A THEORETICAL FRAMEWORK FOR SPATIO-TEMPORAL CONTINUAL LEARNING[*]

KRISHNAN RAGHAVAN AND PRASANNA BALAPRAKASH[†]

Abstract. Continual learning (CL) is a field concerned with learning a series of inter-related task with the tasks typically defined in the sense of either regression or classification. In recent years, CL has been studied extensively when these tasks are defined using Euclidean data– data, such as images, that can be described by a set of vectors in an n-dimensional real space. However, the literature is quite sparse, when the data corresponding to a CL task is nonEuclidean– data , such as graphs, point clouds or manifold, where the notion of similarity in the sense of Euclidean metric does not hold. For instance, a graph is described by a tuple of vertices and edges and similarities between two graphs is not well defined through a Euclidean metric. Due to this fundamental nature of the data, developing CL for nonEuclidean data presents several theoretical and methodological challenges. In particular, CL for graphs requires explicit modelling of nonstationary behavior of vertices and edges and their effects on the learning problem. Therefore, in this work, we develop a adaptive dynamic programming viewpoint for CL with graphs. In this work, we formulate a two-player sequential game between the act of learning new tasks (generalization) and remembering previously learned tasks (forgetting). We prove mathematically the existence of a solution to the game and demonstrate convergence to the solution of the game. Finally, we demonstrate the efficacy of our method on a number of graph benchmarks with a comprehensive ablation study while establishing state-of-the-art performance.

Key words. Continual learning, dynamic programming, vertex edge random graph, optimal control, Stackelberg Equillibrium

MSC codes. 68T05, 37N35, 91A99

1. Introduction.

The contributions of this paper

- Single cascaded model of the end to end training of neural network.
  - Makes analysis easier compared to a lack of characterization of the certain parts of the network.
  - explainability interpretability through system theoretic characterizations, a big plus for most mathematical communities.
  - Typical Neural ode models are limited to res-net driven interpretations
- We will show how this structure can be utilized to analyze a control problem in climate modelling.
- we will demonstrate in theory the guarantees you can achieve and in empirical results that this does converge.
- We will start with simple neural network and build the structure for a regular neural network.
- Next we will extend this to the case of graph neural networks.

This work is comprised of mathematical sophistication and will not go in any of the standard ML conferences.

[*]Submitted to the editors DATE.

Funding: This work was funded by the Fog Research Institute under contract no. FRI-454.

[†]Mathematics and Computer Science, Argonne National Laboratory (kraghavan@anl.gov).

This page is left blank intentionally.

1.1. Notations. Let the interval $\boldsymbol{\Omega} = [0, \Omega]$ for $\Omega \in \mathbb{R}^+$ represent the sample space for all tasks instances (the total interval in which tasks may be observed) and $\mathbb{R}^+$ referring to the set of real positive numbers, we will refer to this interval as domain. Where task instances are synonymous with time, then $\boldsymbol{\Omega}$ is the time interval over which the data is observed. The sample space is considered continuous and therefore intervals in Ω are denoted by $\boldsymbol{t} \subseteq \boldsymbol{\Omega} \mid \boldsymbol{t} = [0, t]$. Any partial derivatives are given by $\partial_y(x)$—the partial derivative of x with respect to y. An asymptotic value is indicated by a superscript ∞, optimal value by a superscript $*$ and iterations are indicated by a superscript in parenthesis. For instance, V_t^* refers to the optimal value of V with respect to the task t notations for task and time should be resolved and $\boldsymbol{w}^{(n)}$ provides the value of $\boldsymbol{w}$ after n updates. Furthermore, we will denote $\boldsymbol{w}$ to collectively denote a column vector of all parameters. To indicate weights at d layers in a deep network, we will describe $\boldsymbol{w} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d]^\dagger$, where $\dagger$ $\intercal$ can be used for transpose $\dagger$ is often used for pseudoinverse is the transpose operator. We consider the class of problems where the tasks are generated from a differential equation (ordinary or partial differential equation) continuously over all intervals in $\boldsymbol{\Omega}'$[. Therefore, we will first describe data arising from a task, then describe the function approximator and state the learning problem.

1.2. Data. Here we need some preliminaries from tensor analysis to proceed further. The notations below are directly derived from [1] and for additional details, the reference should be referred.

