The effect of architecture during continual learning

Allyson Hahn *ADD EMAIL!!*
Mathematics and Computer Science
Argonne National Laboratory

Krishnan Raghavan *kraghavan@anl.gov*
Mathematics and Computer Science
Argonne National Laboratory

Reviewed on OpenReview: *https://openreview.net/forum?id=XXXX*

Abstract

We consider the effects of Neural Network architecture in the setting of continual learning. Using dynamic programming we complete a bilevel optimization to determine the optimal architecture for the current and previous task data followed by the optimal weights for the network.

1 Introduction

With the advent of large scale AI models, the computational expense of training such model has also increased drastically; training these models efficiently often requires large scale high performance computing infrastructure. Despite such drastic expenditure, these models quickly become stale. This is because the data-distribution shifts or new data that gets generated as these model often drift (become stale) due to change in the data distribution which requires realignment. This necessity presents a quandary. Naive realignment leads to a phenomenon called catastrophic forgetting where the model overwrites prior information. On the other hand, a complete retraining of the model would require significant resource and this solution is extremely unattractive.

The more viable option is that of continual learning, where approaches seek to constrain the parameters of the model to reduce the amount of catastrophic forgetting the model experiences. Several works in the literature have attempted this direction, starting from early work on small multilayer perceptron McCloskey & Cohen (1989) to recent works on large language models such as Luo et al. (2025); Lai et al. (2025); Biderman et al. (2024) and Lin et al. (2025). One key commonality across all of these approaches is that, they look to constrain the weight/parameters of the AI model to induce minimal forgetting. On the other hand, in some cases and more appropriately, many approaches seek to balance between forgetting and the learning on new data (generalization) as in Raghavan & Balaprakash (2021); Lu et al. (2025) and others Lin et al. (2023). Despite demonstrated success, this is not the full story. In fact, under mild assumptions, it has been proven that simply changing the weight of the AI model is not sufficient to capture the drift in the data distribution and the capacity of the NN (its ability to represent tasks) eventually diverges if the data distribution keeps changing Chakraborty & Raghavan (2025). These intractability results from Chakraborty & Raghavan (2025) highlight a conundrum, that is, if the capacity of the network eventually diverges, then, should we still aim to continually learn AI models even though continually learning such models is invaluable? *In this paper, we demonstrate that the answer to this question is indeed "yes," and, in fact, the solution is to reliably change the architecture of the AI model on the fly according to the need of the data distribution.*

However, there are three key bottlenecks before attaining this goal: (i) reliably changing the architecture requires a methodical understanding of "how the weights and the architecture of the model interact with each other," that is, the coupling between the weights and the architecture; (ii) moreover, to decide when to change the architecture, one must understand, "how the forgetting-generalization trade-off is affected by the

aforementioned coupling'. Loosely speaking, the first two bottlenecks have been heuristically attempted in the literature. For example, some recent works such as CLEAS (Continual Learning with Efficient Architecture Search) Gao et al. (2022) and SEAL (Searching Expandable Architectures for Incremental Learning) Gambella et al. (2025) involves dynamically expanding the capacity of the AI model by adding layers and defining metrics that govern the necessity for a change in capacity. However, none of these approaches involve or enable a generic ability to change different aspects of the architecture, number of parameters, activation functions, layers, etc. The key reason is that, if the architecture is generically changed, one must initialize the new network architecture at random. (iii) Then, the transfer of information about the weights from the previous architecture to the new architecture across parameter spaces of two different shapes is required. However, this is currently impossible. In the absence of such a transfer mechanism, current approaches such as CLEAS and SEAL can only modify those components that would not warrant information transfer between parameter space of two different sizes. To address these bottlenecks, we will present a comprehensive approach that includes a novel formulation to understand the coupling between the architecture and the weights of the model, a methodical way of searching for a generic new architecture, and a novel method to transfer information between the old architecture and the new architecture.

1.1 Contribution

The key reason why the coupling between the architecture and the weights is difficult to model is because, architecture dependencies are observed on a function space across tasks and the weight dependencies are visible across the Euclidean parameter space. Any modeling in the weight space such as Chakraborty & Raghavan (2025); Raghavan & Balaprakash (2021) or Lu et al. (2025) cannot capture function space dependencies. To obviate this, we model the problem of continually learning the AI model over a sequence of tasks in a Sobolev space. Using this theoretical framework, we describe, how in the Sobolev space of AI models, parameterized by architecture choices and the weights, weights alone cannot capture the change in the distribution and the architecture must be changed. We then employ a derivative free architecture search approach to determine the new architecture by searching for an appropriate number of neurons. While in this work, we focus on changing the number of parameters in the AI model, our work is general enough to extend to other architecture choices as well. Once the new architecture is chosen, we develop a new algorithm that can efficiently transfer information from the previous architecture to a new architecture across parameter spaces of different sizes with minimal loss of performance on the previous tasks.

We then empirically demonstrate that changing the architecture indeed results in better training loss (at least % more improvement) over the case where the architecture is not changed. We also show that our algorithm achieves substantial improvements (upto %) in terms of training loss when training over large number of tasks; is robust to noise (Gaussian noise up to variance $\sigma =$) and scales from feedforward neural networks to graph neural networks seamlessly across regression and classification problems. We envision that jointly training the architecture and the weights in the continual learning paradigm is the best path to go forward in the field of continual learning and develop the basic theoretical and empirical infrastructure to allow for such training.

1.2 Organization

The paper is organized as follows, we begin by a description on continual learning and motivate the necessity for function space modeling, then, we describe a collection of neural networks as functions in a Sobolev space. We then describe the impact of familiar and unfamiliar tasks on continual learning and describe the necessity to change the architecture along with the weights of the AI model. Finally, we describe the algorithm to change the architecture on the fly and transfer information between the two architectures. We finally describe empirically the advantages of our approach.

1.3 Notations

These notation are adapted from Kolda & Bader (2009), please refer to the original paper for additional details. We use $\mathbb{N}$ to denote the set of natural numbers with $\mathbb{R}$ denoting the set of real numbers. An m^{th} order tensor is viewed as a multi-dimensional array contained in $\mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4 \times \cdots I_m}$, where the order can be

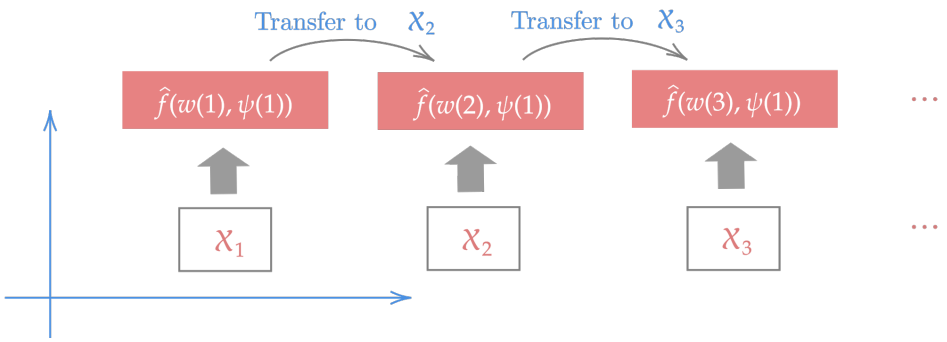

Figure 1: The basic problem of Continual Learning

thought of as the number of dimensions in the tensor. In this paper we will mostly be concerned with tensors of order $0, 1, 2$ and 3 which correspond to scalars, vectors, matrices, and a list of matrices. Therefore, we will write a tensor of order 0; a scalar, with lowercase letters, i.e., x; a tensor of order one is denoted by lowercase bold alphabets such as $\mathbf{x}$. A tensor of order 2 is a matrix denoted by uppercase bold alphabets $\mathbf{X}$ and any tensor of order greater than 2 are denoted by bolder Euler scripts letters such as $\boldsymbol{\mathcal{X}}$. We also use $\|.\|$ to denote the Euclidean norm for vectors, Frobenius norm for matrices and an appropriate tensor norm for tensors. Further, we will let $\langle \cdot, \cdot \rangle$ denote the dot product for vectors, matrices, or tensors. We will make one exception in our notation regarding the tensors that represent learnable/user defined parameters (architecture, step-size/learning rate, etc.), we will denote these with Greek letters. The i^{th} element of a vector $\mathbf{x}$ is denoted by $\mathrm{x}_{[i]}$, while the $(i,j)^{th}$ element of a matrix $\mathbf{X}$ is denoted by $\mathrm{x}_{[ij]}$. Moreover, we denote the i^{th} matrix in a tensor of order 3 by $\mathbf{X}_i$. We will make the indices run from 1 to their capital letters such that $i = 1, \ldots, I$.

2 Continual Learning – Motivation

The problem of continual learning is that of an AI model learning a sequence of tasks. Here, we define a task as a set of data available to learn from. Without going into the specifics of the underlying space of an AI model, we will denote the AI model as $\hat{f}(\mathbf{w}, \psi)$ where f denotes the composition of the weights$-\mathbf{w}$ and the architecture$-\psi$. We will define the particulars of these quantities in the later section; however, the key objective of this model is to learn a series of tasks. In this context, each task is represented by a data set obtained at a task instance, i.e. $t \in \boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{T}} = \{0, 1, \ldots, T\}$. We will assume that the dataset (a list of matrices/vectors/graphs) represented by $\boldsymbol{\mathcal{X}}(t)$ are provided at each task $t \in \boldsymbol{\mathcal{T}}$, where $\boldsymbol{\mathcal{X}}(t)$ is sampled according to the distribution $\mathbb{P}$ and $\boldsymbol{\mathcal{X}}(t) \subset \mathrm{D} \subset \mathbb{R}^n$ such that D – the domain, is a measurable set with a non-empty interior. Moreover, $(\mathrm{D}, \mathcal{B}(\mathrm{D}), \mathbb{P})$ forms the probability triplet with $\mathcal{B}(\mathrm{D})$ being the Borel sigma algebra over the domain D. The problem of interest to us is to understand the behavior of the AI model when it is learning on tasks $\boldsymbol{\mathcal{X}}(t)$ for every task instance t. Notably, tasks instances can belong to $\mathbb{N}$ and $\mathbb{R}$ depending on the application at hand, but the key goal is to understand how the tasks are assimilated by the AI model. The basic notion of continual learning is described in Fig. 1, where there are three tasks at $t = 1, 2$ and 3. For each of these three tasks, an architecture $\psi(1)$ is chosen and the three tasks are provided to the model in series. At $t = 1$, the model $\hat{f}$ learns from the first task. In particular, we solve the optimization problem

$$min_{\mathbf{w}(1)} \ell(\hat{f}(\mathbf{w}(1), \psi(1)), \boldsymbol{\mathcal{X}}(1)) \tag{1}$$

where ℓ is some loss function that summarizes the models performance on the task at $t = 1$. Then, at $t = 2$, the model transfers to learn information from the second task such that the first task is not forgotten. Implicitly, the optimization problem is rewritten as

$$min_{\mathbf{w}(2)} \left[\ell(\hat{f}(\mathbf{w}(2), \psi(1)), \boldsymbol{\mathcal{X}}(1)) + \ell(\hat{f}(\mathbf{w}(2), \psi(1)), \boldsymbol{\mathcal{X}}(2)) \right] \tag{2}$$

Similarly, at $t = 3$, the model transfers to learn information from the third task, but both tasks 1 and 2 must not be forgotten. This implies that

$$min_{\mathbf{w}(3)} \left[\ell(\hat{f}(\mathbf{w}(3), \psi(1)), \boldsymbol{\mathcal{X}}(1)) + \ell(\hat{f}(\mathbf{w}(3), \psi(1)), \boldsymbol{\mathcal{X}}(2)) + \ell(\hat{f}(\mathbf{w}(3), \psi(1)), \boldsymbol{\mathcal{X}}(3)) \right] \tag{3}$$

As evidenced the objective of the model is to remember all earlier tasks. Notably, this implies that the objective function of the continual learning problem is a sum that grows with every new task. Therefore, at any task t, the objective of the CL problem is

$$min_{\mathbf{w}(t)} \sum_{i=1}^{t} \left[\ell(\hat{f}(\mathbf{w}(i), \psi), \boldsymbol{\mathcal{X}}(i)) \right] \tag{Forgetting Loss}$$

where we have removed the index from ψ to indicate that the architecture is fixed. In the CL literature, (Forgetting Loss) is typically optimized at the onset of every new task at t. The most rudimentary approach for optimization of (Forgetting Loss) is conducted through an experience replay array that maintains a memory of all tasks in the past. To gain more insight into what happens at the onset of each new task, observe Figure 2.

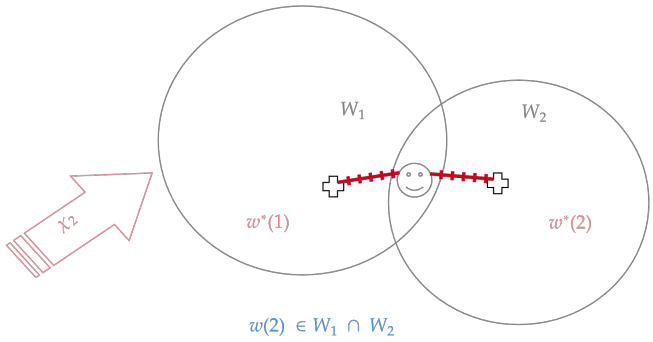

Figure 2: The basic problem of Continual Learning

For the sake of illustration, we assume that $\ell(\hat{f}(\mathbf{w}(i), \psi, \boldsymbol{\mathcal{X}}(i))$ is a strongly convex function and describe in Figure 2, the level set (the balls) in the parameter space corresponding to $\ell(\hat{f}(\mathbf{w}(i), \psi, \boldsymbol{\mathcal{X}}(i)) \leq \eta$ and the plus sign corresponds to the minima for $i = 1$ and $i = 2$. Notably, η is the threshold to identify acceptable loss values and defines the boundary of the level set and by extension, the radius of the ball in Figure 2. For instance, in the case of MNIST dataset, the two balls may represent two tasks of representing digit 0 and digit 1. The boundary of the left ball represents all parameters that provide cross entropy loss less than $\eta = 1$ for digit 0 with the plus sign representing zero cross-entropy loss. Similarly, the right ball represents all parameters that provide cross entropy loss less than $\eta = 1$ for digit 1 and the plus sign corresponds to zero cross entropy loss on digit 1. Within this example, consider an AI model (maybe a CNN) that trains on digit zero and achieves zero loss on digit zero, that is, attain the plus sign in the parameter space. Then, the AI model starts training for the next task- digit 1. First it starts from the plus sign achieved for digit 0, this implies that, there is an inherent bias due to the local minima achieved for digit 0. Second, when the new task is observed the model must converge to a solution that is common to both digit 0 and digit 1. This solution is trivial if the digit 0 and 1 were identical or very close to each other. However, as this is not the case, the solution must lie in the intersection between the two balls, that is, the region of the smiley face. Since the size of the intersection space depends on how different tasks 0 and 1 are, the following informally stated theorem is observed.

Theorem 1 *Fix the number of weight updates required to obtain the optimal value at each task. Assume that each subsequent task introduces a value of change into the model with the change described in some appropriate metric. Let ℓ be the Lipschitz continuous and choose a small learning rate. Then, capacity*

diverges as a function of number of tasks. ***Please see Chakraborty & Raghavan (2025) for a full statement and its proof***

The theorem implies that, the AI models' ability to successfully represent tasks will deteriorate as more tasks are introduced when the new tasks are different from the previous one. In such case, continually learning the AI model is bound to fail if we just update the weights of the network. In this paper, we hypothesize that size of the intersection space in Figure. 2 can be increased by modifying the architecture of the AI model, that is, by increasing the number parameters in the model. However, increasing the number of parameters would require us to model the coupling between the weights, data, and the architecture, which is impossible to imagine with the intuition laid out until now as this intuition corresponds to a Euclidean Allyson: what do you mean by this? Is it "Euclidean space" or am I misunderstanding? where the notion of architecture cannot be modeled. To this end, we will describe a neural network as a member of a class of functions in a function space which we choose to be a Sobolev space.

