

# SUPPLEMENTARY MATERIAL: 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.

## A APPENDIX -DERIVATION AND PROOFS

First we will derive our PDE. Additional details about the derivation can be found in Lewis et al. (2012).

### A.1 DERIVATION OF HAMILTON-JACOBI BELLMAN FOR THE META CONTINUAL LEARNING SETTING

Let the optimal cost be given as

$$V^*(t; \hat{\theta}(t)) = \min_{\hat{\theta}(\tau) \in \Omega: t \leq \tau \leq \Gamma} \left[ \int_{\tau=t}^{\Gamma} J(\tau; \hat{\theta}(\tau)) d\tau \right]. \quad (1)$$

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

$$V^*(t; \hat{\theta}(t)) = \min_{\hat{\theta}(\tau) \in \Omega: t \leq \tau \leq \Gamma} \left[ \int_{\tau=t}^{t+\Delta t} J(\tau; \hat{\theta}(\tau)) d\tau + \int_{\tau=t+\Delta t}^{\Gamma} J(\tau; \hat{\theta}(\tau)) d\tau \right]. \quad (2)$$

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

$$V^*(t; \hat{\theta}(t)) = \min_{\hat{\theta}(\tau) \in \Omega: t \leq \tau \leq \Gamma} \left[ \int_{\tau=t}^{t+\Delta t} J(\tau; \hat{\theta}(\tau)) d\tau + V(t + \Delta t; \hat{\theta}(t + \Delta t)) \right]. \quad (3)$$

Suppose now that all information for  $\tau \geq t + \Delta t$  is known, and also suppose that all optimal configurations of parameters are known. With this information, we can narrow our search to find

just the optimal costs for the interval  $[t, t + \Delta t]$  and assume all the other costs are optimal.

$$V^*(t; \hat{\theta}(t)) = \min_{\hat{\theta}(\tau) \in \Omega: t \leq \tau \leq t + \Delta t} \left[ \int_{\tau=t}^{t+\Delta t} J(\tau; \hat{\theta}(\tau)) d\tau + V^*(t + \Delta t; \hat{\theta}(t + \Delta t)) \right]. \quad (4)$$

Now, the only parameters to be obtained are for the sequence  $t \leq \tau \leq t + \Delta t$ . We must approximate the  $V^*(t + \Delta t; \hat{\theta}(t + \Delta t))$  using the information provided in the interval  $[t, t + \Delta t]$ . To do so, we further simplify this framework by writing the first-order Taylor series expansion. However,  $V^*(t + \Delta t; \hat{\theta}(t + \Delta t))$  is a function of  $\mathbf{y}(t)$ , that is the model. Since  $\mathbf{y}(t)$  is a function of  $(t, \mathbf{x}(t), \hat{\theta}(t))$ , all changes in  $\mathbf{y}(t)$ , can be summarized through  $(t, \mathbf{x}(t), \hat{\theta}(t))$ . Therefore, we evaluate the Taylor series around  $(t, \mathbf{x}(t), \hat{\theta}(t))$ ,

$$V^*(t + \Delta t; \hat{\theta}(t + \Delta t)) = V^*(t; \hat{\theta}(t)) + (V_t^*)^T \Delta t + (V_{\hat{\theta}(t)}^*)^T \Delta \hat{\theta} + (V_{\mathbf{x}(t)}^*)^T \Delta \mathbf{x}(t), \quad (5)$$

where we use the notation  $V_{(\cdot)}^*$  to denote the partial derivative with respect to  $(\cdot)$ . For instance,  $V_{\hat{\theta}(t)}^* = \frac{\partial V^*(t; \hat{\theta}(t))}{\partial \hat{\theta}(t)}$ . Substituting into the original equation, we have

$$V^*(t; \hat{\theta}(t)) = \min_{\hat{\theta}(\tau) \in \Omega: t \leq \tau \leq t + \Delta t} \left[ \int_{\tau=t}^{t+\Delta t} J(\tau; \hat{\theta}(\tau)) d\tau + V^*(t; \hat{\theta}(t)) + (V_t^*)^T \Delta t + (V_{\hat{\theta}(t)}^*)^T \Delta \hat{\theta} + (V_{\mathbf{x}(t)}^*)^T \Delta \mathbf{x}(t) \right]. \quad (6)$$

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

$$V^*(t; \hat{\theta}(t)) = \min_{\hat{\theta}(\tau) \in \Omega: t \leq \tau \leq t + \Delta t} \left[ \int_{\tau=t}^{t+\Delta t} J(\tau; \hat{\theta}(\tau)) d\tau + (V_{\hat{\theta}(t)}^*)^T \Delta \hat{\theta} + (V_{\mathbf{x}(t)}^*)^T \Delta \mathbf{x}(t) \right] + V^*(t; \hat{\theta}(t)) + (V_t^*)^T \Delta t. \quad (7)$$

Upon cancellation of common terms we have

$$-(V_t^*)^T \Delta t = \min_{\hat{\theta}(\tau) \in \Omega: t \leq \tau \leq t + \Delta t} \left[ \int_{\tau=t}^{t+\Delta t} J(\tau; \hat{\theta}(\tau)) d\tau + (V_{\hat{\theta}(t)}^*)^T \Delta \hat{\theta} + (V_{\mathbf{x}(t)}^*)^T \Delta \mathbf{x}(t) \right]. \quad (8)$$

Observe that  $\int_{\tau=t}^{t+\Delta t} J(\tau; \hat{\theta}(\tau)) d\tau = J(t; \hat{\theta}(t)) + \int_{\tau=t+}^{t+\Delta t} \gamma(\tau) \ell(\tau) d\tau$ , where  $J(t; \hat{\theta}(t))$  represents all the previous tasks and  $\int_{\tau=t+}^{t+\Delta t} \gamma(\tau) \ell(\tau) d\tau$  represents the new task. Therefore, we get our final PDE as

$$-(V_t^*)^T \Delta t = \min_{\hat{\theta}(\tau) \in \Omega: t \leq \tau \leq t + \Delta t} \left[ J(\tau; \hat{\theta}(\tau)) + \int_{\tau=t+}^{t+\Delta t} \gamma(\tau) \ell(\tau) d\tau + (V_{\hat{\theta}(t)}^*)^T \Delta \hat{\theta} + (V_{\mathbf{x}(t)}^*)^T \Delta \mathbf{x}(t) \right]. \quad (9)$$