2. Preliminaries from Tensor Analysis.

A d^{th} order (or ways or mode) tensor $\boldsymbol{\mathcal{X}}$ is viewed as a multi-dimensional array such that

$$\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4 \times \cdots I_p} \tag{2.1}$$

, where the order can be thought of as the number of dimensions in the tensor. For instance, a tensor of order 0 is a scalar denoted by a lowercase letters, i.e., x whereas a vector is an order one tensor which we will denote with lowercase bold alphabets such as $\boldsymbol{x}$. A tensor of order 2 is a matrix denoted by uppercase alphabets. Any tensor of order greater than 2 are denoted by bolder Euler scripts letters such as $\boldsymbol{\mathcal{X}}$. The i_{th} element of a vector $\boldsymbol{x}$ is denoted by x_i, while the $(i,j)_{th}$ element of a matrix $\boldsymbol{X}$ is denoted by x_{ij} and the $(i, j, \cdots)_{th}$ element of a tensor $\boldsymbol{\mathcal{X}}$ is given by $x_{ijk\cdots}$.

We will make the indices run from 1 to their capital letters such that $i = 1, \cdots, I$. For any sequence $\boldsymbol{A}$, the n^{th} element is denoted by a superscript in parenthesis as $\boldsymbol{A}^{(n)}$. Subarrays are formed when a subset of the indices is fixed. For matrices, these are the rows and columns. A colon is used to indicate all elements of a mode. Thus the j_{th} column of a matrix $\boldsymbol{X}$ is denoted $\boldsymbol{x}_{j:}$ and the i_{th} row of a matrix is indicated $\boldsymbol{x}_{:i}$. Fibers are higher order analogues of matrix rows and column and are obtained by fixed every index but one, slices are two dimensional analogues and are obtained by fixing all but two indices. For a d_{th} order tensor we write $\boldsymbol{x}_{i:k\cdots}$ is one fiber and $\boldsymbol{X}_{::k\cdots}$ as a slice. The norm of the tensor $\boldsymbol{\mathcal{X}}$ is given as the square root of the sum of the square of all its elements, i.e

$$\|\boldsymbol{\mathcal{X}}\| = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_p=1}^{I_p} x^2_{i_1 i_1 i_2 \cdots i_p}}. \tag{2.2}$$

The inner product of two tensors $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ is given as

$$\langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_d=1}^{I_p} x_{i_1 i_1 i_2 \cdots i_p} y_{i_1 i_1 i_2 \cdots i_p}. \tag{2.3}$$

A $n-$ mode matrix product (denoted by $\times_n$) of a tensor $\boldsymbol{\mathcal{X}}$ with a matrix $\boldsymbol{A} \in \mathbb{R}^{J \times I_n}$ is denoted as $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{A}$ and is of size $I_1 \times \cdots I_{n-1} \times J \times I_n + 1 \cdots I_p$ where element-wise we have

$$(\boldsymbol{\mathcal{X}} \times_n \boldsymbol{A})_{i_1 \cdots i_{n-1} j i_{n+1}} \cdots i_p = \sum_{i_n=1}^{I_n} = x_{i_1 i_2 \cdots i_N} a_{j i_n}. \tag{2.4}$$

Similarly, the $n-$ mode tensor-vector product $\boldsymbol{\mathcal{X}} \overline{\times}_n \boldsymbol{v}$ is given as

$$(\boldsymbol{\mathcal{X}} \overline{\times}_n \boldsymbol{A})_{i_1 \cdots i_{n-1} i_{n+1}} \cdots i_p = \sum_{i_n=1}^{I_n} = x_{i_1 i_2 \cdots i_N} v_{i_n}, \tag{2.5}$$

where we have computer the inner-product of each mode-n fiber with the vector $\boldsymbol{v}$.

Data will be represented in this work by measurable functions defined on each interval in the domain. That is for every subinterval $t \subset \boldsymbol{\Omega}$, we will denote the data (states) by a tensor $\boldsymbol{\mathcal{X}}(t) \in \mathbb{R}^{\boldsymbol{t} \times I_1 \times I2 \times \cdots \times I_p}$ where $t \in \boldsymbol{\Omega}$. The intuition here is that, for each sub-interval in $\boldsymbol{\Omega}$, we can obtain a p dimensional random tensor that will constitute the data. Since, an ordered subinterval on the domain can lead to any random tensor, we have a random-valued tensor field that is random with respect to time. This data could be a 3 channel image obtained by a camera or a matrix. The notation generalizes to all these problems.

