution, as demonstrated in\([4]\); 3) the data corresponding to all the tasks needs to be known in advance\([8]\), which is often impractical, because in real-world environments the tasks are observed sequentially\([9,3]\), and 4) approaches described in\([9,3]\) build on\([7]\) and therefore they can suffer from the disadvantages of use of second-order derivative such as poor local minima and saddle points\([6]\).

To address these issues, we introduce a theoretical framework in which we approach the problem of meta-continual learning from a dynamic programming perspective. The problem of meta-continual learning is first posed as the minimization of a cost (loss) function that is integrated over the lifetime of the model. At a given instant \(t\), we have access to tasks until time \(t\) and do not know about the future tasks. This makes the integral calculation intractable. Therefore, we use principles from dynamic programming to recast this problem to minimizing the sum of generalization and catastrophic forgetting cost. We present a theoretical analysis on generalization and catastrophic forgetting using the proposed dynamic programming framework. Moreover, we demonstrate that the continual learning approaches proposed in\([3,7,9]\) can be derived from the framework.

Preprint. Under review.
Next, we derive a theoretically-grounded dynamic programming based meta-continual learning (DPMCL) approach for supervised learning, where similar tasks sampled from a task distribution are available sequentially as in [8, 9, 3]. See Fig. 1 for an illustration. In this approach, we seek to minimize the current cost and the future cost. The current cost measures the generalization by evaluating the model on the current task data. The future cost measures the catastrophic forgetting by evaluating the model on the task memory (all the tasks that have been observed till now) when the model is trained on the new task. We alternately minimize the current cost to improve generalization to new task and the future cost to reduce catastrophic forgetting for a predefined number of iterations to achieve a balance between the two.

We analyze the performance of the DPMCL approach experimentally on classification and regression benchmark data sets. Overall, the key contributions of the paper are (1) a new dynamic-programming-based theoretical framework for meta-continual learning; (2) a theoretically-grounded meta-continual learning approach with convergence properties that either outperforms or achieves similar performance to the existing meta-continual learning methods.

2 Problem Statement

We let \( \mathbb{R} \) denote the set of real numbers and use boldface to denote vectors and matrices. Let the overall lifetime of the model be described by \( [0, \Gamma] : \Gamma \in \mathbb{R} \) and let the distribution over tasks be denoted by \( p(\mathcal{T}) \). Let a task \( \mathcal{T}_t \) be sampled from \( p(\mathcal{T}) \) and provided to us sequentially at an instant \( t \in [0, \Gamma] \). Let \( \mathcal{T}_t = (\mathcal{X}_t, \mathcal{Y}_t) \) be a tuple where \( \mathcal{X}_t \in \mathbb{R}^{n \times p} \) denotes the input data and \( \mathcal{Y}_t \in \mathbb{R}^{n \times 1} \) denotes the target labels (output). The number of samples is described by \( n \) and the dimensions in the data are described by \( p \). For a few-shot learning problem, \( n \) can be a small value, and for a general learning problem, \( n \) can be large. We consider a general supervised learning setting and the goal is to find the best approximation of the association between inputs and targets through a parametric model \( g(\cdot) \) with parameters \( \theta \). Therefore, we can write \( y(t; \tilde{\theta}(t)) = g(x(t); \tilde{\theta}(t) + \epsilon) \), where \( \epsilon \) is the approximation error. Although, we focus on neural networks, any parametric form can be utilized with our framework. We define the expected value of the loss (cost function) at instant \( t \) as \( J(t, y(t; \tilde{\theta}(t))) \), which can be thought of as feedback from task \( \mathcal{T}_t \) based on the model \( g \). This cost function can be the cross entropy (for classification) or mean squared error (for regression) or as required by the problem in hand.

The goal of meta-continual learning is to achieve a small error on all the tasks observed in the past and those that will be observed in the future, that is maintain a low cumulative cost over all the tasks over the lifetime of the model. A cumulative cost can be written as an integral over the lifetime of the model, i.e. \( t \in [0, \Gamma] \). Therefore, the minimization problem to minimize the cumulative cost can be written as

\[
V^*(t) = \min_{\theta(\tau) \in \Omega: t \leq \tau \leq \Gamma} \int_{\tau=t}^{\Gamma} \gamma(\tau) J(\tau, y(\tau; \tilde{\theta}(\tau))) \ d \tau, \tag{1}
\]

Figure 1: Illustration of the DPMCL method. The proposed approach can be described in two steps, compensation for the current cost and compensation for catastrophic forgetting. Compensation for the current cost is performed by evaluating the model on the current task. Compensation for the future cost is obtained by observing the change in the output of the model due to the change in the parameters of the model. These two steps are alternatively repeated for a pre-defined number of iterations to achieve a balance between the two costs.
where $V^*(t)$ is referred to as the optimal cumulative cost. The choice of the parameter $\gamma$ ensures that the integral is bounded (details are provided in Lemma 1). Note from Eq. (1) that the minimization distributes over all the terms in the integral and therefore the optimal cost at instant $t$ can be thought of as an integral over the optimal cost through the lifetime of the model. Therefore, we would seek a sequence of parameters $t \leq \tau \leq \Gamma$ that would provide an optimal cost for each term in the integral. The solution to the meta continual problem as described in Eq. (1) is a sequence of parameters over the lifetime of the model such that a low optimal cumulative cost is obtained for all $t$. However, we only have data corresponding to all the tasks prior to instant $t$, thus solving this problem in Eq. (1) is intractable.

In addition, there are two challenges that are faced while addressing this problem: generalization to new tasks and catastrophic forgetting. Although there have been promising empirical results in the literature [7][9][3] that address both these challenges, to the best of our knowledge, there is no theoretical framework to systematically analyze and address these challenges. To circumvent this issue, we will take a dynamic programming view of the meta-continual learning problem and introduce a novel theoretical framework.

3 Theoretical Meta-Continual Learning Framework

The meta-continual learning problem involves estimating a sequence of parameters to solve the optimization problem in Eq. (1), which is intractable. Using ideas from dynamic programming (Bellman’s principle of optimality [12]) to simplify the problem, we can write the learning problem as a partial difference equation (PDE):

\[
- \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t} = \min_{\hat{\theta}(t) \in \Omega} \left[ \gamma(t) J(t, y(t; \hat{\theta}(t))) \right] + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right)^T \Delta \hat{\theta}(t) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta x(t)} \right)^T \Delta x(t)
\]

(2)

where $V^*$ describes the optimal cost and $(\cdot)^T$ refers to the transpose operator. Observe here that the problem of generating a sequence of parameters for each task in $[0, \Gamma]$ is now cast as the problem of generating an optimal parameter set for just task $t$. The full derivation is provided in appendix A.