Let us push  $\Delta t \rightarrow 0$  and denote  $\lim_{\Delta t \rightarrow 0} \int_{\tau=t+}^{t+\Delta t} \gamma(\tau) \ell(\tau) d\tau$  as  $J_N(\tau; \hat{\theta}(\tau))$ . We get

$$\begin{aligned} -(V_t^*)^T \Delta t = \min_{\hat{\theta}(t) \in \Omega} & \left[ J(t; \hat{\theta}(t)) + J_N(t; \hat{\theta}(t)) \right. \\ & \left. + (V_{\hat{\theta}(t)}^*)^T \Delta \hat{\theta} + (V_{x(t)}^*)^T \Delta x(t) \right]. \end{aligned} \quad (10)$$

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

## A.2 PROOF OF LEMMA AND COROLLARY 1

**Lemma 1.** Let  $J(t; \hat{\theta}(t)) = \int_{\tau=0}^t \gamma(\tau) \ell(\tau) d\tau$ , and let  $\ell$  be continuous and  $L \geq \ell(\tau) \geq \epsilon, \forall \tau, \epsilon > 0$ . Let  $\gamma(t) = 1$ , then  $J(t; \hat{\theta}(t))$  is divergent. *Proof:* See Appendix A.2.

*Proof of Lemma 1.* consider  $J(t; \hat{\theta}(t)) = \int_{\tau=0}^t \gamma(\tau) \ell(\tau) d\tau$  and we want to show that  $\lim_{t \rightarrow \infty} \int_{\tau=0}^t \gamma(\tau) \ell(\tau) d\tau$  diverges. We may write  $\lim_{t \rightarrow \infty} \int_{t=0}^{\infty} \epsilon dt$ , where  $\lim_{t \rightarrow \infty} \int_{t=0}^{\infty} \epsilon dt = \lim_{t \rightarrow \infty} [\epsilon t]_0^{\Gamma} = \infty$ . Therefore, the limit is divergent.  $\square$

**Corollary 1.** Consider  $t \in [0, \Gamma] : \Gamma \rightarrow \infty$  and  $J(t; \hat{\theta}(t)) = \int_{\tau=0}^t \gamma(\tau) \ell(\tau) d\tau$ , and let  $\epsilon \leq \ell(\tau) \leq L, \forall \epsilon > 0$  and let  $\ell$  be a continuous and integrable  $\forall \tau \in [0, t]$ . Choose  $\gamma$  such that  $\gamma(t) \rightarrow 1, t \rightarrow \infty$  such that  $\gamma(\tau) \leq 1, \forall \tau \in [0, t]$ . Under these assumptions,  $J(t; \hat{\theta}(t))$  is convergent if  $\int_{\tau=0}^t \gamma(\tau) d\tau \leq M$ . *Proof:* See Appendix A.2.

*Proof of Corollary 1.* Consider  $J(t; \hat{\theta}(t)) = \int_{\tau=0}^t \gamma(\tau) \ell(\tau) d\tau$  and we want to show that  $\lim_{t \rightarrow \infty} \int_{\tau=0}^t \gamma(\tau) \ell(\tau) d\tau$  exists. Since,  $\ell(\tau) \leq L$ , we may write  $\lim_{t \rightarrow \infty} \int_{\tau=0}^t \gamma(\tau) \ell(\tau) d\tau \leq \lim_{t \rightarrow \infty} \int_{\tau=0}^t \gamma(\tau) L d\tau \leq L \lim_{t \rightarrow \infty} \int_{\tau=0}^t \gamma(\tau) d\tau$ . As,  $\gamma(\tau)$  is a monotonic function which converges point-wise to 1, we may use the monotone convergence theorem to write  $L \lim_{t \rightarrow \infty} \int_{\tau=0}^t \gamma(\tau) d\tau \leq \int_{\tau=0}^t \gamma(\tau) d\tau$ . Next, if  $\int_{\tau=0}^t \gamma(\tau) d\tau \leq M$ , then  $L \int_{\tau=0}^t \gamma(\tau) d\tau$  is convergent. Therefore,  $J(t; \hat{\theta}(t))$  is upper bounded by a sequence that is convergent and  $J(t; \hat{\theta}(t))$  is convergent.  $\square$

## A.3 THEOREM 1

**Theorem 1.** Let the cumulative cost be bounded (according to Lemma 1) and defined in (??) with its time derivative is defined in (??). Let  $\Omega$  be a compact set with  $\hat{\theta}(t) \in \Omega$ . Consider the assumptions  $\|J_{\hat{\theta}(t)}\| > 0, \|J_x\| \|\Delta x(t)\| < 1, \|J_x\| > 0$  and  $\frac{\partial}{\partial(\hat{\theta}(t))} \int_{\tau=t}^c J(\tau, \hat{\theta}(\tau)) d\tau = J(t, \hat{\theta}(t)), c \in [t, t + \Delta t], \forall t \in [0, \Gamma]$  Consider the gradient update given as  $\alpha(t) V_{\hat{\theta}(t)}$  with  $\alpha(t) > 0$  being the learning rate. Choose  $\alpha(t) = \frac{1 - \|J_x\| \|\Delta x(t)\|}{\beta \|J_{\hat{\theta}(t)}\|}$ . The first derivative of  $V(t; \hat{\theta}(t))$  is less than equal to zero and  $V(t; \hat{\theta}(t))$  is ultimately bounded with the bound on  $J(t; \hat{\theta}(t))$  given as  $J(t; \hat{\theta}(t)) \leq \beta$ , where  $\beta$  is a user defined threshold on the cost.