2.1. Learning Objective:. For each instant t, we presume a tensor $\boldsymbol{\mathcal{X}}(t)$ such that the associated problem is to learn an unknown function $g(.) : \mathbb{R}^{t \times I_1, I_2, \cdots, I_p} \to \mathbb{R}^{\boldsymbol{t} \times I_1, I_2, \cdots, I_q}$ such that $\boldsymbol{\mathcal{Y}}(t) = \rho(\boldsymbol{\mathcal{X}})(t)$. When the function involves a PDE/ODE or regression in standard ML, ρ is a push-forward map. In particular, $\boldsymbol{\mathcal{Y}}(t) \in \mathbb{R}^{t \times I_1 \times I2 \times \cdots \times I_p}$ is also a tensor valued variable such that

$$\boldsymbol{\mathcal{X}}(t) \to \rho \to \boldsymbol{\mathcal{Y}}(t),$$

where $\boldsymbol{\mathcal{Y}}(t)$ is considered output and $\boldsymbol{\mathcal{X}}(t)$ is the input. We seek to approximate this unknown function ρ using a neural network parameterized by some weight parameters. Therefore, we define a compact parameter space $\mathcal{W} \subset \mathbb{R}^m$ and a vector-valued function that provides an $\mathbb{R}^m$ weight function that provides a weight vector for $t \in \boldsymbol{\Omega}$ where $\boldsymbol{w}(t) : \boldsymbol{\Omega} \to \mathcal{W}$.

Here, we will strive a generic neural network architecture that is applicable to all different architectures. We describe this NN output be denoted as $\boldsymbol{\mathcal{X}}(t), \forall t \subset \boldsymbol{\Omega}$

$$\boldsymbol{\mathcal{Y}}(t) = g(\boldsymbol{\mathcal{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d]^\dagger, [\phi_1, \phi_1, \cdots, \phi_d]^\dagger)(t) \tag{2.6}$$

where, ϕ_i is the activation function of the i^{th} layer in the NN, $\boldsymbol{w}_i$ is the weight at the layer i. This representation is inclusive of feedforward networks, convolutional neural network and graph neural networks. Therefore, a composition of d layers is given as

$$\begin{aligned} \boldsymbol{\mathcal{Y}}(t) &= g(\boldsymbol{\mathcal{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d]^\dagger, [\phi_1, \phi_1, \cdots, \phi_d]^\dagger)(t) \\ &= \phi_d(\phi_{d-1}(\phi_{d-2}(\phi_{d-3}(\cdots \phi_1(\boldsymbol{\mathcal{X}}; \boldsymbol{w}_1); \cdots; \boldsymbol{w}_{d-3}); \boldsymbol{w}_{d-2}); \boldsymbol{w}_{d-1}); \boldsymbol{w}_d)(t) \end{aligned} \tag{2.7}$$

This definition naturally extends to different architectures irrespective of the layerwise operations. Specifically, for $t, \Delta t \in \boldsymbol{\Omega}$, define the interval $[t, t+\Delta t]$ and let $\boldsymbol{\mathfrak{X}}(t)$ represent a tensor valued random variable. Denote the NN model as $g(\boldsymbol{\mathfrak{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d]^\dagger, [\phi_1, \phi_1, \cdots, \phi_d]^\dagger)(t)$ with a associated loss function given as $\ell : \boldsymbol{\Omega} \to \mathbb{R}$. Let $\acute{J}([t, t+\Delta t]; g(\boldsymbol{\mathfrak{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d]^\dagger, [\phi_1, \phi_1, \cdots, \phi_d]^\dagger)([t, t+\Delta t])) = \int_{\tau=t}^{t+\Delta t} \ell(\tau; g(\boldsymbol{\mathfrak{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d]^\dagger, [\phi_1, \phi_1, \cdots, \phi_d]^\dagger)(\tau))$ be the cost over the interval $[t, t+\Delta t]$. Then, a learning task $\mathcal{T}([t, t+\Delta t])$ is described by the tuple

$$\Big(\boldsymbol{\mathfrak{X}}([t, t+\Delta t]), \acute{J}([t, t+\Delta t]; g(\boldsymbol{\mathfrak{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d]^\dagger, [\phi_1, \phi_1, \cdots, \phi_d]^\dagger)([t, t+\Delta t]))\Big). \tag{2.8}$$