The second term in Eq. (2) describes the change in the optimal cost with respect to the model (sensitivity due to model) and the third term describes the change in the optimal cost with respect to the change in the data (sensitivity due to data). The third term in Eq. (2) (which is the change in the cost due to the change in the data) is not completely quantifiable. In dynamical systems, where there is a known differential equation for the change in the state (data), we can directly substitute the state changes into this PDE and the solution of the problem will be exact. However, we have no representations of the system under consideration in meta-continual learning setting. Therefore, this term denoted as $\epsilon_x$ must be assumed bounded. Intuitively, we know that we can only compensate for small changes in the distribution: that is, transfer knowledge between similar tasks. If the change is large (going from predicting on images to understanding texts), any approach (including ours) will fail. The need for a bound on $\epsilon_x$ highlights this issue. (Additional insights are provided after Theorem 1). The PDE can be rewritten as

\[
-(\frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t}) + \epsilon_x = \min_{\hat{\theta}(t) \in \Omega} \left[ \gamma(t) J(t, y(t; \hat{\theta}(t))) \right] + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right)^T \Delta \hat{\theta}(t)
\]

(3)

The main learning problem is therefore to find $\hat{\theta}(t)$ that satisfies the PDE described in Eq. (3). This PDE is also known as the Hamilton-Jacobi Bellman equation in optimal control [12].

Different approaches in the literature differ only in their choice of their cost function $J(t, y(t))$. For example, continual supervised learning would involve using cross entropy cost for classification and mean squared error for regression. Several existing meta-learning approaches can be easily derived from our framework. For instance, consider the learning process in OML/MAML [8]. There are two loops in MAML. The goal of the inner loop is to update the model on the current task, and the goal
of the outer loop is to update with data corresponding to all the tasks. If we take each term on the right side of Eq. (3) as a loop, we will achieve the OML learning approach, as the first term provides the current task update. Furthermore, if we choose to perform the update for the second term using the second-order derivatives, we can arrive at the outer loop in MAML/OML. With the choice of different architectures for the neural network, all of the approaches that build on MAML/OML as in [9, 3] can be directly derived. We therefore observe that our framework allows the flexibility to derive different algorithms and models for meta-continual learning.

3.1 Analysis

Here, we will analyze the problem setup to see if it is feasible as an optimization problem. According to Eq. (1), we are working with an infinite horizon integral when \( \Gamma \rightarrow \infty \). The first point of analysis therefore is to decide whether this integral has a converging point. We demonstrate the existence of this point under mild assumptions in the next lemma.

**Lemma 1.** Let \( V(t) = \lim_{\Gamma \rightarrow \infty} \int_{0}^{\Gamma} \gamma(t)J(t, y(t))dt \) and let \( J(t, y(t)) \) be a twice differentiable cost function and be bounded. Consider \( \gamma(t) = \frac{1}{t+1} \) to be a monotonically decreasing sequence. Then \( V(t) \) is bounded and has a converging point. **Proof:** See appendix A.

The following corollary is immediate.

**Corollary 1.** Let \( V = \lim_{\Gamma \rightarrow \infty} \int_{0}^{\Gamma} \gamma(t)J(t, y(t))dt \), let \( J(t, y(t)) \) be twice differentiable and bounded. Consider \( \gamma(t) \) to not be a monotonically decreasing sequence then \( V \) is divergent. **Proof:** Direct consequence of Lemma 1.

Corollary 1 implies that if each task contributes equally to the cost, the integral will diverge and cannot be minimized. It can be intuitively understood from Corollary 1 that no methodology can have infinite memory. This implies that if each term in the integral contributes large values, the function may not be integrable and therefore cannot be minimized. One may note that by introducing a forgetting factor, we can control the memory loss by forgetting old tasks. Consequentially, we can use \( \gamma(t) \) to control catastrophic forgetting. Since our cost has a converging point, Banach fixed-point theorem allows us to reach this converging point under certain conditions [2] (convergence proof will be similar to the proof of generalized policy iterations provided in [12]). The goal of the following analysis is to comment on the goodness of the point of convergence \( V^* \) in a meta-continual learning setting. The question we are asking is, can we keep optimal cost \( V^* \) small and bounded? We therefore present the following theorem.

**Theorem 1.** Let \( V^*(t) = \min_{\theta \in \Omega} \int_{t}^{T} \gamma(\tau)J(\tau, y(\tau; \theta(\tau)))d\tau \) be the optimal cost and let \( J(t, y(t)) \) be twice differentiable cost function and bounded with Lemma 1 being true. Consider \( \gamma(t) = \frac{1}{t+1} \) to be a monotonically decreasing sequence and let 
\[
\left( \frac{\delta V^*(t, y(t; \theta(t)))}{\delta x(t)} \right)^T \Delta x(t) \leq \epsilon_B.
\]
Let the update for the model be provided as \( \Delta \theta = \frac{\delta V^*(t, y(t; \theta(t)))}{\delta \theta(t)} \). The first derivative of the optimal cost is negative semi-definite and therefore the optimal cost is ultimately bounded. **Proof:** See appendix A.

The optimal cost cannot reach zero and can only be small and bounded due to \( \epsilon_B \). Since, \( \epsilon_B \) describes the boundedness of change in the data, this idea reinforces another point we made earlier, where we mentioned how we are limited by the lack of prior knowledge of the tasks and we cannot compensate for large changes in the data distribution (tasks). Another point to highlight here is that, if we can explicitly track the change in the data distribution, compensation for \( \epsilon_B \) is possible. The point describes the need for strong representation learning methodologies for addressing the issue [9, 3].

As a consequence of Theorem 1, the update for the parameters is provided by 
\[
\frac{\delta V^*(t, y(t; \theta(t)))}{\delta \theta(t)}.
\]
Since, \( V^* \) is not completely known, we will have to approximate this gradient. We will derive this approximation in the next section.

3.2 Dynamic programming-based meta-continual learning (DPMCL)

We will discretize the overall setup for use in practical scenarios. let \( k \) be the discrete sampling instant such that \( t = k \Delta t \), where \( \Delta t \) is the sampling interval. Let the parametric map be written
as $\hat{y} = g(h(x; \hat{\theta}_1); \hat{\theta}_2)$, where the inner map $h(.)$ can be understood as a memory (representation learning network) for the past tasks and the future tasks (see Step 1 in Fig. 1). Let $\theta = [\theta_1, \theta_2]$ be the parameters of interest; the overall map can then be denoted as $\hat{y}(\theta(t))$. We can discretize Eq. (2) through Euler’s discretization and derive our update rule through finite difference approximation (11):

$$\hat{\theta}(k + 1) = \hat{\theta}(k) - \alpha \nabla_{\hat{\theta}(k)} \left[ \gamma(k)J(k) + \beta[V^*(k + 1) - V^*(k)] + \gamma(k)[V^*(k)] \right].$$  (4)

(The derivation and the discretization are presented in full in the appendix A.) With a little abuse of notation we define $\beta = \frac{1}{\Delta t}$ and $\gamma(k) = \gamma(k) + \beta$, where $\Delta t$ is the time between two tasks. We define a current data-sample (that is the sample at instant $k$, $\mathcal{D}(k) = \{X_k, y_k\}$) and a task memory ($\mathcal{D}(k) \subset \bigcup_{k=0}^{k} \mathcal{T}_k$). We can approximate different terms in our update rule using the samples from $\mathcal{D}(k)$ and $\mathcal{D}(0)(k)$. The first term (See Step 1 in Fig. 1) can be approximated by directly evaluating the gradient using samples from $\mathcal{D}(k)$. The second term (See Step 2 in Fig. 1) in the update rule cannot be written directly. We will approximate the second term using finite difference approximation through samples from $\mathcal{D}(k)$.