*Proof of Theorem 1.* To show that the cumulative cost will achieve a bound, we need only to show that the first derivative of the cumulative 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 ((Lewis et al., 2012))). 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 decreseant 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 decreseant 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 decreseant 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 decreseant 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 cumulative cost with respect to change in the parameters (controllable by choice of the parameter update) and change in the input (uncontrollable). We assume that the changes due to input and parameter update are both bounded, and we 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  $0 < a < c$ ,  $\exists T = T(a, b) \geq 0$  such that  $\|x(t_0)\| \leq a \implies \|x(t)\| \leq b, \forall t \geq t_0 + T$ .

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

$$V(t; \hat{\theta}(t)) = \int_{\tau=t}^{\Gamma} J(\tau, \hat{\theta}(\tau)) d\tau. \quad (11)$$

The first step is to observe that cumulative cost is a suitable function to summarize the state of the learning at any time  $t$ . The integral is indefinite and therefore represents a family of non-negative functions, parameterized by  $\hat{\theta}(\tau)$  and  $x(t)$ . 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  $\hat{\theta}(\tau)$  and  $x(t)$  are zero. The function is continuously differentiable by construction. Lemma 1 describes boundedness and existence of the convergence point for  $J(\tau, \hat{\theta}(\tau))$  for all  $\tau \in [t, \Gamma]$ . We will also assume that there exists a bounded compact set (search space) for  $\hat{\theta}(\tau)$  (the weight initialization procedure ensures this) and that input  $x(t)$  is always numerically bounded (this can be achieved through data normalization methods). With all these conditions being true, we can observe that Eq. 11 is a reasonable candidate for this analysis (Athalye, 2015) and also explains the behavior of the system completely.

Note that we can write the first derivative of the cost function (*this was derived as part of the HJB derivation*) under the assumption that the boundary condition for the optimal cost  $V^*$  is  $V$  such that

$$\begin{aligned} \frac{\partial V(t; \hat{\theta}(t))}{\partial t} = & - \left[ J(t; \hat{\theta}(t)) + \left( V_{\hat{\theta}(t)} \right)^T \Delta \hat{\theta}(t) \right. \\ & \left. + \left( V_x \right)^T \Delta x(t) \right]. \end{aligned} \quad (12)$$

Consider the update as  $\Delta \hat{\theta}(t) = \alpha V_{\hat{\theta}(t)}$ , with  $\alpha > 0$ , and write by the fundamental theorem of calculus and chain rule

$$\begin{aligned}
V_{\hat{\theta}(t)} &= \frac{\partial}{\partial \hat{\theta}(t)} \int_{\tau=t}^{\Gamma} J(\tau, \hat{\theta}(\tau)) d\tau \\
&= -\frac{\partial}{\partial \hat{\theta}(t)} \int_{\tau=\Gamma}^t J(\tau, \mathbf{y}(\tau; \hat{\theta}(\tau))) d\tau \\
&= -\frac{\partial}{\partial \hat{\theta}(t)} \left[ \int_{\tau=\Gamma}^c J(\tau, \mathbf{y}(\tau; \hat{\theta}(\tau))) d\tau \right. \\
&\quad \left. + \int_{\tau=c}^t J(\tau, \mathbf{y}(\tau; \hat{\theta}(\tau))) d\tau \right] \\
&= -J(t, \mathbf{y}(t; \hat{\theta}(t))) \frac{\partial J(t; \hat{\theta}(t))}{\partial \hat{\theta}(t)} \\
&= -J(t; \hat{\theta}(t)) J_{\hat{\theta}(t)}(t; \hat{\theta}(t)) \\
&= -J(t; \hat{\theta}(t)) J_{\hat{\theta}(t)}.
\end{aligned} \tag{13}$$

Similarly, simplify  $V_x$ , and write by the fundamental theorem of calculus and chain rule

$$\begin{aligned}
V_x &= \frac{\partial}{\partial x} \int_{\tau=t}^{\Gamma} J(\tau, \hat{\theta}(\tau)) d\tau \\
&= -\frac{\partial}{\partial x} \int_{\tau=\Gamma}^t J(\tau, \mathbf{y}(\tau; \hat{\theta}(\tau))) d\tau \\
&= \frac{\partial}{\partial x} \left[ \int_{\tau=\Gamma}^c J(\tau, \mathbf{y}(\tau; \hat{\theta}(\tau))) d\tau \right. \\
&\quad \left. + \int_{\tau=c}^t J(\tau, \mathbf{y}(\tau; \hat{\theta}(\tau))) d\tau \right] \\
&= -J(t, \mathbf{y}(t; \hat{\theta}(t))) \frac{\partial J(t; \hat{\theta}(t))}{\partial x} \\
&= -J(t; \hat{\theta}(t)) J_x(t; \hat{\theta}(t)) \\
&= -J(t; \hat{\theta}(t)) J_x.
\end{aligned} \tag{14}$$

Substituting Eqs. (13) and (14) into (12), we can write

$$\begin{aligned}
\frac{\partial V(t; \hat{\theta}(t))}{\partial t} &= - \left[ J(t; \hat{\theta}(t)) \right. \\
&\quad \left. - \left( J(t; \hat{\theta}(t)) J_{\hat{\theta}(t)} \right)^T (\alpha(t) J(t; \hat{\theta}(t)) J_{\hat{\theta}(t)}) \right. \\
&\quad \left. - \left( J(t; \hat{\theta}(t)) J_x \right)^T \Delta x(t) \right],
\end{aligned} \tag{15}$$

which when simplified provides

$$\begin{aligned}
\frac{\partial V(t; \hat{\theta}(t))}{\partial t} &= - \left[ J(t; \hat{\theta}(t)) - \alpha(t) J(t; \hat{\theta}(t))^2 \|J_{\hat{\theta}(t)}\|^2 \right. \\
&\quad \left. - J(t; \hat{\theta}(t)) \left( J_x \right)^T \Delta x(t) \right].
\end{aligned} \tag{16}$$