This notation, easily extends to a collection of tasks. For instance, all the tasks in the interval $[0, t]$ are collectively provided by $\mathcal{T}([0, t])$. As we use $\boldsymbol{t}$ to represent the interval $[0, t]$, it follows that $\mathcal{T}([0, t])$ is rewritten as $\mathcal{T}(\boldsymbol{t})$ which represents all tasks in the interval $[0, t]$ where $\mathcal{T}(\boldsymbol{t}) = \Big(\boldsymbol{\mathfrak{X}}(\boldsymbol{t}), \acute{J}(g(\boldsymbol{\mathfrak{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d]^\dagger, [\phi_1, \phi_1, \cdots, \phi_d]^\dagger))(\boldsymbol{t})\Big)$. We then have our optimization problem at a particular time step as

$$\boldsymbol{w}^*(\boldsymbol{t}) = \underset{\boldsymbol{w}(\boldsymbol{t}) \in \boldsymbol{\mathcal{W}}}{\arg\min}\, \acute{J}(g(\boldsymbol{\mathfrak{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d]^\dagger, [\phi_1, \phi_1, \cdots, \phi_d]^\dagger))(\boldsymbol{t}), \tag{2.9}$$

where $\boldsymbol{\mathcal{W}}$ denotes the parameter search space. Specifically, the problem of learning is posed as an optimization problem with an objective to drive $\acute{J} \to 0$ as $t \to \infty$, where $t > 0$ is the training time.

3. Differential Learning Approach.

In this section, we reformulate the optimization problem described in the previous section such that the nested structure is decomposed into a system of differential equations and transients of each of these levels can be traced as part of the learning problem. First, we introduce a new variable $\boldsymbol{z}_d$ to represent the output of the top-most hidden layer such that

$$\acute{J}(g(\boldsymbol{\mathfrak{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d]^\dagger, [\phi_1, \phi_1, \cdots, \phi_d]^\dagger))(\boldsymbol{t}) \approx J(g(\boldsymbol{z}_d; [\boldsymbol{w}_d]^\dagger, [\phi_d]^\dagger))(\boldsymbol{t}) \tag{3.1}$$

where $\boldsymbol{z}_d(\boldsymbol{t}) = g(\boldsymbol{\mathfrak{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{d-1}]^\dagger, [\phi_1, \phi_1, \cdots, \phi_{d-1}]^\dagger))$ is the output of the NN. It follows that the loss function depends on $\boldsymbol{z}^{(d)}$, and a constraint must be introduced on $\boldsymbol{z}^{(d)}$ to assert equivalence with the traditional cost function. This provides the new cost as

$$\begin{aligned} &J(g(\boldsymbol{z}_d; [\boldsymbol{w}_d]^\dagger, [\phi_d]^\dagger))(\boldsymbol{t}) \quad \text{such that} \\ &\qquad\qquad \boldsymbol{z}_d(\boldsymbol{t}) = g(\boldsymbol{\mathfrak{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{d-1}]^\dagger, [\phi_1, \phi_2, \cdots, \phi_{d-1}]^\dagger)(\boldsymbol{t}). \end{aligned}$$

Next, introduce another variable $\boldsymbol{z}_{d-1}$ that is constrained to mimic the penultimate layer in the NN such that the topmost layer depends on $\boldsymbol{z}_{d-1}$ instead of the input. Therefore, a new constraint is introduced on $\boldsymbol{z}_{d-1}(t)$ such that

$$\begin{aligned} &J(g(\boldsymbol{z}_d; [\boldsymbol{w}_d]^\dagger, [\phi_d]^\dagger))(\boldsymbol{t}) \quad \text{such that} \\ &\qquad \boldsymbol{z}_d(\boldsymbol{t}) = g(\boldsymbol{z}_{d-1}; [\boldsymbol{w}_d]^\dagger, [\phi_d]^\dagger))(\boldsymbol{t}) \\ &\qquad \boldsymbol{z}_{d-1}(\boldsymbol{t}) = g(\boldsymbol{\mathfrak{X}}; [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{d-2}]^\dagger, [\phi_1, \phi_2, \cdots, \phi_{d-2}]^\dagger)(\boldsymbol{t}). \end{aligned} \tag{3.2}$$