Consider now a small sample $b \in \mathcal{D}(k) \cup \mathcal{D}(0)(k)$. This small sample can be understood as a batch of data. For this batch of data, we can write $V^*(k) = \gamma(k)J_b(k)$, where the subscript $b$ denotes the use of batch $b$ for evaluating the cost. The term $V^*(k + 1) = \gamma(k)J_b(k)$, can then be approximated by updating the weight $\theta$ for a predefined number of iterations $\zeta$ using the data batch $b$.

Equipped with the gradient update rules, we develop a dynamic-programming-based meta-continual learning (DPMCL). See Figure 1, the pseudo-code is described in the appendix B. We choose a network with two models, one is the representation learning network and the other is a prediction learning network. For a batch of data from the task at $k$, we will update the parameters of the model for the current task. However, learning to predict a new task might erase the memory of old tasks. Therefore, we will compensate for catastrophic forgetting (the second and the third term in the update provided in Eq. (4)). We will sample a batch of data from the task memory and approximate both $V^*(k)$ and $V^*(k + 1)$. With these two approximations, we will compensate for the future cost. We will alternately update with the current cost and the future cost for $\kappa$ number of iteration or until all batches of data from the task are exhausted. After this process, we will move to the next task and repeat the procedure for the next task.

3.3 Related work

Several methodologies have been introduced to address the problem of continual learning and they can be broadly classified in four classes: 1) dynamic architectures and flexible knowledge representation [17][16][18]; 2) regularization approaches, (10, 19, 11); 3) memory/experience replay, (14, 5, 15); and 4) meta-continual learning approaches [9, 7, 8]. Flexible knowledge representations aimed at building recurrent neural networks based representation for keep track of changes in the tasks. However, these approaches require computationally expensive mechanisms and are limited by the capacity of the computing environment. Regularization approaches [17, 16, 18, 14, 5, 15] attempt to compensate for changes in the data distribution in advance. However, as we demonstrated in Section 3.1 through theoretical analysis, these approaches are subject to failure because there is no way estimate the distribution changes in advance. Memory/experience replay-driven approaches can address catastrophic forgetting but may be inflexible to new information because they are not suitable for quick adaptation to new data.

Meta-continual learning approaches have been investigated in [7][8]. In [7], the authors presented a method where an additional term is introduced into the cost function (the gradient of the cost function with respect to the older tasks). The additional term signifies the concept of sensitivity of the model to older tasks. However, this method required all the data to be known prior to the start of the learning procedure. To obviate this constraint, an online meta-learning approach was introduced in [8] which is the closest to DPMCL. However, the method in [8] still suffers from the need to evaluate a second order derivative.

In contrast with [7], our method can learn sequentially as the new tasks are observed and in contrast with [7][8], our algorithm does not suffer from calculation of a second order derivative. Although sequential learning was possible in [8], the work highlighted the fact that there is an trade-off between
We will evaluate the efficacy of the proposed meta-continual learning method on two commonly used ANML implementations, we use 100 hidden units for incremental sine wave, 1000 hidden units for split-Omniglot with relu activations (same as DPMCL). Moreover, we keep the batch size, learning rate and other hyperparameters the same across these different implementations (details will be given in the Appendix for reproducibility). For the implementations of Naive, ER, and OML, a single six-layer network (1000 hidden units, relu activation) is utilized for both the datasets. For CML [9], we use two networks: representation learning network and prediction learning networks, respectively. For ANML, the representation learning network becomes the neuro-modulatory network [3]. To keep consistency with our computing environment and the task structure, we implement a sequential online version [8] (where each task is exposed to the model sequentially) of all these algorithms in our environment. For CML [9], we use two networks: representation learning network and prediction network. The value of \( \kappa \) is set to 10.

For each task, we split the given data into training (80%), validation (20%) and testing (20%) data sets. Meta learning methods such as OML, CML and ANML [8, 9, 3] consists of two loops. The training model are learned sequentially in DPCML. Furthermore, we do not need a pre-training step. Finally and most importantly, our approach is a comprehensive framework that allows choice of cost functions and architecture. These choices lead to different algorithms, some of which have been already proposed in the literature [9, 7, 8, 3]. The key the ideas in this paper have been adapted from optimal control theory and additional details can be found in [11].

### 4 Experiments

We will evaluate the efficacy of the proposed meta-continual learning method on two commonly used continual learning data sets: incremental sine wave (regression) and split-Omniglot (classification). These data sets have been used in [8, 7]. We generate a sine wave data set consisting of 50 tasks. Each task is shown sequentially to the model and is described by its amplitude, phase, frequency and the time \( t \in \{0, 0.001, \ldots, 0.01\} \). Each task is generated by making incremental changes to amplitude, phase and frequency while following the protocol in [7]. The split-Omniglot is a data set with 1623 characters from 50 different alphabets, with just 20 images per class. We randomly choose a total of 50 classes where each class of the data is shown to the model sequentially. (See appendix B for more details on the data sets). The code is available at https://github.com/krm9c/Meta-Continual-Learning.git

For comparison, we consider a number of methods: Naive (no attempt in this approach is made to minimize catastrophic forgetting, training is performed only on the current task data); Experience-Replay (ER) [14] (training is performed by sampling batches of data from the task memory and not the current task data); online-meta learning (OML) [8], online meta-continual learning (CML) [9], neuro-modulated meta learning (ANML) [3].

For the incremental sine wave data, the input is phase \( \mathbb{R}^{1 \times 1} \), frequency \( \mathbb{R}^{1 \times 1} \) and the amplitude \( \mathbb{R}^{1 \times 1} \) therefore, the input vector is \( 3 \times 1 \). DPMCL uses two neural networks: one for the representation and one for the prediction. Both are three layered (one input, one hidden, and one output layer) feed-forward network with 100 hidden layer neurons and relu activation function. The output of the prediction network is a \( 1000 \times 1 \) vector with a linear activation function at the output layer. For the split-Omniglot data, the input is of size 784 and output is a one-hot encoded vector of 50 dimensions. Similar to the incremental sine wave data, DPMCL uses two three layered neural networks with 1000 hidden units of relu and softmax output for the prediction network. The value of \( \kappa \) is set to 10.