3 Neural Networks belong to a class of Sobolev space functions

The goal of this section is to formalize the mathematical modeling required to identify the dependency between the intersection space from Figure. 2, the architecture and the AI model. Towards this end, we first define the AI model $\hat{f}(\mathbf{w}, \psi)$ to belong to a class of functions contained in a Sobolev space with k bounded weak-derivatives. The notion of k bounded weak derivative is the key reason why Sobolev space is an appropriate choice for this problem, which we will discuss once we outline the formal definitions of Sobolev space aligning with definitions provided in Mahan et al. Therefore,

Definition 2 *Let $k \in \mathbb{N}$, the domain-$\mathrm{D} \subset \mathbb{R}^n$ a measurable set with non-empty interior, and $1 < p < \infty$. Then, the Sobolev space $W^{k,p}(\mathrm{D})$ consists of all functions f on D such that for all multi-indices α with $|\alpha| \leq k$, the mixed partial derivative $\partial^{(\alpha)} f$ exists in the weak sense and belongs to $L^p(\mathrm{D})$. That is,*

$$W^{k,p}(\mathrm{D}) = \{f \in L^p(\mathrm{D}) : \partial^{|\alpha|} h \in L^p(\mathrm{D}) \forall |\alpha| \leq k\}.$$

The number k is the order of the Sobolev space and the Sobolev space norm is defined as

$$\|f\|_{W^{k,p}(\mathrm{D})} := \sum_{|\alpha| \leq k} \|\partial^{|\alpha|} f\|_{L^p(\mathrm{D})}.$$

Allyson: Need to revisit first half of this paragraph. Heine-Borel gives compact if and only if closed and bounded. So, should we flip this part by saying it is only practical to assume we have a finite set which is automatically closed and bounded and hence compact? What is currently here, to me, sounds like "because it is a subset of $\mathbb{R}^n$ we can assume it is compact and closed by H-B," which is not true. Additionally, a "domain" is an open and connected subset. In this context, do we mean "domain" in the sense of "points which can be inputs" rather than the topological definition? If so, is this something the audience knows? In definition 2, two key notions must be highlighted. First, the D is by definition a subset of $\mathbb{R}^n$, which by virtue of the Heine-Borel Theorem Rudin (1976), is compact and closed. The implications of this assumption are important to consider. D, in this context is the data sample space and therefore, the assumption of compact and closedness applies to the data space. For any standard AI applications, this assumption implies that the data space is finite and the numerical values are bounded. Both are notable practical considerations that must be followed in any AI model training as it is well known that the without normalization, the AI model training fails to converge Huang et al. (2023). Therefore, this assumption is not only practical but necessary. The next assumption is presence of weak derivatives, notably, in the case of standard neural network, derivatives are assumed to exist as twice differentiability is necessary for training and typical training involves gradient based methods Kingma & Ba. In fact, the existence of weak derivatives can be summarized for different activation functions as in Table 1. As noted in the table, all standard activation functions can be recast in the purview of definition 2 and a Sobolev space can be constructed, that represents the class of functions that can be approximated by neural networks. The appropriateness of approximation and the condition over which such approximations are possible is discussed in Petersen et al. and Adegoke & Layeni.

Name	$\rho(\mathbf{x})$	Smoothness /Boundedness	Corresponsding $W^{k,p}$
Rectified Linear Unit (ReLU)	$max\{0, \mathbf{x}\}$	absolutely continuous, $\rho' \in L^p(\mathrm{D})$	$W^{0,p}(\mathrm{D})$
Exponential Linear Unit (eLU)	$\mathbf{x}.\chi_{\mathbf{x}\geq 0}$ $+(e^{\mathbf{x}}-1)\chi_{\mathbf{x}<0}$	$C^1(\mathbb{R})$, absolutely continuous $\rho'' \in L^p(\mathrm{D})$	$W^{1,p}(\mathrm{D})$
Softsign	$\frac{\mathbf{x}}{1+\lvert\mathbf{x}\rvert}$	$C^1(\mathbb{R})$, ρ' absolutely continuous, $\rho'' \in L^p(\mathrm{D})$	$W^{2,p}$
Inverse Square Root Linear Unit	$\mathbf{x}\chi_{\mathbf{x}\geq 0}$ $+\frac{\mathbf{x}}{\sqrt{1+a\mathbf{x}^2}}\chi_{\mathbf{x}<0}$	$C^2(\mathbb{R})$, ρ'' absolutely continuous, $\rho''' \in L^p(\mathrm{D})$	$W^{3,p}$
Inverse Square Root Unit	$\frac{\mathbf{x}}{\sqrt{1+a\mathbf{x}^2}}$	real analytic, real all derivatives bounded	$W^{k,p}\forall k$
Sigmoid	$\frac{1}{1+e^{\mathbf{x}}}$	real analytic, real all derivatives bounded	$W^{k,p}\forall k$
tanh	$\frac{e^{\mathbf{x}}-e^{-\mathbf{x}}}{e^{\mathbf{x}}+e^{-\mathbf{x}}}$	real analytic, real all derivatives bounded	$W^{k,p}\forall k$

Table 1: Different activation functions and the corresponding Sobolev spaces. Many of these results are described in the following references, Petersen et al. and Adegoke & Layeni for details. χ is an indicator function.

To represent the class of functions described in Definition 2, we will define a neural network as a function $\hat{f}(\mathbf{w}(t), \psi(t))$ with $d \in \mathbb{N}$ layers. In this notation of a neural network, $\mathbf{w}(t)$ is a long $\mathbb{R}^m$ vector that concatenates the weight parameters from the d layers, and $\psi(t)$ is comprised of the architecture and other user-defined parameters that are present in the neural network.

Definition 3 *A d layered neural network is given by an operator (essentially a function) $\hat{f}(\mathbf{w}(t), \psi(t)) \in W^{k,p}(\mathrm{D})$ with $W^{k,p}(\mathrm{D})$ being a Sobolev space. Furthermore,*

$$\hat{f}(\mathbf{w}(t), \psi(t))\big(.\big) = \hat{f}_d(\mathrm{w}_d(t), \psi_d(t)) \circ \hat{f}_{d-1}(\mathrm{w}_{d-1}(t), \psi_{d-1}(t)) \circ \cdots \circ \hat{f}_2(\mathrm{w}_2(t), \psi_2(t)) \circ \hat{f}_1(\mathrm{w}_1(t), \psi_1(t))\big(.\big) \quad (4)$$

describes the layer-wise compositions and $\big(.\big)$ represents the input tensor to which the operator is applied.

Remark 1 *Notation wise, we indicate approximate quantities with $\hat{}$ and true (target) quantities without the hat. For instance, the target function f in definition 2 is indicated without the hat and the approximate function $\hat{f}$ in definition 3 is indicated with a hat.*

Remark 2 *In definition 3, we describe the operation across different layers using the composition, notation wise, we hide this complexity through the definition of the operator. Due to this, definition 3 is generic and can be used to define feedforward neural networks, recurrent, convolutional, graph neural networks and even a large language model. Therefore, any analysis from this point is applicable to all types of neural networks.*

Typically, the user-defined parameters are a combination of integer, categorical, and floating point values and we assume that the architecture can be varied with task instances. Therefore, the parameters corresponding to the architecture ψ are assumed to be a function of the task instance t. As we cannot determine derivatives with respect to discrete parameters in the classical sense (i.e., derivative of $\psi(t)$), we will need to define weak derivative with respect to these quantities such that the derivatives (with respect to or of ψ) can be

approximated. To describe the learning problem, we define a notion of loss function on the space of functions defined in definition 3 and 2.

Definition 4 *Let $f(t) \in W^{k,p}(\mathrm{D})$ for all $t \in \{0, \ldots, T\}$ denote the target function which is to be approximated and let $\hat{f}(\mathbf{w}(t), \psi(t))$ denotes a neural network determined to approximate $f(t)$ at task t, then the loss function (or error of approximation) for $\mathbf{x} \in \boldsymbol{\mathcal{X}}(t) \subseteq \mathrm{D}$ is given by*

$$\ell(\hat{f}(\mathbf{w}(t), \psi(t)), \mathbf{x}) = \|\hat{f}(\mathbf{w}(t), \psi(t))(\mathbf{x}) - f(t)(\mathbf{x})\|_{W^{k,p}(\mathrm{D})}.$$

The function f in the context of ML is an ideal forecasting model that can forecast temperature perfectly or an ideal classification model that performs some classification problem.

Remark 3 *In applications, we will assume* D *is compact to guarantee the target function $f(t)$ exists in a Sobolev space $W^{k,p}(\mathrm{D})$, by Corollary 3.5. in Mahan et al.. However, in the event compactness cannot be assumed, we find the minimum norm solution to the problem in the Sobolev space.*

The loss function in definition 4 is defined over $W^{k,p}(\mathrm{D})$ using the Sobolev space norm. However, in practice, we can directly utilize samples corresponding to f or $\hat{f}$ to construct the loss function from samples. For instance, this has been done in Son et al. (2021) in the context of physics informed neural network - a class of neural networks that are modeled by definition 3. In particular, the sobolev space norm is defined as in definition 2, where the loss function is a sum over L^p norm over all k weak derivative. As a trivial case, choosing $k = 0$ and $p = 2$ gives the standard mean squared error loss function which according to Table 1 and practical consideration is applicable to variety of activation functions used in practical AI.

3.1 Continual Learning in $W^{k,p}(\mathrm{D})$

Extending definition 4 to the case of continual learning will provide us with the objective to optimize. As described in 2, the goal of CL is to learn over a series of tasks such that the prior tasks are not forgotten. In particular, for each task instance t, we must find $\mathbf{w}$ that is optimal with respect to an objective function J defined over the interval $[0, t]$. This objective function is known as the forgetting loss and was informally defined in (Forgetting Loss). Specifically, we rewrite (Forgetting Loss) in terms of data, weights, and architecture, as below

$$J(\mathbf{w}(t), \psi(t), \boldsymbol{\mathcal{X}}(t)) = \int_0^t \left(\int_{\mathbf{x} \in \boldsymbol{\mathcal{X}}(\tau)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\mathbf{x})) \right) \, d\,\tau$$

A traditional learning structure seeks to minimize the cost $J(t)$ through an iterative procedure that drives $J(t) \to 0$ as $t \to \infty$ iff $\boldsymbol{\mathcal{X}}(t_1) = \boldsymbol{\mathcal{X}}(t_2), \forall\ t_1, t_2 \in [0, T]$. This learning structure is also mathematically guaranteed to converge with standard optimizers such as Adam Kingma & Ba. However, since $\boldsymbol{\mathcal{X}}(t_1) \neq \boldsymbol{\mathcal{X}}(t_2), \forall\ t_1, t_2 \in [0, T]$, we encounter the issue described in Figure 2. As described in section 2, there exists an inherent bias due to the order in which the tasks are observed and every time a new task is obtained, the underlying optimization problem needs to be resolved. In particular, we obtain a series of optimization problems corresponding to a series of tasks which leads to a dynamic program Bertsekas (2012) over the target functions for each task. Typically, such dynamic programs Bertsekas (2012) are solved by writing a cumulative optimization problem over the interval $[0, T]$ as in

$$V^*(t, \mathbf{w}(t)) = \min_{\mathbf{w} \in \mathcal{W}(\psi^*(t))} V(t, \mathbf{w}(t)),$$

where $\mathcal{W}(\psi^*(t))$ is the weights search space, and $V(t, \mathbf{w}(t))$ is given below

$$V(t, \mathbf{w}(t)) = \int_t^T J(\mathbf{w}(\tau), \psi^*(t), \boldsymbol{\mathcal{X}}(\tau)).$$

A typical learning process is provided in Raghavan & Balaprakash (2021); Chakraborty & Raghavan (2025) for these approaches involve extracting batches of data from $\boldsymbol{\mathcal{X}}(t)$ and then updating the network based

on these batches of data. All of these updates are performed with a fixed architecture $\psi^*(t)$. However, we are instead interested in the scenario where the architecture and the model parameters are both learned. In order to learn the optimal weights and architecture for our network for all tasks, we must complete a bilevel optimization. First, we determine the optimal architecture, denoted $\psi^*(t)$ for the tth task. This optimization is completed below

$$\psi^*(t) = \arg\min_{\psi \in \Psi} J(\mathbf{w}(t), \psi, \boldsymbol{\mathcal{X}}(t)), \tag{Architecture search}$$

where Ψ is our architecture search space. Once completed, we move to the lower level optimization of the weights by utilizing dynamic programming. We determine

$$V^*(t, \mathbf{w}(t)) = \min_{\mathbf{w} \in \mathcal{W}(\psi^*(t))} V(t, \mathbf{w}(t)), \tag{Cumulative CL}$$

This provides a holistic view of a learning problem for all tasks which incorporates the impact of architecture and weights. As discussed in the introduction, naively solving this problem will involve repeatedly retraining the model from scratch for each new task. This is both computationally and mathematically unattractive. Therefore, we seek to find an elegant solution to this problem and to do this, we must understand the effect of subsequent tasks on the CL learning problem and by extension the effect of architecture on the CL problem.

4 Understanding the existence of the CL solution

For illustration, let us reconsider the example from section 2. In this regard, we draw Figure 3 where three tasks are shown to the model in order and the architecture of the network is fixed as $\psi(1)$. The goal of the CL is to solve the problem in (Cumulative CL) at each task t such that optimal weights can be determined. However, determining the optimal weights is highly dependent on each task's data. Intuitively, if the data in task $\boldsymbol{\mathcal{X}}(t) = \boldsymbol{\mathcal{X}}(t+1), \forall\, t \in [0, T]$, we can find a NN $\hat{f}$ which learns the new task $\boldsymbol{\mathcal{X}}(t+1)$ and transfers learning from $\boldsymbol{\mathcal{X}}(t)$ perfectly. But, if $\boldsymbol{\mathcal{X}}(t) \neq \boldsymbol{\mathcal{X}}(t+1), \forall\, t \in [0, T]$, we may or may not be able to find a common solution.

To illustrate this further, consider Figure 3 and let $\mathcal{F}_t = \{\hat{f}(\mathbf{w}, \psi) \in W^{k,p}(\mathrm{D})\}$ to represent the search space for the set of all possible NN solutions for learning task t (refer to 2 for a similar construction). As $\mathcal{F}_t$ is a subset of $W^{k,p}(\mathrm{D})$, the balls in Figure 3 can be thought of as a representative of a level set such that the boundary of the size of the ball represents the set of all solutions that provide the loss value less than a threshold. At $t = 1$, there is one ball, at task $t = 2$ there are two balls, and at task $t = 3$ there are three balls. The number of balls here correspond to the number of tasks being involved in the learning process at each step of learning.

In each of the balls, ✚ represents the NN solution for only that particular task t and the smileys face represents the solution that transfers learning amongst tasks between 0 to t. Notably, ☺ must lie in the intersection of all the balls, as solution common to tasks 0 to t are only available in this intersection. Curiously, unless the tasks as equivalent, the ☺ and the ✚ will not coincide because no solution that is perfect for one task ends up being perfect for another. Furthermore, the more different the subsequent tasks are, it is intuitive that the intersection space is unlikely to be non-empty. Therefore, as long as the model can ensure that the intersection space is non-empty, the existence of a common CL solution will be guaranteed. Therefore, the necessary conditions for the existence of the CL solution is tied to the size of this intersection space. Furthermore, the quality of the solution is tied to the value of the continual learning cost at the introduction of each new task. In particular, if the existence of the common solution at the introduction of any new task implies that the value of $\nabla_t J(\mathbf{w}(\tau), \psi^*(t), \boldsymbol{\mathcal{X}}(\tau))$ is small. Mathematically, this intuition is implied by the continuity of $J(\mathbf{w}(\tau), \psi^*(t), \boldsymbol{\mathcal{X}}(\tau))$ in the closed interval $[t, t+1]$.