In the same vein, one may define a variable at each layer of the NN constrained to mimic all the information prior to that particular layer in the NN, and as a result, the overall optimization problem is rewritten as

$$\begin{aligned}
J(g(\boldsymbol{z}_d;[\boldsymbol{w}_d]^\dagger,[\phi_d]^\dagger))(\boldsymbol{t}),\quad & \text{such that}\\
& \boldsymbol{z}_d(\boldsymbol{t}) = g(\boldsymbol{z}_{d-1};[\boldsymbol{w}_d]^\dagger,[\phi_d]^\dagger))(\boldsymbol{t}),\\
& \boldsymbol{z}_{d-1}(\boldsymbol{t}) = g(\boldsymbol{z}_{d-2};[\boldsymbol{w}_{d-1}]^\dagger,[\phi_{d-1}]^\dagger))(\boldsymbol{t}),\\
& \boldsymbol{z}_{d-2}(\boldsymbol{t}) = g(\boldsymbol{z}_{d-3};[\boldsymbol{w}_{d-2}]^\dagger,[\phi_{d-2}]^\dagger))(\boldsymbol{t}),\\
& \vdots \qquad\qquad\qquad\qquad (3.3)\\
& \boldsymbol{z}_1(\boldsymbol{t}) = g(\boldsymbol{\mathfrak{X}}(\boldsymbol{t});[\boldsymbol{w}_1]^\dagger,[\phi_1]^\dagger)(\boldsymbol{t}),
\end{aligned}$$

(3.4)

Over the long horizon, the above optimization problem boils to

$$\begin{aligned}
& \underset{\boldsymbol{w}(\boldsymbol{t})}{min} \int_{\tau=0}^{\Omega-\boldsymbol{t}} J(g(\boldsymbol{z}_d;[\boldsymbol{w}_d]^\dagger,[\phi_d]^\dagger))(\boldsymbol{t}+\tau),\\
& \text{such that}\\
& \boldsymbol{z}_d(\boldsymbol{t}) = g(\boldsymbol{z}_{d-1};[\boldsymbol{w}_d]^\dagger,[\phi_d]^\dagger))(\boldsymbol{t}),\\
& \boldsymbol{z}_{d-1}(\boldsymbol{t}) = g(\boldsymbol{z}_{d-2};[\boldsymbol{w}_{d-1}]^\dagger,[\phi_{d-1}]^\dagger))(\boldsymbol{t}),\\
& \boldsymbol{z}_{d-2}(\boldsymbol{t}) = g(\boldsymbol{z}_{d-3};[\boldsymbol{w}_{d-2}]^\dagger,[\phi_{d-2}]^\dagger))(\boldsymbol{t}),\\
& \vdots\\
& \boldsymbol{z}_1(\boldsymbol{t}) = g(\boldsymbol{\mathfrak{X}};[\boldsymbol{w}_1]^\dagger,[\phi_1]^\dagger)(\boldsymbol{t}).
\end{aligned}$$

where we denote $\underset{\boldsymbol{w}(\boldsymbol{t})}{min} \int_{\tau=0}^{\Omega-\boldsymbol{t}} J(g(\boldsymbol{z}_d;[\boldsymbol{w}_d]^\dagger,[\phi_d]^\dagger))(\boldsymbol{t}+\tau)$, as $V^*(\boldsymbol{t})$.

One may observe from (2.9), that due to the presence of nested nonlinear function, it is difficult to characterize the behavior of hidden layers during the learning process. One of the main contributions of this work is to explicitly derive the dynamics of the hidden layers of the neural network to characterizing this behavior. We will discuss this next.

Given the problem

$$V^*(\boldsymbol{t}) = \underset{\boldsymbol{w}(\boldsymbol{t})}{min} \int_{\tau=0}^{\Omega-\boldsymbol{t}} J(g(\boldsymbol{z}_d;[\boldsymbol{w}_d]^\dagger,[\phi_d]^\dagger))(\boldsymbol{t}+\tau) \tag{3.5}$$

Split the integral with the time interval $,t$rewrite the optimization problem as

$$\begin{aligned}
V^*(\boldsymbol{t}) = \underset{\boldsymbol{w}(\boldsymbol{t})}{min} \Big[\int_{\tau=0}^{\Delta \boldsymbol{t}} & \quad J(g(\boldsymbol{z}_d;[\boldsymbol{w}_d]^\dagger,[\phi_d]^\dagger))(\boldsymbol{t}+\tau)\\
+ \int_{\tau=\Delta t}^{\Omega-\boldsymbol{t}} & \quad J(g(\boldsymbol{z}_d;[\boldsymbol{w}_d]^\dagger,[\phi_d]^\dagger))(\boldsymbol{t}+\tau)\Big]
\end{aligned} \tag{3.6}$$