The testing data is used to report accuracy metrics. We measure generalization and catastrophic forgetting through cumulative task error (CME) and current task error (CTE) respectively. For regression problems, they are computed from mean squared error; for classification problems, they are 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 cross-entropy for classification. Additional details on the implementations are provided in the Appendix B. Our implementations are done in python with the Pytorcu library.
| Method          | Naive CME (sine wave) | Naive CME (split-Omniglot) | DPMCL CME (sine wave) | DPMCL CME (split-Omniglot) | CML CME (sine wave) | CML CME (split-Omniglot) | ANML CME (sine wave) | ANML CME (split-Omniglot) | OML CME (sine wave) | OML CME (split-Omniglot) | ER CME (sine wave) | ER CME (split-Omniglot) |
|-----------------|-----------------------|-----------------------------|-----------------------|-----------------------------|---------------------|--------------------------|-----------------------|--------------------------|---------------------|--------------------------|---------------------|--------------------------|
| CME (sine wave) | 1.048(0.036)          | 0.979(0.002)                | 0.006(0.003)          | 0.385(0.043)                | 0.081(0.243)        | 0.948(0.032)             | 5.123(0.576)          | 0.767(0.108)              | 1.322(1.057)        | 0.392(0.162)              | 1.832(1.057)        | 0.001(0.027)              |
| CME (split-Omniglot) | 0.979(0.002)          | 0.767(0.108)                | 0.081(0.243)          | 0.948(0.032)                | 5.123(0.576)        | 0.006(0.003)             | 0.385(0.043)          | 1.322(1.057)              | 0.767(0.108)        | 0.392(0.162)              | 1.832(1.057)        | 0.001(0.027)              |

Table 1: Cumulative task error (CME) for the incremental sine wave data set (first row) and the split-Omniglot data set (second row). Mean and standard deviations are calculated by averaging across all the tasks over hundred runs.

| Method          | Naive CTE (sine wave) | Naive CTE (split-Omniglot) | DPMCL CTE (sine wave) | DPMCL CTE (split-Omniglot) | CML CTE (sine wave) | CML CTE (split-Omniglot) | ANML CTE (sine wave) | ANML CTE (split-Omniglot) | OML CTE (sine wave) | OML CTE (split-Omniglot) | ER CTE (sine wave) | ER CTE (split-Omniglot) |
|-----------------|-----------------------|-----------------------------|-----------------------|-----------------------------|---------------------|--------------------------|-----------------------|--------------------------|---------------------|--------------------------|---------------------|--------------------------|
| CTE (sine wave) | 0(0)                  | 0.400(0.48)                 | 0.049(0.062)          | 0.001(0.001)                | 0.001(0.001)        | 0.001(0.001)             | 0.400(0.48)          | 0.049(0.062)              | 0.001(0.001)        | 0.001(0.001)              | 1.42(0.270)         | 1(0)                     |
| CTE (split-Omniglot) | 0.400(0.48)          | 1(0)                        | 0(0)                  | 0.049(0.062)                | 0.001(0.001)        | 0.001(0.001)             | 1(0)                  | 0(0)                     | 0.049(0.062)        | 0.001(0.001)              | 1(0)                | 1(0)                     |

Table 2: Current task error (CTE) $\mu(\sigma)$ (the lower the score, the better the performance) for the incremental sine wave and the split-Omniglot data set (second row) for the testing data corresponding to the 50th task over hundred runs.

We use a total of 100 runs (repetition) with different random seeds and report mean and standard deviation.

### 4.1 Results

![Graph](image1.png)  
![Graph](image2.png)  

Figure 2: Cumulative task error (CME) and current task error trends (CTE) for the (a) incremental sine wave data set (log scale) (b) for the split-Omniglot data set (regular scale–accuracy is converted to error percentage by subtracting the values from one). To generate these plots, we calculate the mean $\mu$ and standard deviation $\sigma$ of the cumulative task error and the current task error over hundred runs. The error band describes the area between $\mu + \sigma$ and $\mu - \sigma$. The closer the error band is to zero, the better the performance.

We show that theoretically-grounded DPMCL maintains a balance between cumulative task error (generalization) and current task error (catastrophic forgetting) and it is better than or comparable to other meta-continual learning methods. To substantiate this, we first analyze the test cumulative task error and current task error as each task is incrementally shown to the model. At each instant when a task is observed, cumulative task error and current task error values are recorded on the testing data. The trends of cumulative task error and current task error with respect to each task are shown in Fig 2. In Tables 1 and 2, we report the mean and standard deviation of cumulative task error and current task error calculated on the test data (averaged over hundred repetitions).

First, we will discuss the cumulative task error results for both the data sets. DPMCL achieves cumulative task error values with $\mu(\sigma) = 0.006(0.003)$ for the sine wave data set and $\mu(\sigma) = 0.38(0.04)$ for the split-Omniglot. From Fig 2a, we observe that DPMCL is comparable to ER but outperforms all the other methods. Similar trends are observed on the split-Omniglot data set. An exception is that OML obtains cumulative task error values that are comparable to DPMCL.
With respect to current task error trends, DPMCL outperforms all the other methods except OML. ER performs poorly but Naive achieves the best performance. Similar trend is observed on the split-Omniglot dataset. We observe significant variance in current task error values of the Naive on split-Omniglot data set because the number of images for each task is rather small.

The performance of Naive is poor on the cumulative task error scale because it is trained purely on current task data as evidenced by good performance on the current task error scale. ER is comparable to DMCL on the cumulative task error scale because it is trained with multiple passes over the task memory, however, its performance is poor on current task error. OML seeks to learn quickly and efficiently on a new task; therefore, the OML is comparable to DPCML on the current task error trends but not on cumulative task error scale. ANML and CML perform poorly on both cumulative task error and current task error in our study due to the lack of a learned representation (as we do not have a pre-training phase to learn an encoder). The insight here is that while other methods perform well on either current or cumulative task error, DPMCL is able to achieve a balance between the two.

The balancing behavior is shown in Fig 3a where the trends of cumulative task error and current task error for the sine wave data set with respect all the tasks are shown. We can observe that there is an initial phase of learning. Once this initial phase has passed, the cumulative task error and current task error trends oscillate together as observed by the coinciding peaks and valleys in Fig 3a. The value for the hyperparameter $\kappa$ decides the number of alternating steps that must be utilized to achieve balance between generalization to new tasks and catastrophic forgetting. We analyze the performance of DPMCL with different choices of $\kappa$ and plot the cumulative task error/current task error values of the to the last few tasks in Fig 3b. We observe with $\kappa = 10$ (the default value), DPMCL achieves low cumulative task error and low current task error. However, for $\kappa = 20$ (high), the cumulative task error values improve but current task error values are poor whereas for $\kappa = 2$ (low) current task error improves slightly but cumulative task error values are poor. These results show that large $\kappa$ reduces catastrophic forgetting but at the cost of generalization whereas low $\kappa$ value can improve generalization but increases catastrophic forgetting.