To formalize this intuition and demonstrate the impact of architecture on the continuity of J, we first define the notion of the intersection space in $W^{k,p}(\mathrm{D})$ in terms of its impact on J. Towards this end, we elucidate an ϵ, δ defintion of continuity of a function w.r.t measure as

Definition 5 *Let $\overline{\boldsymbol{\mathcal{X}}} = \bigcup_{t=0}^{\infty} \boldsymbol{\mathcal{X}}(t)$ with the power set $\mathcal{P}(\overline{\boldsymbol{\mathcal{X}}})$ as its topology. Set $\mathcal{B}(\overline{\boldsymbol{\mathcal{X}}})$ to be the Borel sigma algebra on $\overline{\boldsymbol{\mathcal{X}}}$ equipped with a probability measure denoted μ. Then, $(\overline{\boldsymbol{\mathcal{X}}}, \mathcal{B}(\overline{\boldsymbol{\mathcal{X}}}), \mu)$ forms a probability measure*

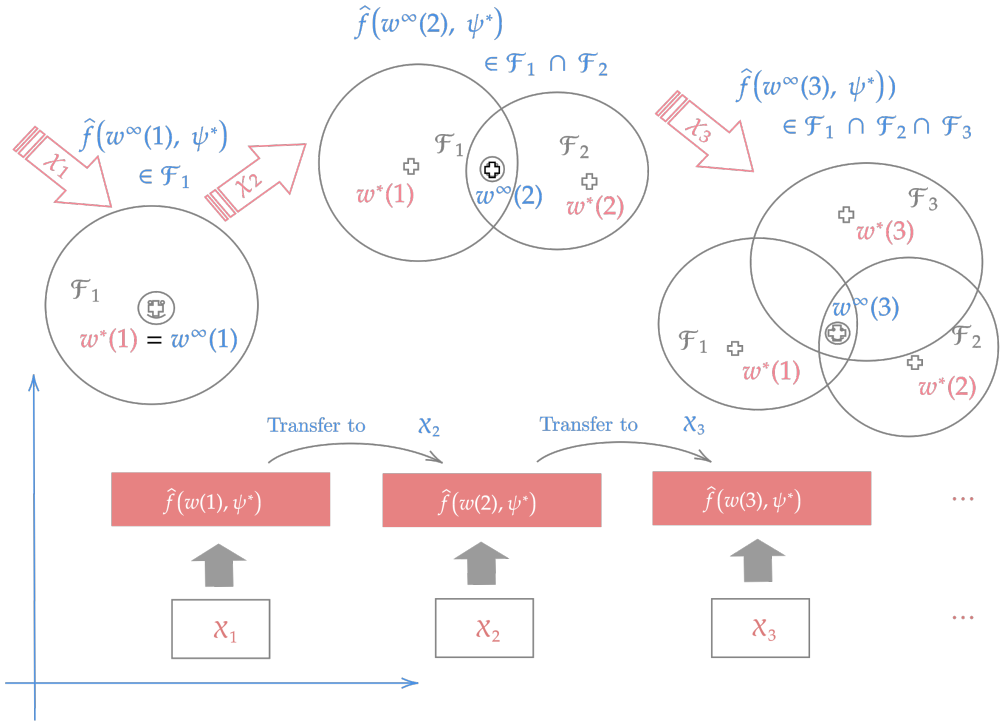

Figure 3: The actual problem

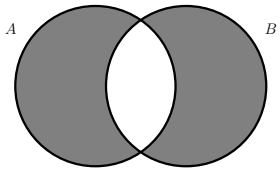

Figure 4: The gray shaded region represents set symmetric difference $A \triangle B$.

space. Let $\hat{f} : \overline{\boldsymbol{\mathcal{X}}} \to \mathbb{R}$ *be a measurable function, and set* $F : \mathcal{B}(\overline{\boldsymbol{\mathcal{X}}}) \to \mathbb{R}$ *to be the function*

$$F(A) = \int_{\mathbf{x} \in A} \hat{f}(\mathbf{x}) d\mu,$$

where $\hat{f}(\mathbf{w}(t), \psi(t))(\mathbf{x}) = \hat{f}(\mathbf{x})$ *Then, we say* F *is continuous with respect to the measure if for every* $\varepsilon > 0$*, there exists* $\delta > 0$ *such that* $|F(A) - F(B)| < \varepsilon$ *whenever* $A, B \in \mathcal{B}(\overline{\boldsymbol{\mathcal{X}}})$ *such that* $\mu(A \triangle B) < \delta$*.*

Definition 5 can be deduced from basic measure theory results found in Weaver (2013). The intuition behind this definition is that sets which are similar (with respect to the probability measure μ) result in integrals of $\hat{f}$ over the respective sets which are close in value. In the set theory context, the notation $A \triangle B$ represents set symmetric difference, which describes the elements two sets do not share (see Figure 4 for an illustration). Thus, as $\mu(A \triangle B)$ gets larger, the size of the intersection space (i.e. $\mu(A \cap B)$) shrinks. Hence, the measure of the set symmetric difference provides a notion of how "similar" or how "different" two sets are. In other words, we can acquire a notion of how similar or different two tasks are.

We now describe the continual learning cost $J(\mathbf{w}(t), \psi(t), \boldsymbol{\mathcal{X}}(t))$ in the context of definition 5. In particular, the expected value of $J(\mathbf{w}(t), \psi(t), \boldsymbol{\mathcal{X}}(t))$ is continuous with respect to measure μ if for a small $\varepsilon > 0$ we can find a $\delta > 0$ such that $|J(\mathbf{w}(t), \psi(t), \boldsymbol{\mathcal{X}}(t)) - J(\mathbf{w}(t+1), \psi(t+1), \boldsymbol{\mathcal{X}}(t+1))| < \varepsilon$ for all $\boldsymbol{\mathcal{X}}(t+1)$ such that $\mu(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t+1)) < \delta$. Thus, if two tasks $\boldsymbol{\mathcal{X}}(t)$ and $\boldsymbol{\mathcal{X}}(t + \Delta t)$ are similar (i.e. $\mu(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t + \Delta t)) < \delta$), then the forgetting cost J are similar (i.e. $|J(\mathbf{w}(t), \psi(t), \boldsymbol{\mathcal{X}}(t)) - J(\mathbf{w}(t+1), \psi(t+1), \boldsymbol{\mathcal{X}}(t+1))| < \varepsilon$). This

intuition is formalized in the following theorem, where we prove that J is continuous with respect to measure μ.

Theorem 6 *Let $(\overline{\boldsymbol{\mathcal{X}}}, \mathcal{B}(\overline{\boldsymbol{\mathcal{X}}}), \mu)$ be the measure space as defined as in definition 5. Assume that the loss function ℓ is continuous and bounded across all tasks, that is $\forall t \in [0, T]$. Define J over a task $\boldsymbol{\mathcal{X}}(\tau)$ as*

$$J(\mathbf{w}(t), \psi(t), \boldsymbol{\mathcal{X}}(t)) = \int_0^t \left(\int_{\mathbf{x} \in \boldsymbol{\mathcal{X}}(\tau)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\mathbf{x})) \right) \, d\ \tau \tag{5}$$

Then, $J(\mathbf{w}(t), \psi(t), \boldsymbol{\mathcal{X}}(t))$ is continuous with respect to the measure for any $\boldsymbol{\mathcal{X}}(t), \boldsymbol{\mathcal{X}}(t+1) \in \mathcal{B}(\overline{\boldsymbol{\mathcal{X}}})$ such that $\mu(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t+1)) < \delta \implies |J(\mathbf{w}(t+1), \psi(t+1), \boldsymbol{\mathcal{X}}(t+1)) - J(\mathbf{w}(t), \psi(t), \boldsymbol{\mathcal{X}}(t))| \leq \epsilon$

Proof 1 *Suppose that the constant $M_0 > 0$ bounds ℓ on every task. Let $\varepsilon > 0$ and set $\delta = \varepsilon / M_0$. Further let $A, B \in \mathcal{B}(\overline{\boldsymbol{\mathcal{X}}})$ such that $\mu(A \triangle B) < \delta$. By disjoint additivity of μ and triangle inequality assuming that the subinterval $[t, t + \Delta t]$ is divided into exactly one unit such that $\Delta t = 1$ we have*

$$\begin{aligned}
&\left| \int_0^{t+\Delta\ t} \left(\int_{\mathbf{x} \in \boldsymbol{\mathcal{X}}(\tau)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\mathbf{x})) \right) \, d\ \tau - \int_0^t \left(\int_{\mathbf{x} \in \boldsymbol{\mathcal{X}}(\tau)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\mathbf{x})) \right) \, d\ \tau \right| \\
&= \Big| \int_t^{t+\Delta t} \left(\int_{\mathbf{x} \in \boldsymbol{\mathcal{X}}(\tau)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\mathbf{x})) \right) \, d\ \tau \Big| \\
&= \left| \int_{\mathbf{x} \in \boldsymbol{\mathcal{X}}(t+1)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\ vx)) d\mu - \int_{\mathbf{x} \in \boldsymbol{\mathcal{X}}(t)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\ vx)) d\mu \right| \\
&= \Bigg| \int_{\boldsymbol{\mathcal{X}}(t) \setminus \boldsymbol{\mathcal{X}}(t+1)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\ vx)) d\mu \\
&+ \int_{\boldsymbol{\mathcal{X}}(t) \cap \boldsymbol{\mathcal{X}}(t+1)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\ vx)) d\mu \\
&- \int_{\boldsymbol{\mathcal{X}}(t+1) \setminus \boldsymbol{\mathcal{X}}(t)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\ vx)) d\mu \\
&- \int_{\boldsymbol{\mathcal{X}}(t+1) \cap \boldsymbol{\mathcal{X}}(t)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\ vx)) d\mu \Bigg| \\
&\leq \int_{\boldsymbol{\mathcal{X}}(t+1) \setminus \boldsymbol{\mathcal{X}}(t)} |\ell(\hat{f}(\mathbf{w}(t), \psi(t))(\ vx))| d\mu \\
&+ \int_{\boldsymbol{\mathcal{X}}(t) \setminus \boldsymbol{\mathcal{X}}(t+1)} |\ell(\hat{f}(\mathbf{w}(t), \psi(t))(\ vx))| d\mu.
\end{aligned}$$

Then, by the boundedness of ℓ,

$$\begin{aligned}
\int_{\boldsymbol{\mathcal{X}}(t+1) \setminus \boldsymbol{\mathcal{X}}(t)} |\ell(\hat{f}(\mathbf{w}(t), \psi(t))(\ vx))| d\mu + \int_{\boldsymbol{\mathcal{X}}(t) \setminus \boldsymbol{\mathcal{X}}(t+1)} |\ell(\hat{f}(\mathbf{w}(t), \psi(t))(\ vx))| d\mu \\
\leq M_0 \mu(\boldsymbol{\mathcal{X}}(t) \setminus \boldsymbol{\mathcal{X}}(t+1)) + M_0 \mu(\boldsymbol{\mathcal{X}}(t+1) \setminus \boldsymbol{\mathcal{X}}(t)).
\end{aligned}$$

Hence,

$$\begin{aligned}
M_0 \mu(\boldsymbol{\mathcal{X}}(t) \setminus \boldsymbol{\mathcal{X}}(t+1)) + M_0 \mu(\boldsymbol{\mathcal{X}}(t+1) \setminus \boldsymbol{\mathcal{X}}(t)) &= M_0 (\mu(\boldsymbol{\mathcal{X}}(t) \setminus \boldsymbol{\mathcal{X}}(t+1)) + \mu(\boldsymbol{\mathcal{X}}(t+1) \setminus \boldsymbol{\mathcal{X}}(t))) \\
&= M_0 \mu(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t+1)) \\
&< M_0 \delta \\
&= M_0 \frac{\varepsilon}{M_0} \\
&= \varepsilon.
\end{aligned}$$

which gives the result.

Remark 4 *It is reasonable to assume this hat there is some c onstant $M_0 < \infty$ for which $M_i \leq M_0$ for all $i \in \mathbb{N}$ because if one did not exist then the problem would be unsolvable as an unbounded loss function would imply that the neural network is not learning anything useful from the data.*

Theorem 6 solidifies the notion that similar tasks produce similar loss values and the idea that, continuity of J is upheld when the tasks are similar. In return, this implies that the intersection search space in Figure 4 and **??** is non-empty. The formal connection of this idea to the intersection space and continuity of fogetting cost is proven for the first time in this work. Moreover, the contrapositive reveals that dissimilar loss values implies dissimilar tasks. However, for an arbitrary CL problem, dissimilar tasks may or may not result in dissimilar loss values as this depends on the weight training mechanism. This intuition implies that if we have two tasks $\boldsymbol{\mathcal{X}}(t)$ and $\boldsymbol{\mathcal{X}}(t+\Delta t)$ such that $\mu\left(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t+\Delta t)\right) \geq \delta$ then, we do not know whether $|J(\boldsymbol{\mathcal{X}}(t)) - J(\boldsymbol{\mathcal{X}}(t+\Delta t))| < \varepsilon$ (i.e, similar loss values) or $|J(\boldsymbol{\mathcal{X}}(t)) - J(\boldsymbol{\mathcal{X}}(t+\Delta t))| \geq \varepsilon$ (i.e. dissimilar loss values). Nonetheless, it is intuitively clear that, if dissimilar tasks do produce dissimilar loss values, eventually, the intersection space becomes empty and the continuity of J is violated. In practice, however, a sudden violation of Theorem 6 is unlikely to occur. Instead, as the number of tasks increase, the value of $|J(\boldsymbol{\mathcal{X}}(t)) - J(\boldsymbol{\mathcal{X}}(t+\Delta t))|$ will tend to increase and eventually exceed ε. Therefore, the model will gradually forget prior tasks and eventually CL will fail. Where we show that this gradual forgetting is observed when the architecture is fixed across all tasks. Krishnan: reached here with the revision towards making the existence argument more rigorous through continuity argument.

Lemma 7 *Suppose $\boldsymbol{\mathcal{X}}(t)$ and $\boldsymbol{\mathcal{X}}(t+1)$ are two consecutive tasks in the measure space $(\overline{\boldsymbol{\mathcal{X}}}, \mathcal{B}(\overline{\boldsymbol{\mathcal{X}}}), \mu)$. Set M_0 to be the upper bound on the loss function ℓ across all tasks and suppose $\psi(t) = \psi(1)$ for all tasks t. Then, for $\mu(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t+1)) \geq \delta$, the following holds*

$$E(\boldsymbol{\mathcal{X}}(t+\Delta t)) = E(\boldsymbol{\mathcal{X}}(t)) + \Delta t \Big[\mu(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)) \cdot \int_{\boldsymbol{\mathcal{X}}(t)\Delta\boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w}, \psi)) d\mu + \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w}, \psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w}, \psi) \cdot \Delta \mathbf{w} \, d\mu \Big] + o(\Delta t).$$

Proof 2 *The first-order Taylor series expansion of $E(\boldsymbol{\mathcal{X}}(t+\Delta t))$ about t, is given by*

$$E(\boldsymbol{\mathcal{X}}(t+\Delta t)) = E(\boldsymbol{\mathcal{X}}(t)) + \Delta t \Big[E_{\boldsymbol{\mathcal{X}}}(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)) + E_{\mathbf{w}}(\boldsymbol{\mathcal{X}}(t)) \Big] + o(\Delta t), \tag{6}$$

where $E_{\boldsymbol{\mathcal{X}}}$ and $E_{\mathbf{w}}$, are the partial derivatives of the expected value function with respect to the data and weights, respectively. Since we assumed $\psi(t) = \psi(1)$ for all t (i.e., the architecture is fixed), note that we did not take the partial derivative of the expected value function with respect to the architecture.

Now, we work on acquiring each partial derivative. Toward that end,

$$E_{\boldsymbol{\mathcal{X}}}(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)) = \mu(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)) \cdot \int_{\boldsymbol{\mathcal{X}}(t)\Delta\boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w}, \psi)) d\mu. \tag{7}$$

To determine the remaining two derivatives, we use the Sobolev function chain rule found in Evans (2022). We can do so as ℓ is real-valued and bounded, and ℓ' is continuously differentiable. Then,

$$E_{\mathbf{w}}(\boldsymbol{\mathcal{X}}(t)) = \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w}, \psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w}, \psi) \cdot \Delta w \, d\mu. \tag{8}$$

*Substituting (7) and (8) into the Taylor series expansion (**??**), we have*

$$E(\boldsymbol{\mathcal{X}}(t+\Delta t)) = E(\boldsymbol{\mathcal{X}}(t)) + \Delta t \Big[\mu(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)) \cdot \int_{\boldsymbol{\mathcal{X}}(t)\Delta\boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w}, \psi)) d\mu + \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w}, \psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w}, \psi) \cdot \Delta w \, d\mu \Big] + o(\Delta t),$$

as desired.