Using the policy $\boldsymbol{w}(\boldsymbol{t})$ if we begin at $\boldsymbol{t}$, $V(\boldsymbol{t})^*$ provides the optimal cost over the complete interval. Since, $undersetw(\boldsymbol{t})min \int_{\tau=0}^{\Omega-\boldsymbol{t}} J(g(\boldsymbol{z}_d;[\boldsymbol{w}_d]^\dagger,[\phi_d]^\dagger))(\boldsymbol{t}+\tau)$. is $V^*(\boldsymbol{t})$, then it stands to reason that $\int_{\tau=\Delta t}^{\Omega-\boldsymbol{t}} J(g(\boldsymbol{z}_d;[\boldsymbol{w}_d]^\dagger,[\phi_d]^\dagger))(\boldsymbol{t}+\tau)$ is the optimal cost after $\boldsymbol{t}$. That is, it is the optimal cost for the interval $\boldsymbol{\Omega}^C$ while following policy $\boldsymbol{w}(\boldsymbol{t})$. This provides

$$V^*(\boldsymbol{t}) = \underset{\boldsymbol{w}(\boldsymbol{t})}{min} \Big[\int_{\tau=0}^{\Delta \boldsymbol{t}} \quad J(g(\boldsymbol{z}_d;[\boldsymbol{w}_d]^\dagger,[\phi_d]^\dagger))(\boldsymbol{t}+\tau) + V(\boldsymbol{t}+\Delta t)\Big] \tag{3.7}$$

Now, all the information about the future must be approximated using $\boldsymbol{t}$. To do this approximation, we will write the Taylor series expansion of $V^*(\boldsymbol{t}+\Delta t)$ around $\boldsymbol{t}$. Taylor series expression is expanded around

$$(\boldsymbol{t}, \boldsymbol{z}_d, \boldsymbol{w}_d).$$

to obtain

$$V(\boldsymbol{t}+\Delta t) \;=\; V(\boldsymbol{t}) + (\partial_{\boldsymbol{t}} V(\boldsymbol{t}))^{\dagger}\, \Delta_{\boldsymbol{t}} + (\partial_{\boldsymbol{z}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{z}}_d + (\partial_{\boldsymbol{w}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{w}}_d + \cdots, \tag{3.8}$$

Substitution into (3.13) reveals

$$\begin{aligned} V^*(\boldsymbol{t}) = \underset{\boldsymbol{w}(\boldsymbol{t})}{min} \Big[\int_{\tau=0}^{\Delta \boldsymbol{t}} & J(g(\boldsymbol{z}_d; [\boldsymbol{w}_d]^{\dagger}, [\phi_d]^{\dagger}))(\boldsymbol{t}+\tau) \\ & + V(\boldsymbol{t}) + (\partial_{\boldsymbol{t}} V(\boldsymbol{t}))^{\dagger}\, \Delta_{\boldsymbol{t}} + (\partial_{\boldsymbol{z}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{z}}_d + (\partial_{\boldsymbol{w}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{w}}_d + \cdots, \Big] \end{aligned} \tag{3.9}$$

Cancelling the common terms to obtained.

$$\begin{aligned} 0 = \underset{\boldsymbol{w}(\boldsymbol{t})}{min} \Big[\int_{\tau=0}^{\Delta \boldsymbol{t}} & J(g(\boldsymbol{z}_d; [\boldsymbol{w}_d]^{\dagger}, [\phi_d]^{\dagger}))(\boldsymbol{t}+\tau) \\ & + (\partial_{\boldsymbol{z}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{z}}_d + (\partial_{\boldsymbol{w}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{w}}_d + \cdots, \Big] + \underset{\boldsymbol{w}(\boldsymbol{t})}{min}\, (\partial_{\boldsymbol{t}} V(\boldsymbol{t}))^{\dagger}\, \Delta_{\boldsymbol{t}} \end{aligned} \tag{3.10}$$