Pulling  $J(t; \hat{\theta}(t))$  out of the bracket provides

$$\begin{aligned}
\frac{\partial V(t; \hat{\theta}(t))}{\partial t} &= -J(t; \hat{\theta}(t)) \left[ 1 - \alpha(t) J(t; \hat{\theta}(t)) \|J_{\hat{\theta}(t)}\|^2 \right. \\
&\quad \left. - \left( J_x \right)^T \Delta x(t) \right].
\end{aligned} \tag{17}$$

The first term  $J(t; \hat{\theta}(t)) \geq 0$  and bounded by construction of the cost function. Equation (17) is negative as long as the terms in the bracket are greater than or equal to zero, which is possible if and only if

$$\left[ \alpha(t) J(t; \hat{\theta}(t)) \|J_{\hat{\theta}(t)}\|^2 + \left( J_x \right)^T \Delta x(t) \right] < 1. \quad (18)$$

Hence, by Cauchy's inequality,

$$\begin{aligned} \left[ \alpha J(t; \hat{\theta}(t)) \|J_{\hat{\theta}(t)}\|^2 + \|J_x\| \|\Delta x(t)\| \right] &< 1, \\ J(t; \hat{\theta}(t)) &< \frac{1 - \|J_x\| \|\Delta x(t)\|}{\alpha(t) \|J_{\hat{\theta}(t)}\|^2}. \end{aligned} \quad (19)$$

Choose  $\alpha(t) = \frac{1 - \|J_x\| \|\Delta x(t)\|}{\beta \|J_{\hat{\theta}(t)}\|^2}$  with  $\beta > 0$ , and get the bound on  $\|J(t; \hat{\theta}(t))\|$  as

$$J(t; \hat{\theta}(t)) < \beta. \quad (20)$$

As a consequence,  $V(t; \hat{\theta}(t))$  is ultimately bounded (Lewis et al., 2012) with  $\|J(t; \hat{\theta}(t))\| < \beta$ .  $\square$

#### A.4 DERIVATION OF THE UPDATE THROUGH FINITE APPROXIMATION

From Theorem 1, the update for the network is chosen as the derivative of the cumulative cost, that is,  $\frac{\partial V(t; \mathbf{y}(t; \hat{\theta}(t)))}{\partial \hat{\theta}(t)}$  providing

$$- \frac{\partial V^*(t; \hat{\theta}(t))}{\partial t} = \min_{\hat{\theta}(t) \in \Omega} \left[ H(t; \hat{\theta}(t)) \right], \quad (21)$$

where  $H(t; \hat{\theta}(t))$  is the CT Hamiltonian, which we will discretize and approximate. Under the assumption that the boundary condition for the optimal cost is the cumulative cost itself, we may write

$$\begin{aligned} H(t; \hat{\theta}(t)) &= J(t; \hat{\theta}(t)) \\ &+ (V_{\hat{\theta}(t)})^T \Delta \hat{\theta} + (V_{x(t)})^T \Delta x(t). \end{aligned} \quad (22)$$

Upon Euler's discretization, we achieve

$$\begin{aligned} \frac{1}{\Delta t} H(k; \hat{\theta}(k)) &= \frac{1}{\Delta t} J(k; \hat{\theta}(k)) \\ &+ (V(k; \hat{\theta}(k+1)) - V(k; \hat{\theta}(k))) \\ &+ (V(k+1; \hat{\theta}(k)) - V(k; \hat{\theta}(k))) \end{aligned} \quad (23)$$

Simplification provides

$$\begin{aligned} H(k; \hat{\theta}(k)) &= J(k; \hat{\theta}(k)) \\ &+ \Delta t (V(k; \hat{\theta}(k+1)) - V(k; \hat{\theta}(k))) \\ &+ \Delta t (V(k+1; \hat{\theta}(k)) - V^*(k; \hat{\theta}(k))) \end{aligned} \quad (24)$$

Taking the derivative and setting it to zero, we get

$$\begin{aligned} 0 &= \frac{\partial}{\partial \hat{\theta}(k)} J(k; \hat{\theta}(k)) \\ &+ \frac{\partial}{\partial \hat{\theta}(k)} \Delta t (V(k; \hat{\theta}(k+1)) - V(k; \hat{\theta}(k))) \\ &+ \frac{\partial}{\partial \hat{\theta}(k)} \Delta t (V(k+1; \hat{\theta}(k)) - V^*(k; \hat{\theta}(k))). \end{aligned} \quad (25)$$

Rearranging, we get

$$\begin{aligned} \frac{\partial V(k; \hat{\theta}(k))}{\partial \hat{\theta}(k)} &= \left[ \frac{\partial}{\partial \hat{\theta}(k)} \frac{J(k; \hat{\theta}(k))}{\Delta t} \right. \\ &+ \frac{\partial}{\partial \hat{\theta}(k)} (V(k; \hat{\theta}(k+1)) - V(k; \hat{\theta}(k))) \\ &\left. + \frac{\partial}{\partial \hat{\theta}(k)} V(k+1; \hat{\theta}(k)) \right]. \end{aligned} \quad (26)$$

Since we have no information about the data from the future, we will think of the last term  $\frac{\partial}{\partial \hat{\theta}(k)} (V(k+1; \hat{\theta}(k)))$  as 0. Under the assumption that  $\frac{\partial}{\partial \hat{\theta}(t)} \int_{\tau=t}^c J(\tau, \hat{\theta}(\tau)) d\tau \propto J(t, \hat{\theta}(t))$ . From the fundamental theorem of calculus we write

$$\begin{aligned} V_{\hat{\theta}(t)} &= \frac{\partial}{\partial \hat{\theta}(t)} \int_{\tau=t}^{\Gamma} J(\tau, \hat{\theta}(\tau)) d\tau \\ &= -\frac{\partial}{\partial \hat{\theta}(t)} \int_{\tau=\Gamma}^t J(\tau, \mathbf{y}(\tau; \hat{\theta}(\tau))) d\tau \\ &\propto -J(t; \hat{\theta}(t)) \frac{\partial J(t; \hat{\theta}(t))}{\partial \hat{\theta}(t)} \\ &\propto -J(t; \hat{\theta}(t)). \end{aligned} \quad (27)$$