The following theorem is immediate to identify the lower bound on the first difference in the CL cost. Krishnan: reached here

Theorem 8 *Suppose $\boldsymbol{\mathcal{X}}(t)$ and $\boldsymbol{\mathcal{X}}(t+1)$ are two consecutive tasks in the measure space $(\overline{\boldsymbol{\mathcal{X}}}, \mathcal{B}(\overline{\boldsymbol{\mathcal{X}}}), \mu)$. Let $L_1 > 0$ and $0 \leq \delta \leq 1$ all be constants. Set M_0 to be the upper bound on the loss function ℓ across all tasks. Moreover, suppose $\psi(t) = \psi(1)$ for all tasks t. Then, for $\mu(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t+1)) \geq \delta$, the following holds*

$$E(\boldsymbol{\mathcal{X}}(T)) - E(\boldsymbol{\mathcal{X}}(t)) \leq \sum_{\tau=t}^{T} \left(M_0 \cdot \delta - \int_{\boldsymbol{\mathcal{X}}(\tau)} \left(\ell'(\hat{f}(\mathbf{w}, \psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w}, \psi) \right)^2 \, d\mu \right)$$

Proof 3 *To begin, recall that*

$$\begin{aligned} J(\mathbf{w}(t), \psi(t), \boldsymbol{\mathcal{X}}(t)) &= \int_0^t \left(\int_{\mathbf{x} \in \boldsymbol{\mathcal{X}}(\tau)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\mathbf{x})) \right) \, d\, \tau \\ &= \int_0^t E(\boldsymbol{\mathcal{X}}(\tau)) d\tau. \end{aligned}$$

As the expected value function is a continuous, real-valued function on the closed interval $[0, t]$, it follows that

$$\nabla J = \nabla \left(\int_0^t E(\boldsymbol{\mathcal{X}}(\tau)) d\tau \right) = \int_0^t \nabla E(\boldsymbol{\mathcal{X}}(\tau)) d\tau = E(\boldsymbol{\mathcal{X}}(t)).$$

By Lemma 7 and the previous equation

$$\begin{aligned} \nabla J &= E(\boldsymbol{\mathcal{X}}(t)) \\ &= E(\boldsymbol{\mathcal{X}}(t+\Delta t)) - \Delta t \Big[\mu(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)) \cdot \int_{\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w}, \psi)) d\mu \\ &+ \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w}, \psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w}, \psi) \cdot \Delta w \, d\mu \Big] - o(\Delta t) \\ &= E(\boldsymbol{\mathcal{X}}(t+\Delta t)) + \Delta t (-\mu(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t))) \cdot \int_{\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w}, \psi)) d\mu \\ &- \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w}, \psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w}, \psi) \cdot \Delta w \, d\mu - o(\Delta t) \\ &\leq E(\boldsymbol{\mathcal{X}}(t+\Delta t)) + \Delta t \cdot \delta \cdot \int_{\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w}, \psi)) d\mu \\ &- \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w}, \psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w}, \psi) \cdot \Delta w \, d\mu - o(\Delta t) \end{aligned}$$

Subtracting $E(\boldsymbol{\mathcal{X}}(t+\Delta t))$ from both sides and combining like terms,

$$\begin{aligned} E(\boldsymbol{\mathcal{X}}(t)) - E(\boldsymbol{\mathcal{X}}(t+\Delta t)) &\leq E(\boldsymbol{\mathcal{X}}(t+\Delta t)) + \Delta t \cdot \delta \cdot \int_{\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w}, \psi)) d\mu \\ &- \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w}, \psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w}, \psi) \cdot \Delta w \, d\mu - o(\Delta t) - E(\boldsymbol{\mathcal{X}}(t+\Delta t)) \\ &\leq \Delta t \cdot \delta \cdot \int_{\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w}, \psi)) d\mu \\ &- \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w}, \psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w}, \psi) \cdot \Delta w \, d\mu - o(\Delta t). \end{aligned}$$

Additionally, we can assume that the loss function ℓ is bounded above by a constant M_0, so

$$E(\boldsymbol{\mathcal{X}}(t)) - E(\boldsymbol{\mathcal{X}}(t+\Delta t)) \leq \Delta t \cdot \delta M_0 - \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \cdot \Delta w \, d\mu - o(\Delta t).$$

To consider this difference in expected values across future tasks, set $\Delta t = 1$ and let us sum across such future tasks,

$$\sum_{\tau=t}^{T} (E(\boldsymbol{\mathcal{X}}(\tau)) - E(\boldsymbol{\mathcal{X}}(\tau+1))) \leq \sum_{\tau=t}^{T} \Big(M_0 \cdot \delta - \int_{\boldsymbol{\mathcal{X}}(\tau)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \cdot \Delta w \, d\mu \Big).$$

Now, we can choose the following for each τ

$$\Delta \mathbf{w} = \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi).$$

Thus,

$$\sum_{\tau=t}^{T} (E(\boldsymbol{\mathcal{X}}(\tau)) - E(\boldsymbol{\mathcal{X}}(\tau+1))) \leq \sum_{\tau=t}^{T} \Big(M_0 \cdot \delta - \int_{\boldsymbol{\mathcal{X}}(\tau)} \Big(\ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \Big)^2 \, d\mu \Big),$$

as desired. Note that the the term $\Big(\ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \Big)^2$ above is positive, and so the integral will be positive. Thus, we can view the subtraction of this integral as the impact of changing the weights to the overall difference in loss values.

The lower bound in (Theorem **??**) confirms that the norm of the gradient is dependent upon the bound on the loss function and the change in the weights. In particular, we see that the more we change the weights $\mathbf{w}(t)$, the smaller our lower bound gets. As a smaller norm of the gradient implies a smaller change in the expected value between two tasks (i.e. smaller $|E(\boldsymbol{\mathcal{X}}(t))) - E(\boldsymbol{\mathcal{X}}(t+\Delta t))|$), it is clear that we could reduce the lower bound in (**??**) by changing the weights. However, we can only change the weights so far before overfitting occurs. In other words, there is a bound on the change in weights, and hence a bound on much we can reduce the lower bound (Theorem **??**). It would be ideal if the lower bound could be reduced further.

It is important to note that in Theorem **??**, we fix the architecture. What if we allow for changing the architecture as well? Intuitively, we know changing the architecture via some architecture search will result in a better performing NN (i.e. smaller expected value). However, let us provide a concrete proof to support this intuition. The following theorem provides us with a lower bound on $\|\nabla E\|$ that is smaller than that presented in (**??**). Thus, we have the potential to reduce $\|\nabla E\|$ and hence reduce the change in the expected value $|E(\boldsymbol{\mathcal{X}}(t))) - E(\boldsymbol{\mathcal{X}}(t+\Delta t))|$.

Theorem 9 *Suppose $\boldsymbol{\mathcal{X}}(t)$ and $\boldsymbol{\mathcal{X}}(t+1)$ are two consecutive tasks in the measure space $(\overline{\boldsymbol{\mathcal{X}}}, \mathcal{B}(\overline{\boldsymbol{\mathcal{X}}}), \mu)$. Let $L_1 > 0$ and $0 \leq \delta \leq 1$ all be constants. Set M_0 to be the upperbound on the loss function ℓ across all tasks. Then, for $\mu(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t+1)) \geq \delta$, the following holds*

$$\sum_{\tau=t}^{T} (E(\boldsymbol{\mathcal{X}}(\tau)) - E(\boldsymbol{\mathcal{X}}(\tau+1))) \leq \sum_{\tau=t}^{T} \Big(M_0 \cdot \delta - \int_{\boldsymbol{\mathcal{X}}(\tau)} \Big(\ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \Big)^2 \, d\mu - \\ - \int_{\boldsymbol{\mathcal{X}}(t)} \Big(\ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \Big)^2 \, d\mu \Big).$$

Proof 4 *To begin, note that the first-order Taylor series expansion of $E(\boldsymbol{\mathcal{X}}(t+\Delta t))$ about t, is given by*

$$E(\boldsymbol{\mathcal{X}}(t+\Delta t)) = E(\boldsymbol{\mathcal{X}}(t)) + \Delta t \Big[E_{\boldsymbol{\mathcal{X}}}(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)) + E_{\mathbf{w}}(\boldsymbol{\mathcal{X}}(t)) + E_{\psi}(\boldsymbol{\mathcal{X}}(t)) \Big] + o(\Delta t), \tag{9}$$

where $E_{\boldsymbol{\mathcal{X}}}, E_{\mathbf{w}}$, and E_{ψ} are the partial derivatives of the expected value function with respect to the data, weights, and architecture, respectively. Now, we work on acquiring each partial derivative. Toward that end,

$$E_{\boldsymbol{\mathcal{X}}}(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)) = \mu(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)) \cdot \int_{\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w},\psi)) d\mu. \tag{10}$$

To determine the remaining two derivatives, we use the Sobolev function chain rule found in Evans (2022). We can do so as ℓ is real-valued and bounded, and ℓ' is continuously differentiable. Then,

$$E_{\mathbf{w}}(\boldsymbol{\mathcal{X}}(t)) = \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \cdot \Delta w \; d\mu., \tag{11}$$

$$E_{\psi}(\boldsymbol{\mathcal{X}}(t)) = \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \cdot \Delta \psi \; d\mu. \tag{12}$$

Substituting (10), (11), and (12) into the Taylor series expansion (9), we have

$$E(\boldsymbol{\mathcal{X}}(t+\Delta t)) = E(\boldsymbol{\mathcal{X}}(t)) + \Delta t \Big[\mu(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)) \cdot \int_{\boldsymbol{\mathcal{X}}(t)\Delta\boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w},\psi)) d\mu \tag{13}$$

$$+ \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \cdot \Delta w \; d\mu + \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \cdot \Delta \psi \; d\mu \Big] + o(\Delta t). \tag{14}$$

recall that

$$\begin{aligned} J(\mathbf{w}(t), \psi(t), \boldsymbol{\mathcal{X}}(t)) &= \int_0^t \left(\int_{\mathbf{x} \in \boldsymbol{\mathcal{X}}(\tau)} \ell(\hat{f}(\mathbf{w}(t), \psi(t))(\mathbf{x})) \right) \; d\; \tau \\ &= \int_0^t E(\boldsymbol{\mathcal{X}}(\tau)) d\tau. \end{aligned}$$

As the expected value function is a continuous, real-valued function on the closed interval $[0, t]$, it follows that

$$\nabla J = \nabla \left(\int_0^t E(\boldsymbol{\mathcal{X}}(\tau)) d\tau \right) = \int_0^t \nabla E(\boldsymbol{\mathcal{X}}(\tau)) d\tau = E(\boldsymbol{\mathcal{X}}(t)).$$

By (14) and the previous equation

$$\begin{aligned} E(\boldsymbol{\mathcal{X}}(t)) &= \nabla J \\ &= E(\boldsymbol{\mathcal{X}}(t+\Delta t)) - \Delta t \Big[\mu(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t)) \cdot \int_{\boldsymbol{\mathcal{X}}(t)\Delta\boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w},\psi)) d\mu \\ &+ \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \cdot \Delta w \; d\mu + \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \cdot \Delta \psi \; d\mu \Big] - o(\Delta t) \\ &= E(\boldsymbol{\mathcal{X}}(t+\Delta t)) + \Delta t(-\mu(\boldsymbol{\mathcal{X}}(t) \Delta \boldsymbol{\mathcal{X}}(t+\Delta t))) \cdot \int_{\boldsymbol{\mathcal{X}}(t)\Delta\boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w},\psi)) d\mu \\ &- \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \cdot \Delta w \; d\mu - \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \cdot \Delta \psi \; d\mu - o(\Delta t) \\ &\leq E(\boldsymbol{\mathcal{X}}(t+\Delta t)) + \Delta t \cdot \delta \cdot \int_{\boldsymbol{\mathcal{X}}(t)\Delta\boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w},\psi)) d\mu \\ &- \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \cdot \Delta w \; d\mu - \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \cdot \Delta \psi \; d\mu - o(\Delta t). \end{aligned}$$

Subtracting $E(\boldsymbol{\mathcal{X}}(t+\Delta t))$ from both sides and combining like terms,

$$\begin{aligned}
E(\boldsymbol{\mathcal{X}}(t)) - E(\boldsymbol{\mathcal{X}}(t+\Delta t)) &\leq E(\boldsymbol{\mathcal{X}}(t+\Delta t)) + \Delta t \cdot \delta \cdot \int_{\boldsymbol{\mathcal{X}}(t)\Delta\boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w},\psi))d\mu \\
&\quad - \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \cdot \Delta w \; d\mu \\
&\quad - \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \cdot \Delta \psi \; d\mu - o(\Delta t) - E(\boldsymbol{\mathcal{X}}(t+\Delta t) \\
&\leq \Delta t \cdot \delta \cdot \int_{\boldsymbol{\mathcal{X}}(t)\Delta\boldsymbol{\mathcal{X}}(t+\Delta t)} \ell(\hat{f}(\mathbf{w},\psi))d\mu \\
&\quad - \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \cdot \Delta w \; d\mu - \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \cdot \Delta \psi \; d\mu - o(\Delta t).
\end{aligned}$$

Additionally, we can assume that the loss function ℓ is bounded above by a constant M_0, so

$$\begin{aligned}
E(\boldsymbol{\mathcal{X}}(t)) - E(\boldsymbol{\mathcal{X}}(t+\Delta t)) &\leq \Delta t \cdot \delta M_0 - \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \cdot \Delta w \; d\mu \\
&\quad - \Delta t \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \cdot \Delta \psi \; d\mu - o(\Delta t).
\end{aligned}$$

To consider this difference in expected values across future tasks, set $\Delta t = 1$ and let us sum across such future tasks,

$$\begin{aligned}
\sum_{\tau=t}^{T} (E(\boldsymbol{\mathcal{X}}(\tau)) - E(\boldsymbol{\mathcal{X}}(\tau+1))) &\leq \sum_{\tau=t}^{T} \Big(M_0 \cdot \delta - \int_{\boldsymbol{\mathcal{X}}(\tau)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \cdot \Delta w \; d\mu \\
&\quad - \int_{\boldsymbol{\mathcal{X}}(t)} \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \cdot \Delta \psi \; d\mu \Big).
\end{aligned}$$

Now, we can choose the following for each τ

$$\Delta \mathbf{w} = \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi),$$

and

$$\Delta \psi = \ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi).$$

Thus,

$$\begin{aligned}
\sum_{\tau=t}^{T} (E(\boldsymbol{\mathcal{X}}(\tau)) - E(\boldsymbol{\mathcal{X}}(\tau+1))) &\leq \sum_{\tau=t}^{T} \Big(M_0 \cdot \delta - \int_{\boldsymbol{\mathcal{X}}(\tau)} \Big(\ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \Big)^2 \; d\mu \\
&\quad - \int_{\boldsymbol{\mathcal{X}}(t)} \Big(\ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \Big)^2 \; d\mu \Big),
\end{aligned}$$

as desired. Note that the the terms $\Big(\ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \Big)^2$ and $\Big(\ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \Big)^2$ above are positive, and so each respective integral will be positive. Thus, we can view the subtraction of the integral of $\Big(\ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\mathbf{w}} \hat{f}(\mathbf{w},\psi) \Big)^2$ as the impact of changing the weights to the overall difference in loss values, and the subtraction of the integral of $\Big(\ell'(\hat{f}(\mathbf{w},\psi)) \cdot \partial^1_{\psi} \hat{f}(\mathbf{w},\psi) \Big)^2$ as the impact of changing the architecture to the overall difference in loss values.