5 Conclusion

We introduced a dynamic-programming-based theoretical framework for meta-continual learning, which encompass known meta-continual learning methods. Within this framework, catastrophic forgetting and generalization, the two central challenges of the meta-continual learning, can be studied and analyzed in a theoretically grounded way. Furthermore, the framework also allowed us to provide theoretical justification for intuitive and empirically proven ideas about generalization and catastrophic forgetting. We then introduced a new dynamic meta-continual learning, 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 current study, we demonstrated the efficacy of our approach to supervised learning with feed forward networks. In the future, we seek to extend this approach to reinforcement and unsupervised learning. Moreover, we seek to study different architectures such as convolutional neural networks and graph neural networks.
**Broader Impact**

The paper presents a framework for continual learning. Since continual learning is one of the first steps for learning to imitate the mind, the proposed framework is of immense importance. Researchers can use the proposed framework to establish and offer new methodologies that can act as building blocks to learning continually in real-life environments. We do not believe there is any direct harm coming from this methodology nor is there any broader risk of the failure of this approach.

**Acknowledgments and Disclosure of Funding**

This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory. The authors also acknowledge Dr. Marieme Ngom and Dr. Raghu Bollapragada, Post-doctoral Researchers, Argonne, Dr. Vignesh Narayanan, Post-doctoral Researcher, WUSTL, for the discussions and their valuable insights.

**A Derivation and Proofs**

For notations, we will let $V^*(t, y(t; \hat{\theta}(t)))$ be the optimal cost that is dependent on $y(t)$, $x(t)$ and $\hat{\theta}(t)$, where $y(t)$ is the model, $x(t)$ is the input data and $\hat{\theta}(t)$, are the model parameters.

**A.1 Derivation of Hamilton-Jacobi Bellman**

Let the optimal cost be given as

$$V^*(t, y(t; \hat{\theta}(t))) = \min_{\theta(t) \in \Omega, t \leq \tau \leq \Gamma} \left[ \int_{\tau=t}^{\Gamma} \gamma(\tau) J(\tau, y(\tau; \hat{\theta}(\tau))) d\tau \right].$$ \hspace{1cm} (5)

We split the interval $[t, \Gamma]$ as $[t, t+\Delta t]$ and $[t+\Delta t, \Gamma]$. With this split, we rewrite the cost function as

$$V^*(t, y(t; \hat{\theta}(t))) = \min_{\theta(t) \in \Omega, t \leq \tau \leq \Gamma} \left[ \int_{\tau=t}^{t+\Delta t} \gamma(\tau) J(\tau, y(\tau; \hat{\theta}(\tau))) d\tau \right. \hspace{1cm} (6)

+ \left. \int_{\tau=t+\Delta t}^{\Gamma} \gamma(\tau) J(\tau, y(\tau; \hat{\theta}(\tau))) d\tau \right].$$

With $V(t) = \int_{\tau=t}^{\Gamma} \gamma(\tau) J(\tau, y(\tau; \hat{\theta}(\tau))) d\tau$ note that $\int_{t+\Delta t}^{\Gamma} \gamma(\tau) J(\tau, y(\tau; \hat{\theta}(\tau))) d\tau$ is $V$ at $t + \Delta t$ and can be defined as $V(t + \Delta t, y(t + \Delta t; \hat{\theta}(t + \Delta t)))$, which provides

$$V^*(t, y(t; \hat{\theta}(t))) = \min_{\theta(t) \in \Omega, t \leq \tau \leq t+\Delta t} \left[ \int_{\tau=t}^{t+\Delta t} \gamma(\tau) J(\tau, y(\tau; \hat{\theta}(\tau))) d\tau \right. \hspace{1cm} (7)

+ \left. V(t + \Delta t, y(t + \Delta t; \hat{\theta}(t))) \right].$$

Suppose now that all information for $\tau \geq t + \Delta t$ is known. With this information, we consider those candidate costs for the future that are the optimal costs, and we suppose that all optimal configurations of weights are known. Then, the only input weights to be obtained are for the sequence $t \leq \tau \leq t + \Delta t$. We can therefore write

$$V^*(t, y(t; \hat{\theta}(t))) = \min_{\theta(t) \in \Omega, t \leq \tau \leq t+\Delta t} \left[ \int_{\tau=t}^{t+\Delta t} \gamma(\tau) J(\tau, y(\tau; \hat{\theta}(\tau))) d\tau \right]$$

$$+ \left. V^*(t + \Delta t, y(t + \Delta t; \hat{\theta}(t + \Delta t))) \right].$$ \hspace{1cm} (8)
Now, we approximate the future cost using the information provided at the current stage. To do so, we further simplify this framework. If we write the first-order Taylor series expansion on the second term, we can write

\[
V^*(t + \Delta t, y(t + \Delta t; \hat{\theta}(t + \Delta t))) = V^*(t, y(t; \hat{\theta}(t))) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t} \right) T \Delta t
\]

Substituting into the original equation, we have

\[
V^*(t, y(t; \hat{\theta}(t))) = \min_{\hat{\theta}(\tau) \in \Omega_t \leq \tau \leq t + \Delta t} \left[ \int_{\tau=t}^{t+\Delta t} \gamma(\tau) J(\tau, y(\tau; \hat{\theta}(\tau))) d\tau \right. \\
\left. + V^*(t, y(t; \hat{\theta}(t))) \right]
\]

Upon cancellation of common terms we have

\[
V^*(t, y(t; \hat{\theta}(t))) = V^*(t, y(t; \hat{\theta}(t))) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t} \right) T \Delta t
\]

The terms \( V^*(t, y(t; \hat{\theta}(t))) + \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t} T \Delta t \) can be brought outside the minimization because they are independent of \( \tau \), the sequence being selected. Therefore,

\[
V^*(t, y(t; \hat{\theta}(t))) = V^*(t, y(t; \hat{\theta}(t))) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t} \right) T \Delta t
\]

Upon cancellation of common terms we have

\[
- \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t} \right) T \Delta t = \min_{\hat{\theta}(\tau) \in \Omega_t \leq \tau \leq t + \Delta t} \left[ \int_{\tau=t}^{t+\Delta t} \gamma(\tau) J(\tau, y(\tau; \hat{\theta}(\tau))) d\tau \right. \\
\left. + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right) T \hat{\theta}(t) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta x(t)} \right) T \Delta x(t) \right]
\]

Letting \( \Delta t \to 0 \), we obtain

\[
- \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t} \right) = \min_{\hat{\theta}(\tau) \in \Omega_t} \left[ \gamma(t) J(t, y(t; \hat{\theta}(t))) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right) \right. \\
\left. + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta x(t)} \right) \right] T \Delta x(t)
\]

This is the Hamilton-Jacobi-Bellman equation that is specific to the meta-continual learning problem.