This provides

$$\begin{aligned} -\underset{\boldsymbol{w}(\boldsymbol{t})}{min}\, (\partial_{\boldsymbol{t}} V(\boldsymbol{t}))^{\dagger}\, \Delta_{\boldsymbol{t}} = \underset{\boldsymbol{w}(\boldsymbol{t})}{min} \Big[\int_{\tau=0}^{\Delta \boldsymbol{t}} & J(g(\boldsymbol{z}_d; [\boldsymbol{w}_d]^{\dagger}, [\phi_d]^{\dagger}))(\boldsymbol{t}+\tau) \\ & + (\partial_{\boldsymbol{z}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{z}}_d + (\partial_{\boldsymbol{w}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{w}}_d + \cdots, \Big] \end{aligned} \tag{3.11}$$

Taking the continuous limit to get the higher order terms to obtain the following.

$$\begin{aligned} -\underset{\boldsymbol{w}(\boldsymbol{t})}{min}\, (\partial_{\boldsymbol{t}} V(\boldsymbol{t}))^{\dagger} = \underset{\boldsymbol{w}(\boldsymbol{t})}{min} \Big[\int_{\tau=0}^{\Delta \boldsymbol{t}} & J(g(\boldsymbol{z}_d; [\boldsymbol{w}_d]^{\dagger}, [\phi_d]^{\dagger}))(\boldsymbol{t}+\tau) \\ & + (\partial_{\boldsymbol{z}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{z}}_d + (\partial_{\boldsymbol{w}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{w}}_d, \Big] \end{aligned} \tag{3.12}$$

Thus, we obtain our problem as

$$\begin{aligned} -\left(\partial_{\boldsymbol{t}} V^*(\boldsymbol{t})\right) = \underset{\boldsymbol{w}(\boldsymbol{t})}{min} \Big[\int_{\tau=0}^{\Delta \boldsymbol{t}} & J(g(\boldsymbol{z}_d; [\boldsymbol{w}_d]^{\dagger}, [\phi_d]^{\dagger}))(\boldsymbol{t}+\tau) \\ & + (\partial_{\boldsymbol{z}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{z}}_d + (\partial_{\boldsymbol{w}_d} V(\boldsymbol{t}))^{\dagger}\, \dot{\boldsymbol{w}}_d \Big]. \end{aligned} \tag{3.13}$$

$$\begin{aligned}
-\left(\partial_{\boldsymbol{t}} V^*(\boldsymbol{t})\right) = \min_{\boldsymbol{w}(\boldsymbol{t})} \Big[& \int_{\tau=0}^{\Delta \boldsymbol{t}} J(g(\boldsymbol{z}_d; [\boldsymbol{w}_d]^\dagger, [\phi_d]^\dagger))(\boldsymbol{t}+\tau) \\
& + \left(\partial_{\boldsymbol{z}_d} V(\boldsymbol{t})\right)^\dagger \dot{\boldsymbol{z}}_d + \left(\partial_{\boldsymbol{w}_d} V(\boldsymbol{t})\right)^\dagger \dot{\boldsymbol{w}}_d \Big]. \\
& \text{such that} \\
& \boldsymbol{z}_d(\boldsymbol{t}) = g(\boldsymbol{z}_{d-1}; [\boldsymbol{w}_d]^\dagger, [\phi_d]^\dagger))(\boldsymbol{t}), \\
& \boldsymbol{z}_{d-1}(\boldsymbol{t}) = g(\boldsymbol{z}_{d-2}; [\boldsymbol{w}_{d-1}]^\dagger, [\phi_{d-1}]^\dagger))(\boldsymbol{t}), \\
& \boldsymbol{z}_{d-2}(\boldsymbol{t}) = g(\boldsymbol{z}_{d-3}; [\boldsymbol{w}_{d-2}]^\dagger, [\phi_{d-2}]^\dagger))(\boldsymbol{t}), \\
& \vdots \\
& \boldsymbol{z}_1(\boldsymbol{t}) = g(\boldsymbol{\mathcal{X}}; [\boldsymbol{w}_1]^\dagger, [\phi_1]^\dagger)(\boldsymbol{t}),
\end{aligned}$$