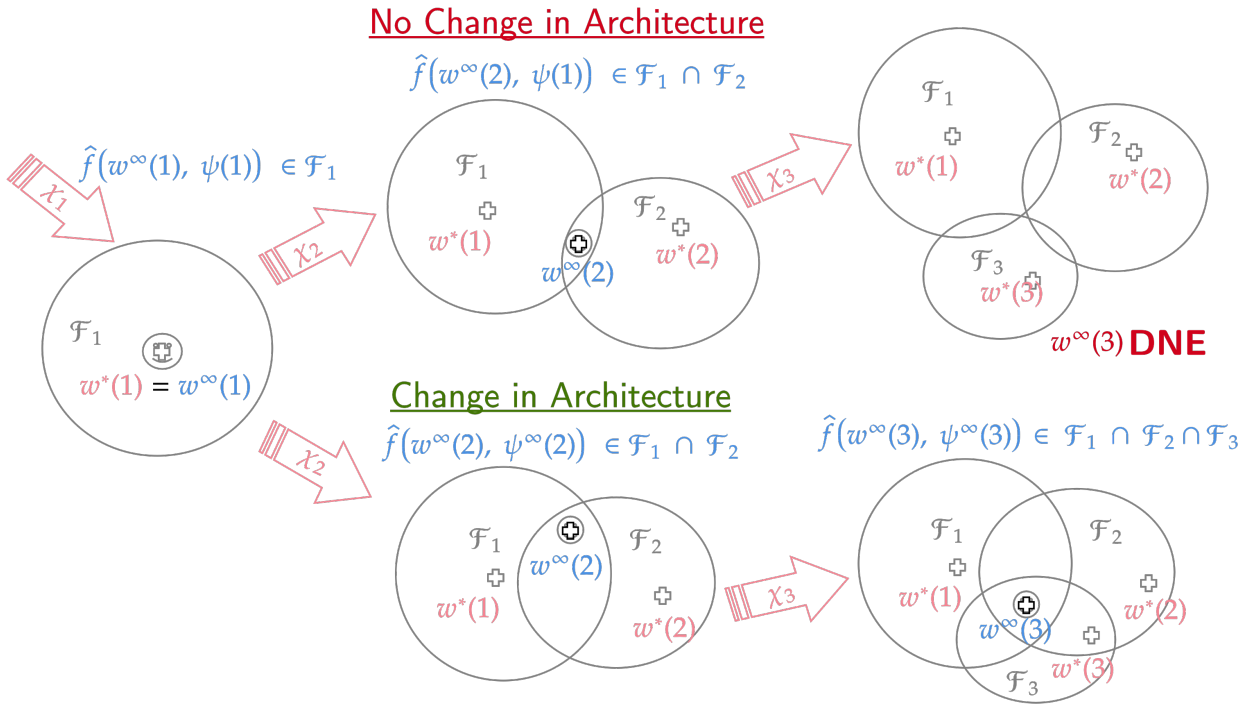

Figure 5: The Solution, where we change the size of the intersection space by introducing more capacity, through choosing novel hyperparameters

The previous theorem provided us with a lower bound (9) which was inherently smaller than (**??**) since we now can subtract the change in the architecture $\psi(t)$ as well as the change in the weights $\mathbf{w}(t)$. This leads us to the following corollary.

Corollary 10 *Suppose $\boldsymbol{\mathcal{X}}(t)$ and $\boldsymbol{\mathcal{X}}(t+\Delta t)$ are two consecutive tasks in the measure space $(\overline{\boldsymbol{\mathcal{X}}}, \mathcal{B}(\overline{\boldsymbol{\mathcal{X}}}), \mu)$. Let $\mu(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t+\Delta t)) \geq \delta$ for $0 < \delta \leq 1$. The architecture ψ of a network can be changed to absorb the impact of $\mu(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t+\Delta t)) \geq \delta$.*

Proof 5 *Let $\varepsilon > 0$ small. Since the expected value function is continuous with respect to measure by Theorem 6, we can find a corresponding $\delta > 0$. Suppose we have two tasks $\boldsymbol{\mathcal{X}}(t)$ and $\boldsymbol{\mathcal{X}}(t+\Delta t))$ for which $\mu(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t+\Delta t)) \geq \delta$. Moreover, suppose $|E(\boldsymbol{\mathcal{X}}(t+\Delta t)) - E(\boldsymbol{\mathcal{X}}(t))| \geq \varepsilon$. Then, by Theorem 9, we see that changing the architecture and training on a new optimal architecture that we can have*

$$|\nabla E(\boldsymbol{\mathcal{X}}(t))| \geq M_0\delta - L_1|\Delta w| \int_{\boldsymbol{\mathcal{X}}(t)} \|\partial^1_{\mathbf{w}} \hat{f}(w(t,\psi), \psi(t))\|_{L^p} d\mu - L_1|\Delta\psi| \int_{\boldsymbol{\mathcal{X}}(t)} \|\partial^1_{\psi} \hat{f}(w(t,\psi), \psi(t))\|_{L^p} d\mu. \tag{15}$$

Thus, changing the architecture of the network can offset the impact of consecutive tasks whose data differ substantially.

Krishnan: I am going to go from here and build the rest of the paper, we will revise the above when you get a chance to fix the theorem.

Due to Corollary 10, the intersection space that offers feasible solutions to the continual learning problem vanishes as the data distribution changes over time. As a consequence, it is then necessary to change the architecture so as to increase the potential size of the intersection space. In particular, we do not fix the architecture for our problem.

To understand how this can be accomplished, consider Figure 5, which describes the method for the first and second task. We begin at task $t = 1$ and determine an optimal weight denoted $w^*(1) = w^\infty(1)$. This gives

us a NN solution in $\mathcal{F}_1$. Then, as the next task is observed, we perform a derivative free hyper-parameter search, in particular, the directional direct search method Larson et al. (2019) to obtain new architecture. The solution to this search is denoted $\psi^{\infty}(2)$ as in Figure 5

Notably, there is a new quandary, since the new architecture introduces a parameter space that may be of different size than the one used for the previous task, it is unfortunately not feasible to trivially transfer information from the previous architecture to the new one. The common and the state of the art solution to this problem is to initialize the parameters in the new architecture from random and retrain on all available tasks, which is resource heavy as well as impractical. To obviate this necessity, we develop a low rank transfer algorithm that seeks to transfer information between the previous architecture to the new architecture in an efficient way. We describe this solution next.

5 CL Solution

In particulaer, our goal is to solve the following bilevel problem

$$V^*(t, \mathbf{w}(t)) = \min_{\mathbf{w}\in\mathcal{W}(\psi^*(t))} V(t, \mathbf{w}(t)), \quad \text{where } \psi^*(t) = \arg\min_{\psi\in\Psi} J(\mathbf{w}(t), \psi, \boldsymbol{\mathcal{X}}(t)), \tag{bi-level}$$

Noting that the weight search space at each t is a function of the optimal architecture, we change the weight space for each t as

$$\mathbf{w}(t + \Delta\ t) = \mathrm{A}^*(t)\mathbf{w}(t)\mathrm{B}(t)^*T, \tag{16}$$

where A and B are matrices that enable the transfer of information between two different parameter space and $\mathrm{A}^*, \mathrm{B}^*$ ar the optimal values such that loss of information is minimal. This change in the weight space for each new task introduce dynamics into the continual learning problem defined by V^*. The following result therefore provides the total variation in V^* as a function of tasks.

Proposition 11 *The total variation in $V^*(t)$ at any given task t is given by*

$$-\frac{\partial V^*}{\partial t} = \min_{\mathbf{w}\in\mathcal{W}(\psi^*(t))} J(\mathbf{w}(t), \psi^*(t), \boldsymbol{\mathcal{X}}(t)) + \frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}\frac{d\boldsymbol{\mathcal{X}}}{dt} + \frac{\partial V^*}{\partial \mathbf{w}}[\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T + u(t)], \tag{17}$$

where $\mathrm{A}^*(t), B^*(t)$ are optimal $\mathrm{A}(t)$ and $\mathrm{B}(t)$ for task t and $u(t)$ represents the updates made to the each weights matrix of the new dimensions.

Proof 6 (proof of Proposition 11) *Let J, V, and V^* be as defined above. To begin, we split the sum in $V(t)$ over the discrete intervals $[t, t + \Delta t]$ and $[t + \Delta t, T]$. Observe,*

$$\begin{aligned}
V^*(t) &= \min_{w\in\mathcal{W}(\psi^*(t))} \int_t^T J(w(\tau), \psi^*(t), \boldsymbol{\mathcal{X}}(\tau))d\tau \\
&= \min_{w\in\mathcal{W}(\psi^*(t))} \left[\int_t^{t+\Delta t} J(w(\tau), \psi^*(t), \boldsymbol{\mathcal{X}}(\tau))d\tau + \int_{t+\Delta t}^T J(w(\tau), \psi^*(t), \boldsymbol{\mathcal{X}}(\tau))d\tau\right] \\
&= \min_{w\in\mathcal{W}(\psi^*(t))} \int_t^{t+\Delta t} J(w(\tau), \psi^*(t), \boldsymbol{\mathcal{X}}(\tau))d\tau + V^*(t + \Delta t) \\
&= \min_{w\in\mathcal{W}(\psi^*(t))} J(\mathbf{w}(t), \psi^*(t), \boldsymbol{\mathcal{X}}(t))\Delta t + V^*(t + \Delta t).
\end{aligned} \tag{18}$$

Now, we provide the Taylor series expansion of $V^*(t + \Delta t)$ about t. Notice,

$$\begin{aligned}
V^*(t + \Delta t) &= V^*(t) + \Delta t\left[\frac{\partial V^*}{\partial t} + \frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}\frac{d\boldsymbol{\mathcal{X}}}{dt} + \frac{\partial V^*}{\partial \mathbf{w}}\frac{\partial \mathbf{w}}{dt}\right] + o(\Delta t) \\
&= V^*(t) + \Delta t\left[\frac{\partial V^*}{\partial t} + \frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}\frac{d\boldsymbol{\mathcal{X}}}{dt} + \frac{\partial V^*}{\partial \mathbf{w}}[\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T + u(t)]\right] + o(\Delta t),
\end{aligned} \tag{19}$$

where $u(t)$ represents the updates made to the each weights matrix of the new dimensions. Substituting 19 into 18, we have

$$V^*(t) = \min_{w \in \mathcal{W}(\psi^*(t))} J(\mathbf{w}(t), \psi^*(t), \boldsymbol{\mathcal{X}}(t))\Delta t + V^*(t) + \Delta t\left[\frac{\partial V^*}{\partial t} + \frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}\frac{d\boldsymbol{\mathcal{X}}}{dt} + \frac{\partial V^*}{\partial \mathbf{w}}[\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T + u(t)]\right] + o(\Delta t).$$

Canceling $V^*(t)$ gives

$$0 = \min_{w \in \mathcal{W}(\psi^*(t))} J(\mathbf{w}(t), \psi^*(t), \boldsymbol{\mathcal{X}}(t))\Delta t + \Delta t\left[\frac{\partial V^*}{\partial t} + \frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}\frac{d\boldsymbol{\mathcal{X}}}{dt} + \frac{\partial V^*}{\partial \mathbf{w}}[\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T + u(t)]\right] + o(\Delta t).$$

Dividing both sides by Δt produces

$$0 = \min_{w \in \mathcal{W}(\psi^*(t))} J(\mathbf{w}(t), \psi^*(t), \boldsymbol{\mathcal{X}}(t)) + \frac{\partial V^*}{\partial t} + \frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}\frac{d\boldsymbol{\mathcal{X}}}{dt} + \frac{\partial V^*}{\partial \mathbf{w}}[\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T + u(t)].$$

Finally, reordering gives

$$-\frac{\partial V^*}{\partial t} = \min_{w \in \mathcal{W}(\psi^*(t))} J(\mathbf{w}(t), \psi^*(t), \boldsymbol{\mathcal{X}}(t)) + \frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}\frac{d\boldsymbol{\mathcal{X}}}{dt} + \frac{\partial V^*}{\partial \mathbf{w}}[\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T + u(t)],$$

as desired.

Observe that (17) is missing the term $\frac{\partial V^*}{\partial \psi}\frac{\partial \psi}{\partial t}$, this is intentional as the exact effects of architecture cannot be measured through derivatives in this PDE. Instead, in this PDE, this effect is absorbed through the change in tensors $\mathrm{A}(t)$ and $\mathrm{B}(t)$. Due to this absorption, it is feasible to now define a lower bound on the first different as follows.

Theorem 12 *Given proposition 11, assume there exists an stochastic gradient based optimization procedure and choose $u(t) = -\sum^I \alpha(i)\mathrm{g}^{(i)}$, where I is the number of updates and $\alpha(i)$ is some learning rate. Choose $\alpha(i)$ such that $\|\mathrm{g}_{\mathrm{MIN}}\|_{W^{k,p}(\mathrm{D})}\left\|\sum^I \alpha(i)\right\|_{W^{k,p}(\mathrm{D})} \to 0$ as $I \to \infty$ and $\min_{\mathbf{w}\in\mathcal{W}(\psi^*(t))} J(\mathbf{w}(t), \psi^*(t), \boldsymbol{\mathcal{X}}(t)) \leq \epsilon$. Then, as long as the architecture is chosen such that $\rho^{(\mathbf{w})}_{\mathrm{MAX}}\left\|\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T\right\|_{W^{k,p}(\mathrm{D})} = \rho^{(\mathrm{x})}_{\mathrm{MAX}}\delta$, the total variance in the continual learning problem due to the data is bounded by ϵ.*

Proof 7 *Given proposition 11, assume there exists an stochastic gradient based optimization procedure and choose $u(t) = -\sum^I \alpha(i)\mathrm{g}^{(i)}$ such that $\min_{\mathbf{w}\in\mathcal{W}(\psi^*(t))} J(\mathbf{w}(t), \psi^*(t), \boldsymbol{\mathcal{X}}(t)) \leq \epsilon$, where I is the number of updates. Then, we have*

$$-\frac{\partial V^*}{\partial t} \leq \epsilon + \frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}\frac{d\boldsymbol{\mathcal{X}}}{dt} + \frac{\partial V^*}{\partial \mathbf{w}}\left(\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T - \sum^I \alpha(i)\mathrm{g}^{(i)}\right)$$

For any dt in terms of tasks t let $\frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}\frac{d\boldsymbol{\mathcal{X}}}{dt} \leq \left\|\frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}\right\|_{W^{k,p}(\mathrm{D})}\left\|\frac{d\boldsymbol{\mathcal{X}}}{dt}\right\|_{W^{k,p}(\mathrm{D})} \leq \rho^{(\mathrm{x})}_{\mathrm{MAX}}\delta$ as $\left\|\frac{d\boldsymbol{\mathcal{X}}}{dt}\right\|_{W^{k,p}(\mathrm{D})} \geq \mu(\boldsymbol{\mathcal{X}}(t) \triangle \boldsymbol{\mathcal{X}}(t+1)) \geq \delta$ and $\left\|\frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}\right\|_{W^{k,p}(\mathrm{D})}$ is less than the largest singular value of $\frac{\partial V^*}{\partial \boldsymbol{\mathcal{X}}}$. Thus we write

$$-\frac{\partial V^*}{\partial t} \leq \epsilon + \rho^{(\mathrm{x})}_{\mathrm{MAX}}\delta + \frac{\partial V^*}{\partial \mathbf{w}}\left(\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T - \sum^I \alpha(i)\mathrm{g}^{(i)}\right)$$

Taking g_{MIN} to be the smallest value of the gradient over all update iterations, we have

$$\begin{aligned}
-\frac{\partial V^*}{\partial t} &\leq \epsilon + \rho^{(\mathrm{x})}_{\mathrm{MAX}}\delta + \frac{\partial V^*}{\partial \mathbf{w}}\left(\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T - \mathrm{g}_{\mathrm{MIN}}\sum^I \alpha(i)\right) \\
&\leq \epsilon + \rho^{(\mathrm{x})}_{\mathrm{MAX}}\delta + \left\|\frac{\partial V^*}{\partial \mathbf{w}}\right\|_{W^{k,p}(\mathrm{D})}\left\|\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T\right\|_{W^{k,p}(\mathrm{D})} - \frac{\partial V^*}{\partial \mathbf{w}}\left(\mathrm{g}_{\mathrm{MIN}}\sum^I \alpha(i)\right) \\
&\leq \epsilon + \left[\rho^{(\mathrm{x})}_{\mathrm{MAX}}\delta + \rho^{(\mathbf{w})}_{\mathrm{MAX}}\left\|\mathrm{A}^*(t)\mathbf{w}(t)(\mathrm{B}^*(t))^T\right\|_{W^{k,p}(\mathrm{D})} - \rho^{(\mathbf{w})}_{\mathrm{MIN}}\|\mathrm{g}_{\mathrm{MIN}}\|_{W^{k,p}(\mathrm{D})}\left\|\sum^I \alpha(i)\right\|_{W^{k,p}(\mathrm{D})}\right]
\end{aligned}$$