### A.2 Proof of Lemma 1

**Proof of Lemma 1.** We know that the cost is given as \( V = \lim_{\Gamma \to \infty} \int_{t=0}^{\Gamma} \gamma(t) J(t, y(t)) dt \). Letting \( \Gamma \to \infty \), we have \( V(t) = \int_{t=0}^{\infty} \gamma(t) J(t, y(t)) dt \). Consider \( J(t, y(t)) \leq L, \forall t \) as \( J(t, y(t)) \) assumed bounded. Therefore, \( V(t) \leq \int_{t=0}^{\infty} \gamma(t) L dt \). Since \( \gamma(t) \) is a function of choice, we can choose \( \gamma(t) = \frac{1}{(t+1)^{\Gamma}}, \forall t \). We get \( V(t) \leq \int_{t=0}^{\infty} \frac{L}{(t+1)^{\Gamma}} dt \) and \( \frac{1}{(t+1)^{\Gamma}} \to 0 \) as \( t \to \infty \). Therefore, \( V(t) \) is upper bounded by a sequence that is convergent. Hence, \( V(t) \) is convergent, thus giving us our result. \( \square \)
A.3 Proof of Theorem 1

Proof of Theorem 1. To show that the optimal cost will achieve a bound, we need only to show that the first derivative of the optimal cost is negative semi-definite and bounded. This idea stems from the principles of Lyapunov which we discuss below.

Lyapunov Principles The basic idea is to demonstrate stability of the system described by the PDE in the sense of Lyapunov.

Definition 1 (Definition of stability in the Lyapunov sense ([2])). Let \( V(x, t) \) be a non-negative function with derivative \( \dot{V}(x, t) \) along the system. The following is then true:

1. If \( V(x, t) \) is locally positive definite and \( \dot{V}(x, t) \leq 0 \) locally in \( x \) and for all \( t \), then the equilibrium point is locally stable (in the sense of Lyapunov).

2. If \( V(x, t) \) is locally positive definite and decreasent and \( \dot{V}(x, t) \leq 0 \) locally in \( x \) and for all \( t \), then the equilibrium point is uniformly locally stable (in the sense of Lyapunov).

3. If \( V(x, t) \) is locally positive definite and decreasent and \( \dot{V}(x, t) < a \) in \( x \) and for all \( t \), then the equilibrium point is Lyapunov stable, and the equilibrium point is ultimately bounded. Refer to Definition [2]

4. If \( V(x, t) \) is locally positive definite and decreasent and \( \dot{V}(x, t) < 0 \) locally in \( x \) and for all \( t \), then the equilibrium point is locally asymptotically stable.

5. If \( V(x, t) \) is locally positive definite and decreasent and \( \dot{V}(x, t) < 0 \) in \( x \) and for all \( t \), then the equilibrium point is globally asymptotically stable.

In our analysis we seek to determine the behavior of optimal cost with respect to change in the parameters (controllable by choice of the parameter update) and change in the data (uncontrollable). We assume that the change due the data and parameter update are both bounded and conclude that \( V \) is ultimately bounded and stable in the sense of Lyapunov. The idea of ultimately bounded is described by the next definition.

Definition 2. The solution of a differential equation \( \dot{x} = f(t, x) \) is uniformly ultimately bounded with ultimate bound \( b \) if \( b \) and \( c \) and for every \( 0 < \alpha < \gamma \), \( \exists T = T(a, b) \geq 0 \) such that \( \|x(t)\| \leq a \implies \|x(t)\| \leq b, \forall t \geq t_0 + T \).

The full proof is as follows. Let the Lyapunov function be given as

\[
V^*(t, y(t; \hat{\theta}(t))) = \min_{\hat{\theta}(\gamma) \in \Theta} \int_{t=\gamma}^{\infty} \gamma(\tau) J(\tau; y(\tau, \hat{\theta}(\tau))) d\tau. \tag{14}
\]

The first step is to observe whether the optimal cost is a suitable function to summarize the state of the learning at any time \( t \). In the limit \( \Gamma \to \infty \), the integral is indefinite and therefore represents a family of non-negative functions, parameterized by \( \hat{\theta}(\gamma) \) and \( x \). The non-negative nature of the function is guaranteed by the construction of the cost and the choice of the cost. Furthermore, the function is zero when both \( \hat{\theta}(\gamma) \) and \( x \) are zero. The function is continuously differentiable by construction. Lemma 1 describes boundedness and existence of the convergence point for all \( t \). We will assume that there exists a bounded compact set (search space) for \( \hat{\theta}(\gamma) \) (the weight initialization procedure ensures this.) and that input \( x \) is always numerically bounded (this can be achieved through data normalization methods). With all these conditions being true, we can observe that Eq. [14] is a reasonable candidate for this analysis [2].

Note that we can write the first derivative of the cost function (this was derived as part of the HJB derivation) as

\[
\frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t} = -\left[ \min_{\hat{\theta}(\gamma) \in \Theta} \left[ \gamma(t) J(t, y(t; \hat{\theta}(t))) \right] \right. \\
+ \left. \left( \frac{\delta \dot{V}^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right)^T \Delta \hat{\theta}(t) \right] + \epsilon_x. \tag{15}
\]
Consider the update as $\Delta \hat{\theta}(t) = \delta V^*(t, y(t; \hat{\theta}(t))) / \delta \hat{\theta}(t)$, and rewrite the preceding equation as

$$\frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} = - \left[ \min_{\theta(t) \in \Omega} \left[ \gamma(t) J(t, y(t; \hat{\theta}(t))) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right)^T \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right] + \epsilon_x \right].$$

(16)

The first term $\gamma(t) J(t, y(t; \hat{\theta}(t)))$ is non-negative and bounded by construction of the cost function. The second term

$$\left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right)^T \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \geq 0$$

is non-negative because it is a quadratic form. Next, note that $J(\tau, y(\tau; \hat{\theta}(\tau)))$ is bounded by construction.

Therefore,

$$\gamma(t) J(t, y(t; \hat{\theta}(t))) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right)^T \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \geq 0.$$

Because of the properties of the minimizer, we get

$$\min_{\theta(t) \in \Omega} \left[ \gamma(t) J(t, y(t; \hat{\theta}(t))) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right)^T \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right] \geq 0.$$

Further, $0 \leq \epsilon_x \leq \epsilon_B$ is by assumption. Finally,

$$0 \leq \min_{\theta(t) \in \Omega} \left[ \gamma(t) J(t, y(t; \hat{\theta}(t))) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right)^T \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right] + \epsilon_x \leq \epsilon_B.$$

(17)

Therefore, $0 \geq \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t} \geq -\epsilon_B$. As a consequence, $V^*(t, y(t; \hat{\theta}(t)))$ is ultimately bounded

(12).

The bounded of the first two terms can be ensured by the construction of cost function. However, the bound of $\epsilon_x$ comes from the data. Therefore, the theorem above only stands, if the data is numerically bounded and the change due to the data-distribution is bounded. Numerical boundedness of the data can be ensured through normalization procedures. However, there is no control over the change in the data distribution.

### A.4 Derivation of the Update through Finite Approximation

From Theorem 1, the update for the network is chosen as the derivative of the optimal cost, that is, $\frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)}$. However, the optimal cost is unknown and the derivative cannot be described directly.