We now achieve our update as

$$\begin{aligned} \frac{\partial V(k; \hat{\theta}(k))}{\partial \hat{\theta}(k)} &\propto \left[ \frac{\partial}{\partial \hat{\theta}(k)} \frac{J(k; \hat{\theta}(k))}{\Delta t} \right. \\ &\left. - \frac{\partial}{\partial \hat{\theta}(k)} (J(k; \hat{\theta}(k+1)) - J(k; \hat{\theta}(k))) \right]. \end{aligned} \quad (28)$$

To obtain the first term, we replace  $1/\Delta t$  with a small value  $\eta$  and write  $J(k; \hat{\theta}(k)) = J_N(k; \hat{\theta}(k)) + J_P(k; \hat{\theta}(k))$ , where  $J_P(k; \hat{\theta}(k))$  refers to the cost on all the previous tasks and  $J_N(k; \hat{\theta}(k))$  refers to the cost on the new task. For the second term, we replace the difference  $(J(k; \hat{\theta}(k+1)) - J(k; \hat{\theta}(k)))$  with  $(J(k; \hat{\theta}(k+\zeta)) - J(k; \hat{\theta}(k)))$  with  $\zeta$  being the finite number of steps for the approximation. We get the gradient

$$\begin{aligned} \frac{\partial V(k; \hat{\theta}(k))}{\partial \hat{\theta}(k)} &\propto \left[ \frac{\partial}{\partial \hat{\theta}(k)} \eta [J_N(k; \hat{\theta}(k)) + J_P(k; \hat{\theta}(k))] \right. \\ &\left. + \frac{\partial}{\partial \hat{\theta}(k)} (J(k; \hat{\theta}(k)) - J(k; \hat{\theta}(k+\zeta))) \right], \end{aligned} \quad (29)$$

where  $\zeta$  is a predefined number of iterations for the parameters. To simplify notation, we write  $J_N(k; \hat{\theta}(k))$  as  $J_N(k)$ ,  $J_P(k; \hat{\theta}(k))$  as  $J_P(k)$ ,  $J(k; \hat{\theta}(k))$  as  $J_{PN}(k)$ , and  $J(k; \hat{\theta}(k+\zeta))$  as  $J_{PN}(k; \hat{\theta}(k+\zeta))$  and

$$\begin{aligned} \frac{\partial V(k; \hat{\theta}(k))}{\partial \hat{\theta}(k)} &\propto \left[ \frac{\partial}{\partial \hat{\theta}(k)} \eta [J_N(k) + J_P(k)] \right. \\ &\left. + \frac{\partial}{\partial \hat{\theta}(k)} (J_{PN}(k) - J_{PN}(k; \hat{\theta}(k+\zeta))) \right]. \end{aligned} \quad (30)$$

Our update rule with  $\eta = 1$  then is

$$\begin{aligned} \hat{\theta}(k+1) &= \hat{\theta}(k) - \alpha(t) \times \left[ \frac{\partial}{\partial \hat{\theta}(k)} [J_N(k) + J_P(k)] \right. \\ &\left. + \frac{\partial}{\partial \hat{\theta}(k)} (J_{PN}(k) - J_{PN}(k; \hat{\theta}(k+\zeta))) \right]. \end{aligned} \quad (31)$$

## B APPENDIX - RESULTS AND IMPLEMENTATION DETAILS

Our implementations are done in Python with the Pytorch library (version 1.4). All the code including the data obtained from the our runs is provided in a separate zip file. First, we will discuss the datasets.

### B.1 DATA SET

*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,  $t \in \{0, 0.001, \dots, 0.01\}$ . 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 time  $t \in \{0, 0.001, \dots, 0.01\}$ . Each task is generated by making incremental changes to amplitude, phase, and frequency while following the protocol in (Finn et al., 2017).

*Split-Omniglot data set*: We choose the first fifty classes to constitute our problem. Each character has 20 handwritten images. The data set 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.

*MNIST CIFAR10 data set*: These data-sets are comprised of ten classes. We incrementally show each of these classes to our model. Therefore, each task in our problem is comprised of exactly one class. We choose 60 % of the data from each task to constitute our training data and the rest is split into validation and test.

### B.2 MODEL SETUP

*SINE*: DPMCL uses two neural networks: one for the representation and one for the prediction. Both have a three-layer feed-forward network (one input, one hidden, and one output layer) with 100 hidden layer neurons and *relu* activation function. The input vector is  $3 \times 1m$  and the output of the prediction network is a  $100 \times 1$  vector with a linear activation function at the output layer. For the implementations of Naive, ER, and FTML, a single six-layer network (100 hidden units, relu activation) is utilized. For OML, we use two networks: representation learning network and prediction learning networks. For ANML, the representation learning network becomes the neuromodulatory network (Beaulieu et al., 2020). For OML and ANML implementations, we use 100 hidden units (same as DPMCL).