This finally provides

$$\frac{\partial V^*}{\partial t} \leq \text{MAX}\epsilon - \text{MIN}\left[\rho^{(\text{x})}_{\text{MAX}}\delta + \rho^{(\mathbf{w})}_{\text{MAX}}\left\|\text{A}^*(t)\mathbf{w}(t)(\text{B}^*(t))^T\right\|_{W^{k,p}(\text{D})} - \rho^{(\mathbf{w})}_{\text{MIN}}\left\|\text{g}_{\text{MIN}}\right\|_{W^{k,p}(\text{D})}\left\|\sum^{I}\alpha(i)\right\|_{W^{k,p}(\text{D})}\right]$$

For the total variation to be upper bounded, we need the quantity in the brackets to go to zero, this provides.

$$\left[\rho^{(\text{x})}_{\text{MAX}}\delta + \rho^{(\mathbf{w})}_{\text{MAX}}\left\|\text{A}^*(t)\mathbf{w}(t)(\text{B}^*(t))^T\right\|_{W^{k,p}(\text{D})} - \rho^{(\mathbf{w})}_{\text{MIN}}\left\|\text{g}_{\text{MIN}}\right\|_{W^{k,p}(\text{D})}\left\|\sum^{I}\alpha(i)\right\|_{W^{k,p}(\text{D})}\right] = 0$$

$$\rho^{(\text{x})}_{\text{MAX}}\delta = \left[\rho^{(\mathbf{w})}_{\text{MIN}}\left\|\text{g}_{\text{MIN}}\right\|_{W^{k,p}(\text{D})}\left\|\sum^{I}\alpha(i)\right\|_{W^{k,p}(\text{D})} - \rho^{(\mathbf{w})}_{\text{MAX}}\left\|\text{A}^*(t)\mathbf{w}(t)(\text{B}^*(t))^T\right\|_{W^{k,p}(\text{D})}\right]$$

Therefore, even when $\alpha(i)$ is chosen such that $\left\|\text{g}_{\text{MIN}}\right\|_{W^{k,p}(\text{D})}\left\|\sum^{I}\alpha(i)\right\|_{W^{k,p}(\text{D})}$ as $I \to \infty$, $\rho^{(\text{x})}_{\text{MAX}}\delta$ can be countered by $\rho^{(\mathbf{w})}_{\text{MAX}}\left\|\text{A}^*(t)\mathbf{w}(t)(\text{B}^*(t))^T\right\|_{W^{k,p}(\text{D})}$ and this bound is tunable due to the change in the architecture. Therefore, as long as $\rho^{(\mathbf{w})}_{\text{MAX}}\left\|\text{A}^*(t)\mathbf{w}(t)(\text{B}^*(t))^T\right\|_{W^{k,p}(\text{D})} = \rho^{(\text{x})}_{\text{MAX}}\delta$ the total variance is bounded for all t.

Remark 5 *Theorem 12 is crucial because, it shows that, in the presence of a perfect stochastic gradient algorithm, the change in the tasks can be countered by the choice of the architecture. By choosing a new architecture and introducing the A and B matrices such that, within the CL problem, the effect of varying data distribution can be mitigated and the CL problem can be acceptably solved.*

Therefore, the rest of the section is dedicated to constructing an algorithm such that the condition to ensure convergence is satisfied. In summary (refer Figure 6), for every new task, we utilize an off the shelf algorithm to search for a new optimal architecture. Once a new architecture is found, we complete what we call "Low Rank Transfer" from previous task's optimal weights to the new task parameter space. Then, using an off the shelf continual learning algorithm, we train the new architecture for the new task while balancing memory on all the prior tasks. In the process of Low Rank Transfer, we guarantee transfer of learning by requiring that the $\frac{\partial V^*}{\partial t} \leq \eta$, allowing for the intersection of the NN search spaces between subsequent tasks to increase. Thus, there are three components in our algorithm, step 1: the architecture search, step 2: the low rank transfer and step 3: the continual learning. While step 3 is directly adapted from Chakraborty & Raghavan (2025); Raghavan & Balaprakash (2021), step 1 and 2 are novel contributions for this work.

5.1 Step 1: Architecture Search

Although there are numerous architecture search methods that we could employ, we chose a derivative-free approach that completes a local search using finite difference approximations. This choice was made to mimic the weak derivatives available to us from viewing NN as functions of Sobolev spaces. It will become evident in section 11 that a notion of the change in the architecture is needed in order to model the dynamics and solve the lower optimization. We call the search method used a neighborhood directional direct-search (NDDS). The standard directional direct-search method is described in the survey by Larson et al. (2019).

The intuition behind NDDS is that we check "neighboring" architectures and compare the expected value of the NN when it trains on the neighboring architecture to that of the current architecture. If the neighboring architecture has a smaller expected value, then we "move" in that direction by selecting it as the new architecture. We then check the "neighbors" of this new architecture. As the architecture is a discrete variable, this search method provides a notion of a "gradient."

With this intuition, we now describe NDDS in depth. As we discuss this method, consult Algorithm 1. For task $t \in \boldsymbol{\mathcal{T}}$, set $\boldsymbol{\mathcal{X}} = \mathcal{Y}$ to be our training data and $J(\mathbf{w}(t), \psi(t), \mathcal{Y})$ the expected value function. Set

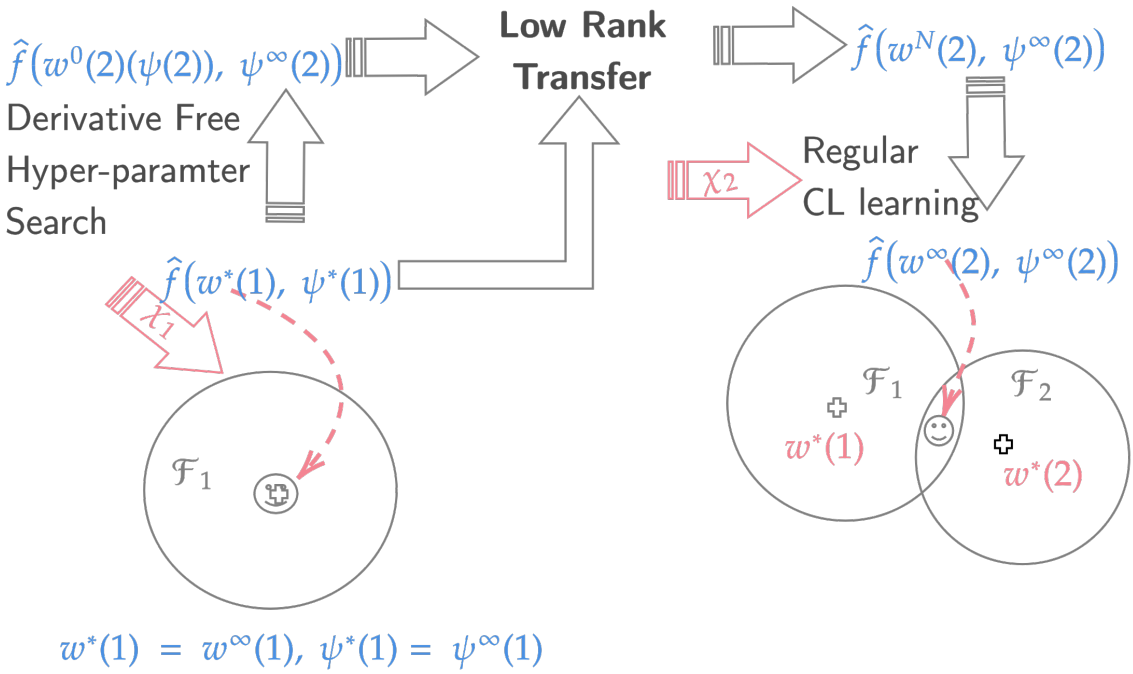

Figure 6: How we do this?

$x_s = \psi(t)$ and let D_s be a finite set of directions. Further, we choose a "step size" $\alpha_s \in \mathbb{N}$. Now we generate the "neighboring" points for x_s via D_s and α_s. We call these points *poll points* and define them below

$$PollPoints = \{x_s + \alpha_s d_i : d_i \in D_s\}.$$

Now, using a randomly selected subset of our data, denoted $\mathcal{Y}_s$, we determine and compare the expected value of NN with randomly initialize weights according to each poll point architecture. If one of these poll points results in a smaller expected value than x_s, we set x_s equal to the poll point. We then repeat this process starting with this new architecture. This search terminates when the expected value is beneath a previously chosen threshold or after five times of repeating. In practice, we chose the threshold to be a percentage lower than the expected value produced by the original architecture. Krishnan: Wasnt there a nice diagram you had to illustrate this?

Let us walk through an example. Suppose we would like to learn the optimal number of neurons per layer in our hidden layers for training a feedforward neural network (FNN) on a data set. Due to the data, suppose the input layer is fixed at 784 neurons and the output layer is fixed at 10 neurons per layer. Let us fix the number of hidden layers at 2 begin with 50 neurons in each hidden layer. We set $\mathcal{Y}_s$ to be a randomly chosen appropriately sized subset of task t data and previous task data. Our direction set is $D_s = D = \{[0, 0, 10, 0], [0, 10, 0, 0]\}$ and will be the same for every s. For our search, we start by setting the step size $\alpha_s = 10$. For the first round, we have $x_s = [784, 50, 50, 10]$, so our poll points are $[784, 60, 50, 10]$, $[784, 50, 60, 10]$, and $[784, 60, 60, 10]$. We then train a FNN of each size on the data set Y_s and determine the corresponding expected values. If the expected value of the FNN $[784, 60, 50, 10]$ is smaller than that of the FNN $[784, 50, 50, 10]$, for example, then $x_s = [784, 60, 50, 10]$. We would then continue this process until the expected value of our network is less than our set threshold or we exhaust our neighborhood search.

5.2 Step 2: Low Rank Transfer

Now that the optimal architecture (i.e. number of neurons per layer) has been determined for the current task, it is immediately obvious that that the size of the weights tensor will not match with the new architecture.

Algorithm 1: Neighborhood Direct-Directional Search (NDDS)

Input: Initial architecture x_s, step size $\alpha_0 \in \mathbb{N}$
Input: Threshold parameter threshold $\in \mathbb{R}$
Output: Updated architecture x_s
$j \leftarrow 0$;
loss $\leftarrow$ `training_loop`$(x_s, \mathcal{Y}_s)$;
while loss $>$ threshold ***or*** $j < 5$ **do**
  pollPoints $\leftarrow \{x_s + \alpha_s d_i : d_i \in D_s\}$;
  foreach poll $\in$ pollPoints **do**
    loss$_s \leftarrow$ `training_loop`$(\text{poll}, \mathcal{Y}_s)$;
    if loss $\leq$ loss$_s$ **then**
      $x_s \leftarrow x_s$;
    else
      $x_s \leftarrow$ poll;
      loss $\leftarrow$ loss$_s$;
    end
  end
  $j \leftarrow j + 1$;
end

Thus, we seek a method to determine a weights tensor corresponding to the new optimal architecture that retains previously learned information and transfers it. We propose a method we call Low Rank Transfer.

Before we dive into the steps of this method, let us set some assumptions and notation. Recall that the only component of the architecture we seek to learn is the number of neurons per layer in our neural network. Thus, we assume our network has d layers, and we fix all other architecture parameters except for the number of neurons in our hidden layers. From our Definition 3, $\psi_i(t)$ provides the dimensions for the corresponding weights matrix in each layer of our network.

In order of appearance, we assign the values of $\psi(t)$ to the values r_i, s_i for each i such that $1 \leq i \leq d$. If $\psi^*(t)$ represents the optimal architecture returned from the NDDS completed for task t, then similarly we can assign the values of the dimensions of the weights matrices of each layer of $\psi_i^*(t)$ to be a_i, b_i for $1 \leq i \leq d$. Our goal is to utilizing the original weights matrices to project into the weights matrix space for the new architecture.

To begin, we initialize dimension 3 tensors $\mathrm{A}(t), \mathrm{B}(t)$, which are each comprised of d matrices. Let each matrix $\mathrm{A}_i(t)$ in $\mathrm{A}(t)$ be randomly generated with dimensions $a_i \times r_i$ for $1 \leq i \leq d$. Similarly, let each matrix B_i in $\mathrm{B}(t)$ be randomly generated with size $b_i \times s_i$ for $1 \leq i \leq d$. Then, set $C(t) = \mathrm{A}(t)\mathbf{w}(t)\mathrm{B}^T(t)$. More specifically $C(t)$ is comprised of d matrices such that $C_i(t) = \mathrm{A}_i(t)\mathbf{w}_i(t)B_i^T(t)$ for all $1 \leq i \leq d$. Notice, that the dimensions of each $C_i(t)$ are $a_i \times b_i$. These are the dimensions of the weights matrices for a NN with the new optimal architecture $\psi^*(t)$. See Figure 7 to understand the construction of $C(t)$ more explicitly.

To prevent loss of information from $\mathbf{w}(t)$, when we make the transfer to $C(t)$, we shall train only the $\mathrm{A}(t)$ and $\mathrm{B}(t)$ portions of the new weights tensor $C(t)$ on the task data for a chosen number of epochs, while freezing the $\mathbf{w}(t)$ tensor. This additional training ensures a transfer of learning to the new weights corresponding to the new optimal architecture. If $C^*(t)$ represents $C(t)$ after this training on just tensors $\mathrm{A}(t)$ and $\mathrm{B}(t)$ and $\psi(t+1) = \psi^*(t)$, then we conclude our process by letting $w(t+1) = C^*(t)$ and completing the standard training of our neural network $\hat{f}(w(t+1), \psi(t+1)$ on task data.

In algorithm 2, we summarize each step of our method described in the previous subsections.