Therefore, we need to approximate the optimal cost. The optimal cost $V^*(t, y(t; \hat{\theta}(t)))$ is described by two terms, the optimal cost at the current task and the optimal cost for the future. Let us describe the optimal cost for the current task at each $t$ to be $J^*(t, y(t; \hat{\theta}(t)))$. As a consequence of the HJB derivation, we can write

$$V^*(t, y(t; \hat{\theta}(t))) = J^*(\tau, y(t; \hat{\theta}(t))) + V^*(t + \Delta t, y(t + \Delta t; \hat{\theta}(t + \Delta t))).$$

(18)

Now, we approximate the future cost using the information provided at the current stage. To do so, we further simplify this framework. If we expand the second term through the first-order Taylor series expansion, we can write

$$V^*(t + \Delta t, y(t + \Delta t; \hat{\theta}(t + \Delta t))) = V^*(t, y(t; \hat{\theta}(t))) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t} \right)^T \Delta t + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \hat{\theta}(t)} \right)^T \Delta \hat{\theta}(t) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta x(t)} \right)^T \Delta x(t)$$

(19),
which when substituted to the original optimal cost gives the following equation:

\[
V^*(t, y(t)) = J^*(\tau, y(t; \hat{\theta}(t))) + V^*(t, y(t; \hat{\theta}(t))) + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \theta(t)} \right)^T \Delta \theta
+ \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta x(t)} \right)^T \Delta x(t)
\]

(20)

Let \( \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta t} \right) \Delta t + \left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta x(t)} \right)^T \Delta x(t) \) be a small bounded value (bounded disturbances and bounded change in the data). Let this bound be given by \( V_B \). We then obtain

\[
V^*(t, y(t)) = J^*(\tau, y(t; \hat{\theta}(t))) + \underbrace{V^*(t, y(t; \hat{\theta}(t)))}_{\text{Current task}} + \underbrace{\left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta \theta(t)} \right)^T \Delta \theta}_{\text{Cost of All the previous tasks}} + \underbrace{\left( \frac{\delta V^*(t, y(t; \hat{\theta}(t)))}{\delta x(t)} \right)^T \Delta x(t)}_{\text{Cost of Catastrophic Forgetting (cost of change)}} + V_B.
\]

(21)

Since, the last term in Eq. (21) is not a function of \( t \), we will ignore it from hereon. We discretize the overall setup for use in practical scenarios. Let \( k \) be the discrete sampling instant such that \( t = k dt \), and define \( \beta = \frac{1}{\Delta t} \) and with a little abuse of notation \( \gamma(k) = \gamma(k) + \beta \). Performing Euler’s discretization [13], we then get

\[
V^*(t, y(t)) = \underbrace{\gamma(k) J^*(k)}_{\text{Current task cost}} + \underbrace{\gamma(k) V^*(k)}_{\text{Cost of All the previous tasks}} + \underbrace{\beta [V^*(k+1) - V^*(k)]}_{\text{Cost of Catastrophic Forgetting (cost of change)}},
\]

(22)

Note that, for simplicity, we have dropped the notations for \( y(t; \hat{\theta}(t)) \). Next, we will replace all the optimal cost by their empirical counterpart and the derivative of each term on the right-hand side provides the final update as

\[
\hat{\theta}(k+1) = \hat{\theta}(k) - \alpha \left[ \gamma(k) J(k) + \gamma(k) V^*(k) + \beta [V^*(k+1) - V^*(k)] \right]
\]

(23)

where \( \alpha \) is the learning rate.

B Results – Implementation Details

B.1 Dataset Details

**Incremental Sine Waves** (Regression Problem): An incremental sine wave problem is defined by fifty (randomly generated) sine functions where each sine wave is considered a task and is incrementally shown to the model. Each sine function is generated by randomly selecting an amplitude in the range \([0.1, 5]\) and phase in \([0, \pi]\). For training, we generate 40 minibatches from the first sine function in the sequence (each minibatch has eight elements) and then 40 from the second and so on. We use a single regression head to predict these tasks. For the time, \( t \in \{0, 0.001, \cdots, 0.01\} \).

**Split-Omniglot Dataset**: We choose the first fifty classes to constitute our problem. Each character has 20 handwritten images. The dataset is divided into two parts. These classes are shown incrementally to all the approaches. We choose 12 images to be part of the training data for each task, 3 images for validation, and 5 images for the testing.
B.2 Algorithm DPMCL

Algorithm 1: Alternative optimization algorithm for optimal continual learning.

1. Initialize $\theta_1(t), \theta_2(t)$.
2. Choose $\epsilon \in \mathbb{R}$.
3. Initialize $D(k)$ as empty arrays.
4. while $k < k_T$ do
   5. Update $D(k)$ with the current task data $D^0(k)$.
   6. while each sample in $D^0(k)$. do
      7. while $i < \kappa$ do
         8. Generalization to New Task step
            Calculate Cost $J(k)$
            $\theta_2(k+1) \leftarrow$ Update $\theta_2(k)$.
            $\theta_1(k+1) \leftarrow$ Update $\theta_1(k)$.
      9. Compensation for Catastrophic Forgetting step
         Extract a sample from $D(k)$.
         Copy $\theta_2$ into $\theta_B$
         while $i < \zeta$ do
            10. Calculate cost with $\theta_B$ and $\theta_1$
                $\theta_B \leftarrow$ Update $\theta_B$.
         end
         Calculate Cost as
         $\gamma V^*(k; \theta_1(k), \theta_2(k)) + \beta * (V^*(\theta_1(k), \theta_B(k)) - V^*(k; \theta_1(k), \theta_2(k)))$
         $\theta_1(k+1) \leftarrow$ Update $\theta_1(k)$.
         $\theta_2(k+1) \leftarrow$ Update $\theta_2(k)$.
      end
   end
end

|                | Sine   | Omniglot |
|----------------|--------|----------|
| Learning rate  | 1e-03  | 1e-03    |
| num layers     | 2 three-layer models | 2 three-layer models |
| activation function | relu, output-linear | relu, output-softmax |
| num tasks      | 50     | 50       |
| num runs       | 100    | 100      |
| (N, in, H, out) | (8, 3, 100, 100) | (4, 784, 100, 50) |
| optimizer      | Adam   | Adam     |
| batch size     | 8      | 8        |
| $\beta$        | 0.1    | 0.1      |
| $\gamma'$      | 1      | 1        |
| $\zeta$        | 5      | 5        |
| $\kappa$       | 10     | 10       |

Table 3: DPMCL Model Parameters.

B.3 Algorithms, Comparison Methods.

Five algorithms were used for comparison. Naive For the naive implementation, we use the training data for each task to train our approach. The core idea behind such methodologies is to greedily learn
on any new task. We run gradient updates for each task data for a predetermined number of epochs,

Algorithm 2: Naive algorithm.

```
while i < iterations do
    Set random seed with the value i
    Initialize θ(t).
    while j < num tasks do
        Initialize task data T_j
        while k < num epochs do
            Sample a batch from the task data. θ(k + 1) <- Update θ(k).
        end
    end
end
Delete the model
```

Table 4: Naive, Experience Replay Model Parameters.

|                  | Sine            | Omniglot       |
|------------------|-----------------|----------------|
| Learning rate    | 1e-03           | 1e-03          |
| num epoch        | 20              | 20             |
| num layers       | 6               | 6              |
| activation function | relu, output-linear | relu, output-softmax |
| num tasks        | 50              | 50             |
| num runs (N, in, H, out) | 100            | 1000           |
| optimizer        | Adam            | Adam           |
| batch size       | 8               | 8              |
| Loss function    | Cross entropy   | MSE            |

Experience-Replay (ER [14]): This approach aims at maintaining the performance of all the tasks till now. We therefore define a task memory array. We store samples from each new task into the experience replay array. At the onslaught of every new task, we use the samples from the task memory array for training the network multiple epochs through the data. This method focuses on minimizing catastrophic forgetting.