#### B.2.1 SINE MODEL DEFINITIONS

```
# Model Feature extractions.
self.model_F = torch.nn.Sequential(
    torch.nn.Linear(self.config['D_in'], self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['D_in'])
)

# The g model and the buffer model are the same
# Model g
self.model_P = torch.nn.Sequential(
    torch.nn.Linear(self.config['D_in'], self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['D_out'])
)
```



```

)

# Model buffer
self.model_buffer = torch.nn.Sequential(
    torch.nn.Linear(self.config['D_in'], self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['D_out'])
)

```

*MNIST, OMNI, CIFAR10*: For all these data sets, we use a combination of convolutional neural network (two convolutional layers, max pooling, relu-activation function) and a feed-forward network (2 feed-forward layer, relu activation function, softmax output). For CIFAR10, we use three channels, of which only one channel is used with OMNI and MNIST. For the implementations of Naive, ER, and FTML, a single four-layer network (2 convolutional layers, 2 feed-forward layers) is utilized. For OML (Javed and White, 2019), we use two networks: a representation learning network (convolutional neural network) and a prediction learning network (feed-forward network), respectively. For ANML, the representation learning network becomes the neuromodulatory network (Beaulieu et al., 2020).

#### B.2.2 OMNI

```

# Feature Extractor
self.model_F = torch.nn.Sequential(
    torch.nn.Conv2d(1, 32, kernel_size=5, stride=1, padding=2),
    torch.nn.MaxPool2d(kernel_size=2, stride=2),
    torch.nn.ReLU(),
    torch.nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
    torch.nn.MaxPool2d(kernel_size=2, stride=2),
    torch.nn.ReLU(),
)

# Feedforward Layer
self.model_P = torch.nn.Sequential(
    torch.nn.Linear(7 * 7 * 64, self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['D_out'])
)

# Buffer Layer
self.model_buffer = torch.nn.Sequential(
    torch.nn.Linear(7 * 7 * 64, self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['D_out'])
)

```

#### B.2.3 CIFAR10 AND MNIST

```

# Feature Extractor
self.model_F = torch.nn.Sequential(
    torch.nn.Conv2d(3, 6, 5),
    torch.nn.MaxPool2d(kernel_size=2, stride=2),
    torch.nn.ReLU(),
    torch.nn.Conv2d(6, 16, 5),
    torch.nn.MaxPool2d(kernel_size=2, stride=2),
    torch.nn.ReLU(),
    torch.nn.Dropout()
)

```

```

# Feed Forward Layer
self.model_P = torch.nn.Sequential(
    torch.nn.Linear(256, self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['D_out']))
)
# Buffer Layer
self.model_buffer = torch.nn.Sequential(
    torch.nn.Linear(256, self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['H']),
    torch.nn.ReLU(),
    torch.nn.Linear(self.config['H'], self.config['D_out']))
)

```

### B.3 HYPERPARAMETERS

Since, DPMCL update rule involves division by the norm of the gradient, we use the Adagrad optimizer throughout.

Parameters	Sine	Omniglot	MNIST	CIFAR10
Learning rate	$1e-03$	$1e-04$	$1e-04$	$1e-04$
total runs	50	50	50	50
num tasks	50	50	10	10
Num of Hidden Layers.	100	100	512	512
Input <sub>size</sub>	3	$28 \times 28$	$28 \times 28$	$28 \times 28$
Output Size	100	50	10	10
$\rho$	300	200	300	600
$\zeta$	2	2	5	5
$N_{meta}$	150	100	150	300
$N_{grad}$	150	100	150	300
$N$	300	200	300	600
length of $D_P$	1000	20000	20000	100000
$\beta$	1000	10000	10000	10000
batch <sub>size</sub>	64	8	32	512
activation function	relu, output-linear	relu, output-softmax	relu, output-softmax	relu, output-softmax
optimizer	Adagrad	Adagrad	Adagrad	Adagrad
Loss function	MSE	Cross Entropy	Cross Entropy	Cross Entropy

Next, we will discuss the different methods that have been used in the study. We start by describing the algorithm for DPMCL.

### B.4 DPMCL

We define a new task sample,  $\mathcal{D}_N(k) = \{\mathcal{X}_k, \mathcal{Y}_k\}$ , and a task memory (samples from all the previous tasks)  $\mathcal{D}_P(k) \subset \cup_{\tau=0}^{k-1} \mathcal{T}_\tau$ . We can approximate the required terms in our update rule using samples (batches) from  $\mathcal{D}_P(k)$  and  $\mathcal{D}_N(k)$ . The overall algorithm consists of two steps: generalization and catastrophic forgetting (see Algorithm 1 in Appendix C). DPMCL comprises representation and prediction neural networks parameterized by  $\hat{\theta}_1$  and  $\hat{\theta}_2$ , respectively. For each batch  $b_N \in \mathcal{D}_N(k)$ , DPMCL alternatively performs generalization and catastrophic forgetting cost updates  $\rho$  times. The generalization cost update consists of computing the cost  $J_N$  and using that to update  $\hat{\theta}_1$

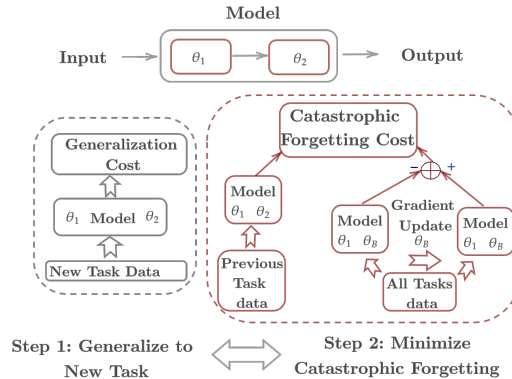


Figure 1: Illustration of DPMCL to learn parameters  $\hat{\theta}_1, \hat{\theta}_2$  (the copy of  $\hat{\theta}_2$  is  $\hat{\theta}_B$ ).

and  $\hat{\theta}_2$ ; 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  $b_P \in \mathcal{D}_P(k)$ . Second, to approximate the term  $(J_{PN}(k; \hat{\theta}(k+\zeta)))$ , we copy  $\hat{\theta}_2$  (prediction network) into a temporary network parameterized by  $\hat{\theta}_B$ . We then perform  $\zeta$  updates on  $\hat{\theta}_B$  while keeping  $\hat{\theta}_1$  fixed. Third, using  $\hat{\theta}_B(k+\zeta)$ , we compute  $J_{PN}(k; \hat{\theta}_B(k+\zeta))$  and update  $\hat{\theta}_1, \hat{\theta}_2$  with  $J_P(k) + (J_{PN}(k) - J_{PN}(k; \hat{\theta}_B(k+\zeta)))$  (lines 16-17 in Alg. 1).