Krishnan: reached here

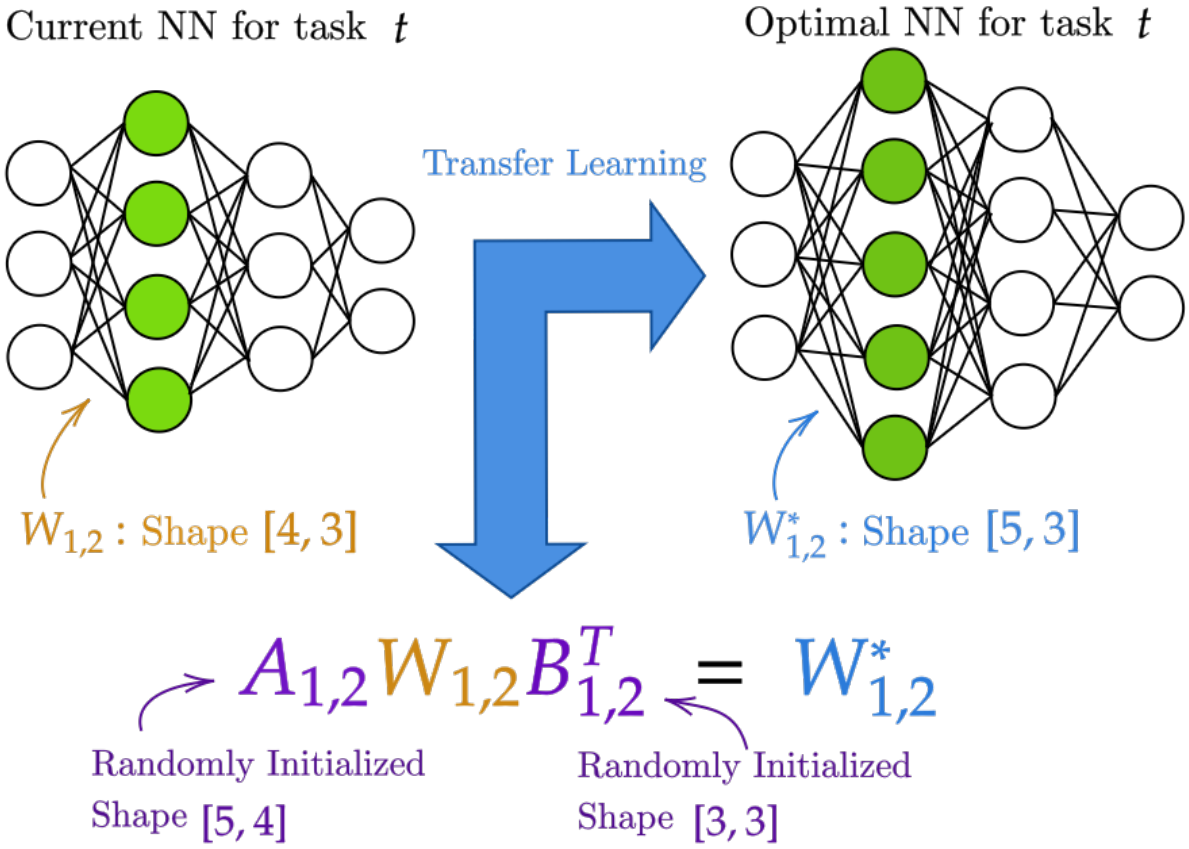

Figure 7: Method of Low Rank Transfer at a single layer

Algorithm 2: Main Training Loop

Choose $\mathbf{w}(t)$ and $\psi(t)$ to begin
Set `epoch hyper-parameter`
for $t = 0, 1, \ldots, T$ **do**
  Step 1: Standard Training of $\mathbf{w}(t)$
  $\mathbf{w}(t) \leftarrow$`training_loop`$(\mathbf{w}(t), \psi(t),\ \boldsymbol{\mathcal{X}}(t),$ `epochs`$)$
  Step 2: Architecture Search
  $\psi^*(t) \leftarrow$ `NDDS`$(\psi(t),\ \boldsymbol{\mathcal{X}}(t))$
  Step 3: Initialize $\mathrm{A}(t), \mathrm{B}(t)$
  for $i = 1, ..., d$ **do**
    $\mathrm{A}_i(t), \mathrm{B}_i \leftarrow$`init_AB`$(a_i, b_i, r_i, s_i)$
  end
  Step 4: Set $C(t)$
  for $i = 1, ..., d$ **do**
    $C_i(t) = \mathrm{A}_i(t)\mathbf{w}_i(t)B_i^T(t)$
  end
  Step 5: Fix $\mathbf{w}(t)$, Train $\mathrm{A}(t), \mathrm{B}(t)$ for $\hat{f}(C(t), \psi(t+1))$
  $C^*(t) \leftarrow$ `train_AB`$(C(t), \psi(t+1),\ \boldsymbol{\mathcal{X}}(t),$ `epochs`$)$
  Step 6: Set New Weights and Architecture
  $w(t+1) = C^*(t)$
  $\psi(t+1) = \psi^*(t)$
  Step 7: Standard Training on New NN `training_loop`
  $(\mathbf{w}(t+1), \psi(t+1),\ \boldsymbol{\mathcal{X}}(t),$ `epochs`$)$
end

6 Related Works

As noted in the introduction, theoretical and empirical results have proven that even in traditional machine learning, training weights of the network alone is not enough. By utilizing tools such as NAS (Neural Architecture Search), LoRA (Low-Rank Adaptation), and PEFT (Parameter-Efficient Fine Tuning), the accuracy of a trained model can readily be increased (Liu et al. (2021), Hu et al. (2022), and Han et al. (2024)). However, in the context of CL, learning optimal hyperparameters or architecture for each task poses a challenge. Not only is it necessary to determine what the optimal hyperparameters are at each task, but it is imperative that learning of previous tasks is transferred.

Toward this end, networks with dynamic structures, such as Progressive Neural Networks (PGN) and Dynamically Expandable Networks (DEN) were introduced in Rusu et al. (2016) and Yoon et al. (2017). Progressive Neural Networks add a layer to the neural network at each observed task Rusu et al. (2016). While the model's accuracy did improve with such structure, extensive compute time is unavoidable as more tasks, and hence layers, are observed. Dynamically Expandable Networks attempted to remedy the computing issues of PGNs. DENs first determine a sub-collection of neurons to train on for a given task (via a metric) and then add a layer of neurons to the network where appropriate Yoon et al. (2017). This method also completes sparse regularization at each task to prune the network in attempt to reduce compute time compared to PGNs. The use of a DEN did improve the model's accuracy. However, drawbacks such as compute time, overfitting, and transfer of learning still remained. Learning from previous tasks was not fully transferred, as the weights corresponding to the newly added network layers were randomly initialized.

In more recent years, researchers have built upon the idea of DENs by developing methods which optimally choose how and when to adjust network architecture. CLEAS (Continual Learning with Efficient Architecture Search) is the first integration of NAS into the CL framework Gao et al. (2022). This method only changes architecture when deemed necessary by utilizing a neuron-level NAS, rather than adding an entire layer to the neural network. Moreover, they prune their network at each step to remove unnecessary layers and/or neurons. The algorithm touts its smaller, purposely learned network structure, as it allows for reduced

network complexity, and hence computation time. In particular, CLEAS improves model accuracy by up to 6.70% and reduced network complexity by up to 51.0% on the three benchmark datasets. A strategy similar to CLEAS is CAS (Continual Architecture Search) by Pasunuru & Bansal (2019). The premise of only adding and removing to the architecture when deemed optimal for a given task, remains the same. However, rather than using NAS, they use what they call Efficient Neural Architecture Search (ENAS), which incorporates a weight-sharing strategy. SEAL (Searching Expandable Architectures for Incremental Learning) again seeks to change only the hyperparameters and architecture of the NN when necessary, but the method differs from CLEAS in how it determines when to alter the network Gambella et al. (2025). In particular, SEAL uses a capacity estimation metric to make such decision.

The previous methods all sought to expand the network in some fashion to increase the accuracy of the network. Now, we shift to how parameter-efficient fine-tuning methods have been utilized in CL. In particular, we investigate the use of LoRA (Low-Rank Adaptation) Hu et al. (2022). The two methods we will discuss here are intended for pre-trained transformers which are now attempting to learn tasks. The CoLoRA method utilizes a new LoRA adapter for each new task to train a task expert model Wistuba et al. (2023). The expert model is then used to train the transformer. At the time, this method produced state of the art results, but it maintains a high computational cost. Since then, the CLoRA method has been introduced Muralidhara et al. (2025). The key difference between these two strategies is the number of LoRA adapters. Rather than reinitializing and training a new LoRA adapter for each task, they use a single-adapter approach. This reduces the compute time, while maintaining accuracy and transferring learning.

In this paper, we attempt to combine the most profitable aspects of methods like CLEAS and CLoRA. In other words, our goal is to learn the optimal architecture for each task and utilize a low-rank transfer method to optimally transfer learning to the new architecture. The use of a LoRA-like method to easily guarentee transfer of learning from one architecture to another in standard CL has not been accomplished. Moreover, no previous algorithm has provided a theoretical mathematics framework to support their empirical data.

7 Experimental Results

In this section, we evaluate the algorithm presented above for regression, classification, and graph classification continual learning problems. For all three learning problems, we compare the performance of our algorithm to the standard continual learning approach utilized in Raghavan & Balaprakash (2021). For the regression problem, we consider the randomly generated sine data set and complete experiments where we learn five and 10 tasks. For the classification problem, we consider the Omniglot data set, which consists of handwritten characters. We also complete experiments where we learn five tasks in this setting. For the graph classification problem, we consider a set of randomly generated images utilizing the FakeDataset from PyTorch and complete experiments where we learn five tasks.

7.1 Regression

For the regression problem, we will complete three experiments. In each experiment, we use a feedforward neural network with four layers where we begin with 75 neurons in each of the hidden layers. We use AdamW with a learning rate of 1×10^{-4} in the training regime.

In the first experiment, we learn five tasks and train 500 epochs on each task. Figure 8 reveals the loss values as the NN trains (on the training data) with respect to different methods on the five tasks. The red graph is the standard continual learning method used in Raghavan & Balaprakash (2021), where no change is made to the architecture across tasks. This serves as our baseline for comparison. The blue graph describes the loss value if we change the architecture at each task, but rather than utilizing the low rank transfer method we describe in 2, we randomly reinitialize the weights. The green graph shows the loss value when our method of changing the architecture at each step and completing the low rank transfer to retain learning.

The blue graph suggests that changing architecture without transfer of learning leads to loss of learning. As the green graph displays a decrease in the loss value on the training data in comparison to the baseline continual learning, we can deduce that changing the architecture and transferring learning at each task leads

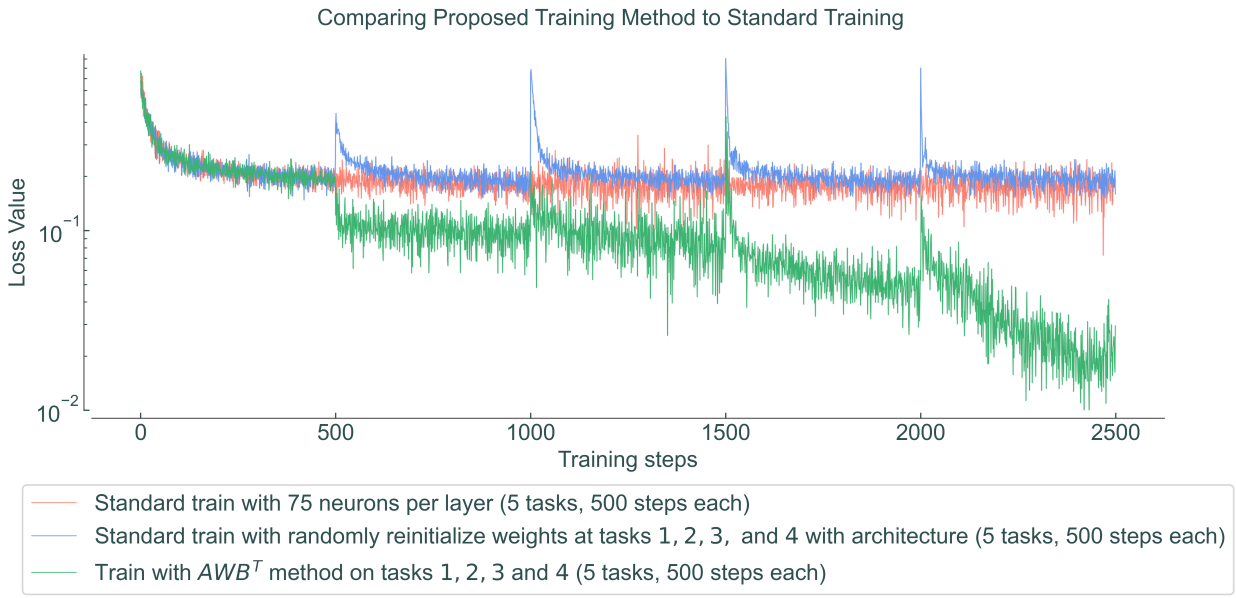

Figure 8: Experiment 1: Comparing loss values on training data for baseline continual learning method and method of learning optimal task architecture

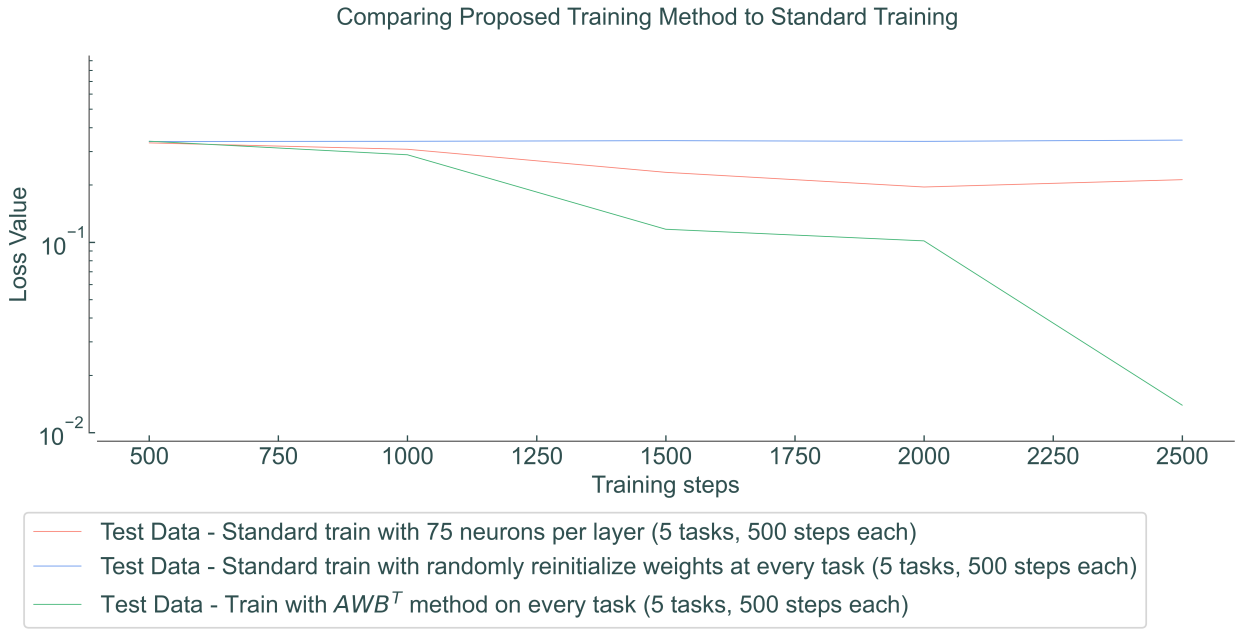

Figure 9: Experiment 1: Comparing loss values on test data for baseline continual learning method and method of learning optimal task architecture

to increased performance. This conclusion is supported by the performance of the the NN on the test data for this experiment, which is shown in Figure 9.