Differentiating $\boldsymbol{z}_d$ to obtain $\dot{\boldsymbol{z}}_d$ provides

$$\begin{aligned}
-\left(\partial_{\boldsymbol{t}} V^*(\boldsymbol{t})\right) = \min_{\boldsymbol{w}(\boldsymbol{t})} \Big[& \int_{\tau=0}^{\Delta \boldsymbol{t}} J(g(\boldsymbol{z}_d; [\boldsymbol{w}_d]^\dagger, [\phi_d]^\dagger))(\boldsymbol{t}+\tau) + \left(\partial_{\boldsymbol{z}_d} V(\boldsymbol{t})\right)^\dagger \dot{\boldsymbol{z}}_d + \left(\partial_{\boldsymbol{w}_d} V(\boldsymbol{t})\right)^\dagger \dot{\boldsymbol{w}}_d \Big] \\
& \text{such that} \\
& \dot{\boldsymbol{z}}_d(\boldsymbol{t}) = g'(\boldsymbol{z}_{d-1}; [\boldsymbol{w}_d]^\dagger, [\phi_d]^\dagger)(\boldsymbol{t}) \\
& \Big[\partial_{\boldsymbol{z}_{d-1}(\boldsymbol{t})} g(\boldsymbol{z}_{d-1}; [\boldsymbol{w}_d]^\dagger, [\phi_d]^\dagger)(\boldsymbol{t}) \dot{\boldsymbol{z}}_{d-1}(\boldsymbol{t}) \\
& + \partial_{\boldsymbol{w}_d(\boldsymbol{t})} g(\boldsymbol{z}_{d-1}; [\boldsymbol{w}_d]^\dagger, [\phi_d]^\dagger)(\boldsymbol{t}) \dot{\boldsymbol{w}}_d(\boldsymbol{t})\Big], \\
& \dot{\boldsymbol{z}}_{d-1}(\boldsymbol{t}) = g'(\boldsymbol{z}_{d-2}; [\boldsymbol{w}_{d-1}]^\dagger, [\phi_{d-1}]^\dagger)(\boldsymbol{t}) \\
& \Big[\partial_{\boldsymbol{z}_{d-2}(\boldsymbol{t})} g(\boldsymbol{z}_{d-2}; [\boldsymbol{w}_{d-1}]^\dagger, [\phi_{d-1}]^\dagger)(\boldsymbol{t}) \dot{\boldsymbol{z}}_{d-2}(\boldsymbol{t}) \\
& + \partial_{\boldsymbol{w}_{d-1}(\boldsymbol{t})} g(\boldsymbol{z}_{d-2}; [\boldsymbol{w}_{d-1}]^\dagger, [\phi_{d-1}]^\dagger)(\boldsymbol{t}) \dot{\boldsymbol{w}}_{d-1}(\boldsymbol{t})\Big], \\
& \dot{\boldsymbol{z}}_{d-2}(\boldsymbol{t}) = g'(\boldsymbol{z}_{d-3}; [\boldsymbol{w}_{d-2}]^\dagger, [\phi_{d-2}]^\dagger)(\boldsymbol{t}) \\
& \Big[\partial_{\boldsymbol{z}_{d-3}(\boldsymbol{t})} g(\boldsymbol{z}_{d-3}; [\boldsymbol{w}_{d-2}]^\dagger, [\phi_{d-2}]^\dagger)(\boldsymbol{t}) \dot{\boldsymbol{z}}_{d-3}(\boldsymbol{t}) \\
& + \partial_{\boldsymbol{w}_{d-2}(\boldsymbol{t})} g(\boldsymbol{z}_{d-3}; [\boldsymbol{w}_{d-2}]^\dagger, [\phi_{d-2}]^\dagger)(\boldsymbol{t}) \dot{\boldsymbol{w}}_{d-2}(\boldsymbol{t})\Big], \\
& \vdots \\
& \dot{\boldsymbol{z}}_1(\boldsymbol{t}) = g'(\boldsymbol{\mathcal{X}}; [\boldsymbol{w}_1]^\dagger, [\phi_1]^\dagger)(\boldsymbol{t}) \Big[\partial_{\boldsymbol{\mathcal{X}}(\boldsymbol{t})} g(\boldsymbol{\mathcal{X}}; [\boldsymbol{w}_1]^\dagger, [\phi_1]^\dagger))(\boldsymbol{t}) \dot{\boldsymbol{\mathcal{X}}}(\boldsymbol{t}) \\
& + \partial_{\boldsymbol{w}_1(\boldsymbol{t})} g(\boldsymbol{\mathcal{X}}; [\boldsymbol{w}_1]^\dagger, [\phi_1]^\dagger))(\boldsymbol{t}) \dot{\boldsymbol{w}}_1(\boldsymbol{t}) \qquad \blacksquare
\end{aligned}$$

which is our end to end map of the learning problem.

this page is left blank intentionally

REFERENCES

[1] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM review, 51 (2009), pp. 455–500.