---

**Algorithm 1:** DPMCL algorithm.

---

```

Initialize  $\hat{\theta}_1, \hat{\theta}_2, D_P, D_N$ 
while  $k = 1, 2, 3, \dots k \times \Gamma$  do
   $i = 0$ 
  while  $i < \rho$  do
    Step 1: Generalization
    Get  $b_N \in D_N(k)$  Update  $\hat{\theta}_1(k), \hat{\theta}_2(k)$  with  $J_N(k)$ 
    Step 2: Catastrophic Forgetting
    Get  $J_P(k)$  with  $b_P \in D_P(k)$ 
    Get  $b_{PN} = b_P \cup b_N$  and copy  $\hat{\theta}_2$  into  $\hat{\theta}_B$ 
     $j = 0$ 
    while  $j + 1 \leq \zeta$  do
      Update  $\hat{\theta}_B(k)$  with  $J_{PN}(k; \hat{\theta}_B(k))$ .
       $j = j + 1$ 
    end
    Update  $\hat{\theta}_2(k), \hat{\theta}_1(k)$  with  $J_P(k) + (J_{PN}(k) - J_{PN}(k; \hat{\theta}_B(k+\zeta)))$ .
     $i = i + 1$ 
  end
  Update  $D_P(k)$  with  $D_N(k)$ .
end

```

---

## B.5 COMPARATIVE METHODS

The five methods are Naive, ER, FTML, OML and ANML. **Naive** For the naive implementation, we use the training data for each task to train our approach. The core idea is to greedily learn any new task. We run gradient updates for each task data for a predetermined number of epochs.

---

**Algorithm 2:** Naive algorithm.

---

```

Initialize  $\theta(k)$ .
while  $j < \text{num tasks}$  do
  Initialize task data  $D_N$ 
   $k = 0$ 
  while  $k < N$  do
     $b_N \in D_N$ 
    Update  $\theta$ 
     $k = k + 1$ 
  end
end

```

---

**Experience-Replay (ER (Lin, 1992)):** 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 start of every new task, we use the samples from the task memory array for training the network multi-

ple epochs through the data. This method focuses on minimizing catastrophic forgetting.

---

**Algorithm 3:** Experience Replay Algorithm.

---

```

Initialize  $\theta(k)$  and  $D_{PN}$ 
while  $j < \text{num tasks}$  do
  Initialize task data  $D_N$  and append to  $D_{PN}$ 
   $k = 0$ 
  while  $k < N$  do
     $b_{PN} \in D_{PN}$ 
    Update  $\theta$ 
     $k = k + 1$ 
  end
end

```

---

**Follow the Meta Leader (FTML), (Finn et al., 2019):** We follow the meta training testing procedures described in (Finn et al., 2019) 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 inner loop and the outer loop updates are performed by using the gradients of the cost function.

---

**Algorithm 4:** Our FTML implementation.

---

```

Initialize  $\theta(t)$ .
Initialize  $D_{PN}$ 
while  $j < \text{num tasks}$  do
  Initialize  $D_N$  and append to  $D_{PN}$  for  $N_{meta}$  samples in  $D_{PN}$  do
    Update  $\theta(k)$  to obtain  $\tilde{\theta}(k)$ 
  end
  for  $N_{grad}$  samples in  $D_N$  do
    Update  $\theta(k)$  using cost calculated with  $\tilde{\theta}$ .
  end
end

```

---

**Online Meta Continual Learning ( OML, (Javed and White, 2019)):** The learning process of this method is the same as that of the one in (Finn et al., 2019) with the key difference being the use of the representation network. The algorithm is provided in (Javed and White, 2019). 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.

---

**Algorithm 5:** Our OML Implementation.

---

```

Initialize  $\theta_1(t), \theta_2(t)$ .
Initialize  $D_{PN}$ 
while  $j < \text{num tasks}$  do
  Initialize  $D_N$  and append to  $D_{PN}$  for  $N_{meta}$  samples in  $D_{PN}$  do
    Update  $\theta_2(k)$  to obtain  $\tilde{\theta}_2(k)$ 
  end
  for  $N_{grad}$  samples in  $D_N$  do
    Update  $\theta(k)$  using cost calculated with  $\tilde{\theta}_2$  and  $\theta_1$ 
  end
end

```

---

**Neuromodulated Meta Learning (ANML (Beaulieu et al., 2020)):** Similar to the OML case, the learning process is the same as that of FTML with the key difference being the neuromodulatory network which is in addition to the representation learning network. The algorithm is provided in (Beaulieu et al., 2020). Similar to the earlier

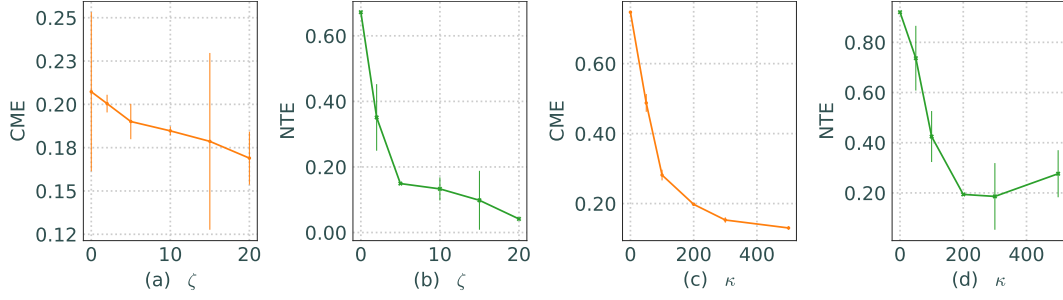


Figure 2: Cumulative error (CME) and new task error (NTE) trends with respect to (a,b)  $\zeta$  with  $\rho = 200$  and (c, d)  $\rho$  with  $\zeta = 2$ . For each value of  $\rho$  or  $\zeta$ , we learn a total of 50 tasks incrementally. We perform 50 repetitions of this learning. Next, we calculate the  $\mu$  and  $\sigma_{err}$  value of cumulative error and new task error over these 50 runs and record them. After we have computed these values for each value of  $\rho$  or  $\zeta$ , we plot the mean values with error bars from  $\sigma_{err}$  as a function of  $\zeta$  or  $\rho$ . We apply a Gaussian smoothing filter with a standard deviation of 2.

scenario, a the neuromodulatory network is learned while the tasks are being observed.