Algorithm 3: Experience Replay Algorithm.

```
while i < iterations do
    Set random seed with the value i
    Initialize θ(t).
    Initialize experience replay D.
    while j < num tasks do
        Initialize task data T_j. Append task data to D
        while k < num epochs do
            Sample a batch from the D. θ(k + 1) <- Update θ(k).
        end
    end
end
Delete the model
```

Online Meta Learning (OML, [8]): We follow the meta training testing procedures described in [8] for this implementation. The process is composed of two loops. In the inner loop, the training is performed on the new task; in the outer loop, the training is performed on the buffer (task memory). We first save samples from each task into the buffer data. Both the in-
ner loop and the outer loop updates are performed by using the gradients of the cost function.

**Algorithm 4: Our OML Implementation.**

```plaintext
while i < iterations do
    Set random seed with the value i
    Initialize $\theta(i)$.
    Initialize experience replay $D$.
    while $j < num\ tasks$ do
        Initialize task data $T_j$. Append task data to $D$.
        while $K$ samples in $T_j$ do
            $\theta(k + 1) \leftarrow$ Update $\theta(k)$.
        end
        while $K$ samples in $D$ do
            $\tilde{\theta} \leftarrow$ Update $\theta(k)$.
            $\theta(k + 1) \leftarrow$ Update $\theta(k)$ using cost calculated with $\tilde{\theta}$.
        end
        Delete the model
    end
end
```

|                                | Sine       | Omniglot  |
|--------------------------------|------------|-----------|
| Learning rate                  | 1e-03      | 1e-03     |
| num epoch                      | 20         | 20        |
| num layers                     | 6          | 6         |
| activation function            | relu, output-linear | relu, output-softmax |
| num tasks                      | 50         | 50        |
| num runs                       | 100        | 100       |
| (N, in, H, out)                | (8, 3, 100, 100) | (4, 784, 100, 50) |
| optimizer                      | Adam       | Adam      |
| batch size                     | 8          | 8         |
| K                              | 100        | 100       |

Table 5: OML Model Parameters.

**Online Meta-Continual Learning (CML, [9]):** The learning process of this method is the same as that of the one in [8] with the key difference being the use of the representation network. The algorithm is provided in [9]. This approach is composed of a prelearned representation. We do not train a representation but try to learn it while the tasks are being observed sequentially. This protocol is followed to highlight the idea that although good representations are necessary, no data is available for training a representation.

**Neuromodulated Meta Learning (ANML, [3]):** Similar to the CML case, the learning process is the same as that of OML with the key difference being the neuromodulatory network. The algorithm
is provided in [3]. Similar to the earlier scenario, a representation is learned while the tasks are being observed.

**Algorithm 5:** Our CML, ANML Implementation.

1. while $i < \text{iterations}$ do
2.     Set random seed with the value $i$
3.     Initialize $\theta_1(t)$ and $\theta_2(t)$.
4.     Initialize experience replay $D$.
5.     while $j < \text{num tasks}$ do
6.         Initialize task data $T_j$
7.         Append task data to $D$
8.     while $K$ samples in $T_j$ do
9.         $\theta_1(k+1) \leftarrow \text{Update } \theta_1(k)$.
10.        $\theta_2(k+1) \leftarrow \text{Update } \theta_2(k)$.
11.     end
12.     while $K$ samples in $D$ do
13.         $\hat{\theta}_2 \leftarrow \text{Update } \theta_2$.
14.        $\hat{\theta}_2 \leftarrow \text{Update } \theta_2$ using cost calculated with $\hat{\theta}_2$
15.     end
16. end
17. Delete the model
18. end

| Learning rate | Sine | Omniglot |
|---------------|------|----------|
| num epoch     | 20   | 20       |
| num layers    | 2 three-layer | 2 three-layer |
| activation function | relu, output-linear | relu, output-softmax |
| num tasks     | 50   | 50       |
| num runs      | 100  | 100      |
| (N, in, H, out) | (8, 3, 100, 100) | (4, 784, 100, 50) |
| optimizer     | Adam | Adam     |
| batch size    | 8    | 8        |
| K             | 100  | 100      |

Table 6: CML and ANML Model Parameters.

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In Proceedings of the European Conference on Computer Vision (ECCV), pages 139–154, 2018.

[2] Chirayu D Athalye. Necessary condition on lyapunov functions corresponding to the globally asymptotically stable equilibrium point. In 2015 54th IEEE Conference on Decision and Control (CDC), pages 1168–1173. IEEE, 2015.

[3] Shawn Beaulieu, Lapo Frati, Thomas Miconi, Joel Lehman, Kenneth O Stanley, Jeff Clune, and Nick Cheney. Learning to continually learn. arXiv preprint arXiv:2002.09571, 2020.

[4] Massimo Caccia, Pau Rodríguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Caccia, Issam H. Laradji, Irina Rish, Alexandre Lacoste, David Vázquez, and Laurent Charlin. Online fast adaptation and knowledge accumulation: a new approach to continual learning. ArXiv, abs/2003.05856, 2020.

[5] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc’Aurelio Ranzato. Continual learning with tiny episodic memories. arXiv preprint arXiv:1902.10486, 2019.
[6] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In Advances in neural information processing systems, pages 2933–2941, 2014.

[7] Chelsea Finn, Pieter Abbeel, and Sergey 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.

[8] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. arXiv preprint arXiv:1902.08438, 2019.

[9] Khurram Javed and Martha White. Meta-learning representations for continual learning. In Advances in Neural Information Processing Systems, pages 1818–1828, 2019.

[10] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences, 114(13):3521–3526, 2017.

[11] 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.

[12] Frank L Lewis, Draguna Vrabie, and Vassilis L Syrmos. Optimal control. John Wiley & Sons, 2012.

[13] FW Lewis, Suresh Jagannathan, and A Yesildirak. Neural network control of robot manipulators and non-linear systems. CRC Press, 1998.

[14] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning, 8(3-4):293–321, 1992.

[15] David Lopez-Paz and Marc’Aurelio Ranzato. Gradient episodic memory for continual learning. In Advances in Neural Information Processing Systems, pages 6467–6476, 2017.

[16] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. arXiv preprint arXiv:1606.04671, 2016.

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

[18] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. arXiv preprint arXiv:1708.01547, 2017.

[19] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 3987–3995. JMLR. org, 2017.