---

**Algorithm 6:** Our OML Implementation.

---

Initialize Network 1 with parameters  $\theta_1(t)$

Initialize Network 2 with  $\theta_2(t)$

Initialize  $D_{PN}$

**while**  $j < \text{num tasks}$  **do**

    Initialize  $D_N$  and append to  $D_{PN}$

**for**  $N_{meta}$  samples in  $D_{PN}$  **do**

        Get output by multiplying Network 1 and Network 2 outputs(similar to gating process in Beaulieu et al. (2020)).

        Update  $\theta_2(k)$

**end**

**for**  $N_{grad}$  samples in  $D_N$  **do**

        Get output by multiplying Network 1 and Network 2 outputs(similar to gating process in Beaulieu et al. (2020)).

        Update  $\theta_1(k)$  using cost calculated with  $\theta_2$  and  $\theta_1$

**end**

**end**

---

**Meta Experience Replay (MER (Riemer et al., 2018)):** For implementation of this method, we flow algorithm 1 in the original paper (Riemer et al., 2018) with the key difference that the experience replay part is kept similar to DPMCL that is utilized in the study. This is done to ensure that the comparisons are fair. Therefore, we use the reptile updates but with random sampling for maintaining the experience replay array

## B.6 ADDITIONAL RESULTS

Table 1: cumulative error (CME) and new task error (NTE) value for different data sets. The mean and standard error of the mean are reported. These values are calculated by averaging across 50 repetitions.

		Naive	DPMCL	FTML	OML	ANML	ER	MER
SINE	CME	0.3(0)	1.12(0)	2.28(0)	3.13(0)	9.85(0)	1.7(0)	10.5(0)
	NTE	0.01(0)	3.89(0)	1.6(0)	1.3(0)	10.2(7)	4.5(0)	2.28(0)
OMNI	CME	0.979(0)	0.175(0.007)	0.221(0.010)	0.268(0.012)	0.976(0.001)	0.195(0.008)	0.163(0.010)
	NTE	0.001(0)	0.189(0.091)	0.512(0.098)	0.077(0.053)	0.945(0)	0.291(0.071)	0.002(0)
MNIST	CME	0.912(0)	0.015(0.001)	0.018(0.001)	0.033(0.004)	0.835(0.002)	0.022(0.001)	0.049(0.002)
	NTE	0.001(0)	0.003(0)	0.009(0)	0.010(0)	0.884(0)	0.023(0.001)	0.009(0.001)
CIFAR10	CME	0.949(2)	0.475(0.003)	0.634(0.006)	0.630(0.004)	0.906(0.016)	0.451(0.002)	0.682(0.006)
	NTE	0.01(0)	0.273(0.008)	0.140(0.005)	0.057(0.003)	0.606(0.140)	0.436(0.008)	0.036(0.003)

For Naive, the orders are  $10^{-5}$ , for instance, 1 refers to  $1 \times 10^{-5}$

### B.6.1 THE PERFORMANCE OF THE THIRD TERM

In Fig. 2, we plot the variation in CME and NTE for the 50<sup>th</sup> task in the OMNI data set with respect to hyperparameter  $\zeta$ . In DPMCL, the value of  $\zeta$  controls the magnitude of the third term in the update rule, namely,  $(J_{PN}(k) - J_{PN}(k; \hat{\theta}(k + \zeta)))$ , where  $J_{PN}(k; \hat{\theta}(k + \zeta))$  is an approximation of the optimal cost. Therefore, the larger the value of  $\zeta$ , the greater the value of  $(J_{PN}(k) - J_{PN}(k; \hat{\theta}(k + \zeta)))$ , and the third term is zero when  $\zeta = 0$ . We observe from Figs. 2(a) and (b) that when the value of  $\zeta$  is zero, DPMCL generalizes poorly to a new task (low NTE) and incurs catastrophic forgetting (low CME). As the value of  $\zeta$  is increased from zero, however, forgetting reduces (decreasing CME in Fig. 2(a)) and generalization improves (decreasing NTE in Fig. 2(b)). Furthermore, the best value of forgetting with generalization is achieved when  $\zeta = 20$ . We know from our theoretical analysis that the larger the value of  $\zeta$ , the closer  $J_{PN}(k; \hat{\theta}(k + \zeta))$  is to the optimal cost. Minimizing the difference  $(J_{PN}(k) - J_{PN}(k; \hat{\theta}(k + \zeta)))$  would push the model toward optimal generalization and catastrophic forgetting. This behavior is observed in Fig. 2(a, b).

Next, we analyzed the impact of  $\rho$ , the parameter controlling the total number of alternative updates. From our theory, we know that an appropriate number of  $\rho$  allows DPMCL to balance forgetting and generalization. We observe this behavior from Figs. 2(c) and (d). Note that with an increase in  $\rho$  from zero, forgetting keeps on reducing (decreasing CME on Fig. 2(c).) When  $\rho > 250$ , however, we observe that while forgetting does improve, generalization no longer improves (Fig. 2(d)). In fact, we observe an inflection point on NTE between  $\rho = 200$  and  $\rho = 250$ , where a balance between CME and NTE is achieved. DPMCL is designed in such a way that with choice of  $\rho$ , this balance point can be engineered.

### REFERENCES

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