For the second experiment, we learn 10 tasks of data and train 500 epochs on each task. Figure 10 reveals the loss values as the NN trains (on the training data) with respect to different methods on the five tasks. The red graph is again the standard continual learning method used in Raghavan & Balaprakash (2021),

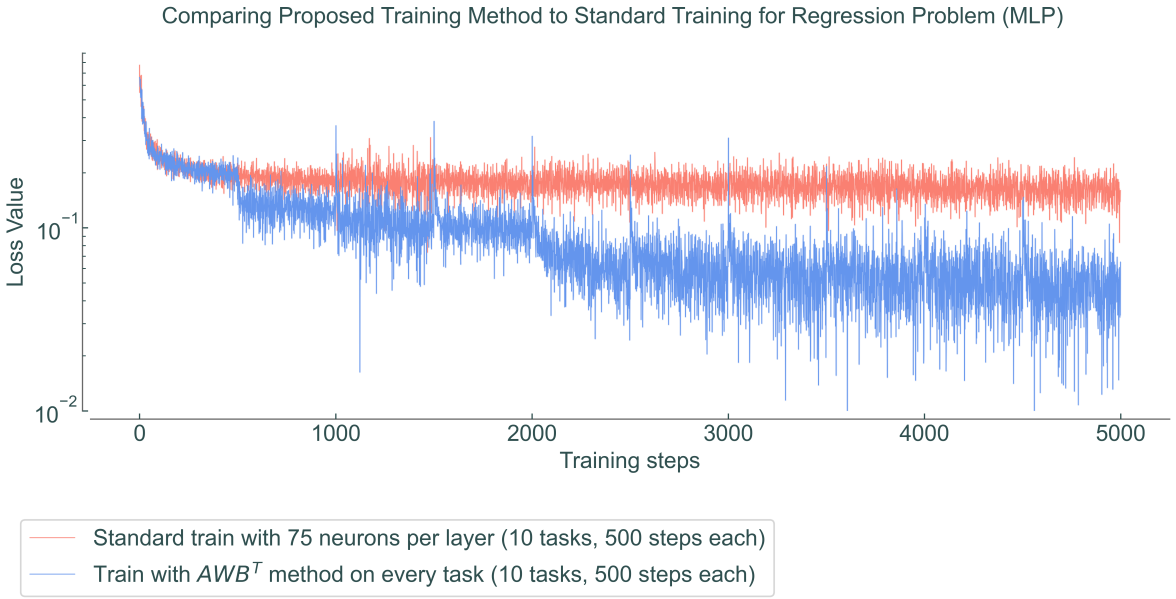

Figure 10: Regression Experiment 2: Comparing loss values on training data for baseline continual learning method and method of learning optimal task architecture

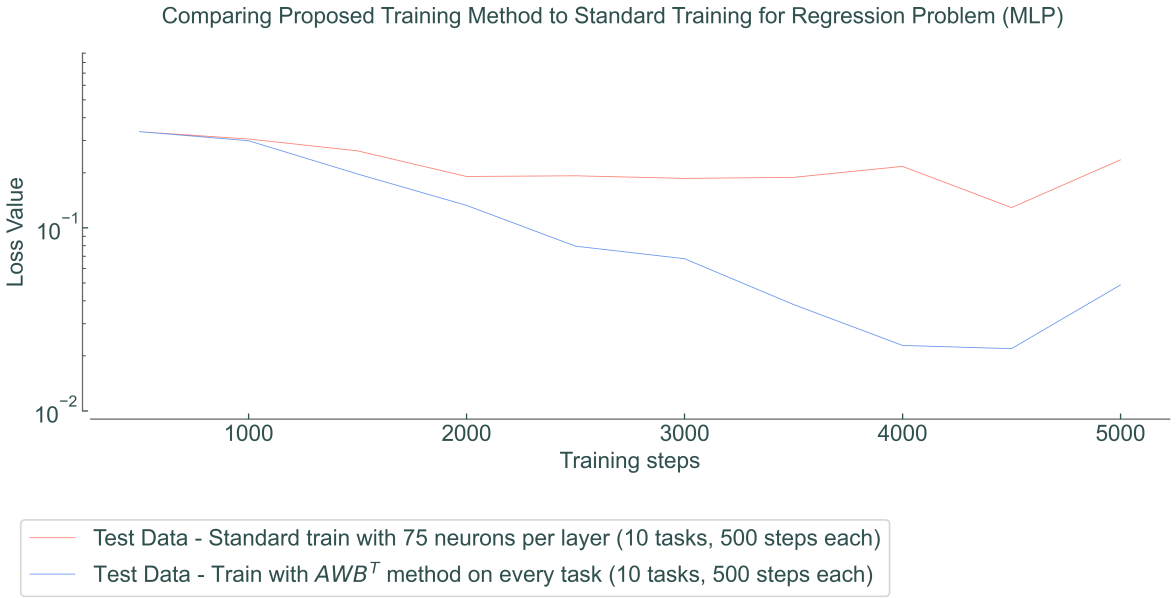

Figure 11: Regression Experiment 2: Comparing loss values on test data for baseline continual learning method and method of learning optimal task architecture

with no change made to the architecture across tasks. The blue graph shows the loss value when our method of changing the architecture at each step and completing the low rank transfer to retain learning.

As the blue graph shows a decrease in the loss value on the training data in comparison to the baseline continual learning, reveals that changing the architecture and transferring learning at each task leads to increased performance. This conclusion is supported by the performance of the the NN on the test data for this experiment, which is shown in Figure 11.

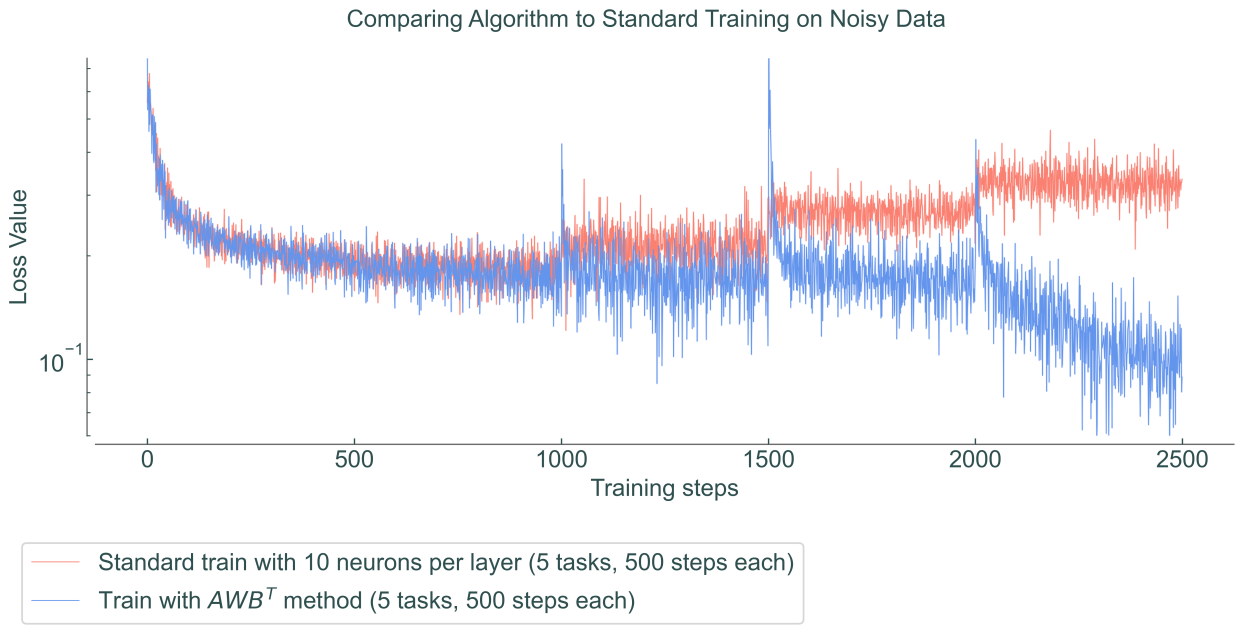

Figure 12: Regression Experiment 3: Comparing loss values on training data for baseline continual learning method and method of learning optimal task architecture when noise in the data is increased

For the third experiment, we learn 5 tasks of data and train 500 epochs on each task, but we have increased the noise in the randomly generated sine dataset. The purpose of this was determine how our proposed method behaves as the noise in the data increases. Figure 12 reveals the loss values as the NN trains (on the training data) with respect to different methods on the five tasks. The red graph is again the standard continual learning method used in Raghavan & Balaprakash (2021), where no change is made to the architecture across tasks. The blue graph shows the loss value when our method of changing the architecture at each step and completing the low rank transfer to retain learning. Since the blue graph displays a decrease in the loss value on the training data in comparison to the baseline continual learning, we conclude that changing the architecture while also transferring learning at each task increases performance even on noisy data.

7.2 Classification

For the classification experiment, we use a convolutional neural network, where we begin with a filter size of three and hidden layers of size 512 and 64. We learn five tasks of data based on the Omniglot data set and train 500 epochs on each task. We use AdamW with a learning rate of 1×10^{-4} in the training regime. Figure 13 reveals the loss values as the CNN trains (on the training data) with respect to different methods on the five tasks. The red graph is the standard continual learning method used in Raghavan & Balaprakash (2021), where no change is made to the architecture across tasks. This serves as our baseline for comparison. The blue graph shows the loss value when our proposed method of changing the architecture at each step and completing the low rank transfer to retain learning. When we determine the architecture at each task, we learn both the optimal filter size and the optimal number of the feedforward neural network. Thus, the low rank transfer is performed for both the weights matrices determined by the filter size and the weights matrices determined by the feedforward neural network.

Since the blue graph displays a decrease in the loss value on the training data in comparison to the baseline continual learning, we can deduce that changing the architecture and transferring learning at each task leads to increased performance in the classification setting as well.

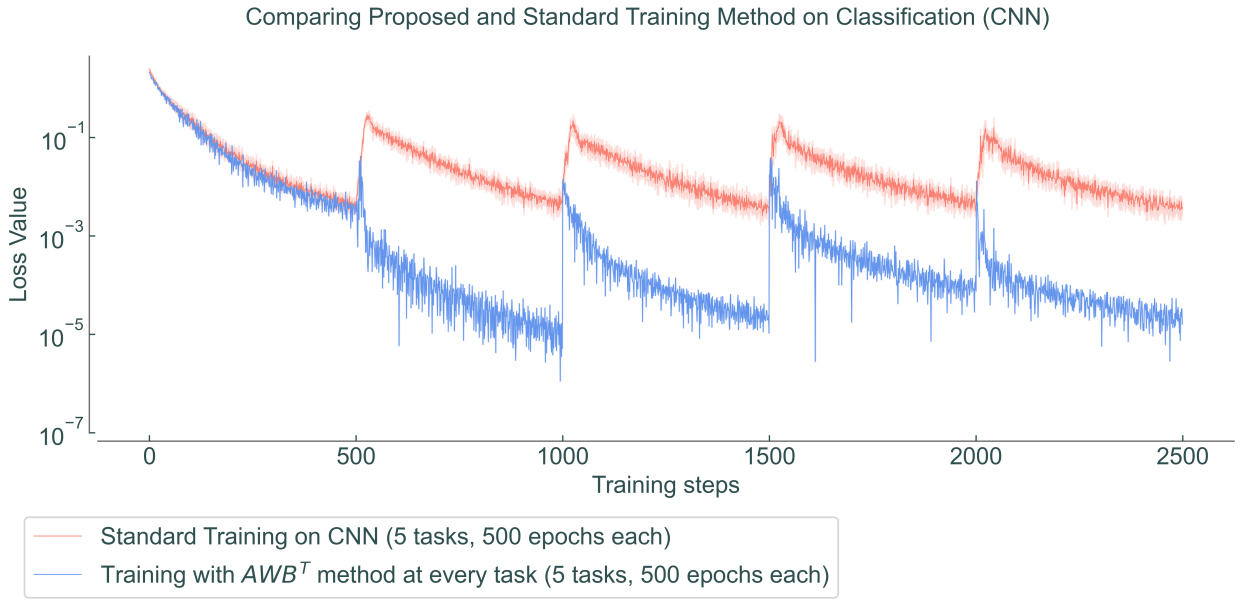

Figure 13: Classification Experiment: Comparing loss values on training data for baseline continual learning method and method of learning optimal task architecture

7.3 Graph Classification

For the graph classification experiment, we use a graph neural network, where we begin with a single GCN layer followed by a feedforward neural network with 4 layer where the hidden layers of size are of size 140. We learn five tasks of data based on the FakeDataset and train 125 epochs on each task. We use AdamW with a learning rate of 1×10^{-4} in the training regime. Figure 14 reveals the loss values as the GNN trains (on the training data) with respect to different methods on the five tasks. The red graph is the standard continual learning method used in Raghavan & Balaprakash (2021), where no change is made to the architecture across tasks. This serves as our baseline for comparison. The blue graph describes the loss value if we change the architecture at each task, but rather than utilizing the low rank transfer method we describe in 2, we randomly reinitialize the weights. The green graph shows the loss value when our proposed method of changing the architecture at each step and completing the low rank transfer to retain learning.

The green graph shows a decrease in the loss value on the training data in comparison to the baseline continual learning, which reveals that changing the architecture and transferring learning at each task leads to increased performance.

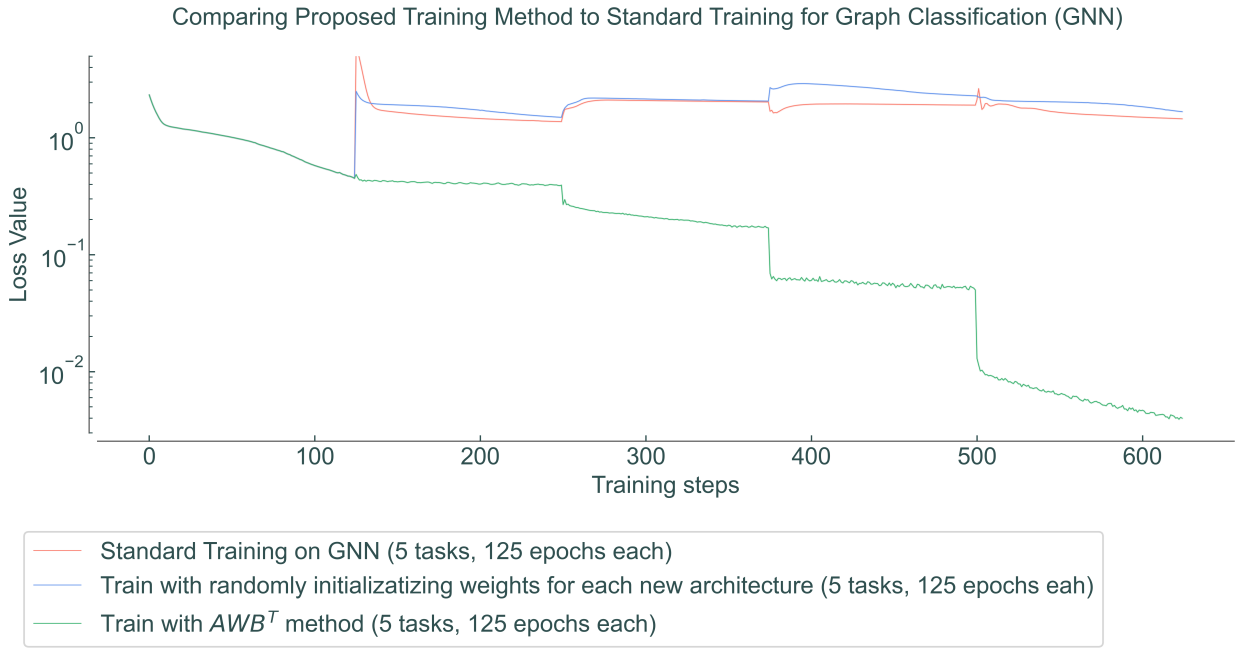

Figure 14: Graph Classification Experiment: Comparing loss values on training data for baseline continual learning method and method of learning optimal task architecture

References

Kunle Adegoke and Olawanle Layeni. The higher derivatives of the inverse tangent function and rapidly convergent BBP-type formulas. URL `http://arxiv.org/abs/1603.08540`.

Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, et al. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024.

Supriyo Chakraborty and Krishnan Raghavan. On understanding of the dynamics of model capacity in continual learning. *arXiv preprint arXiv:2508.08052*, 2025.

Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Society, 2022.

Matteo Gambella, Vicente Javier Castro Solar, and Manuel Roveri. Seal: Searching expandable architectures for incremental learning. *arXiv preprint arXiv:2505.10457*, 2025.

Qiang Gao, Zhipeng Luo, Diego Klabjan, and Fengli Zhang. Efficient architecture search for continual learning. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8555–8565, 2022.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training dnns: Methodology, analysis and application. *IEEE transactions on pattern analysis and machine intelligence*, 45(8):10173–10196, 2023.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. URL http://arxiv.org/abs/1412.6980.

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

Song Lai, Haohan Zhao, Rong Feng, Changyi Ma, Wenzhuo Liu, Hongbo Zhao, Xi Lin, Dong Yi, Min Xie, Qingfu Zhang, et al. Reinforcement fine-tuning naturally mitigates forgetting in continual post-training. *arXiv preprint arXiv:2507.05386*, 2025.

Jeffrey Larson, Matt Menickelly, and Stefan M Wild. Derivative-free optimization methods. *Acta Numerica*, 28:287–404, 2019.

Jessy Lin, Luke Zettlemoyer, Gargi Ghosh, Wen-Tau Yih, Aram Markosyan, Vincent-Pierre Berges, and Barlas Oğuz. Continual learning via sparse memory finetuning. *arXiv preprint arXiv:2510.15103*, 2025.

Sen Lin, Peizhong Ju, Yingbin Liang, and Ness Shroff. Theory on forgetting and generalization of continual learning. In *International Conference on Machine Learning*, pp. 21078–21100. PMLR, 2023.

Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems*, 34(2):550–570, 2021.

Aojun Lu, Hangjie Yuan, Tao Feng, and Yanan Sun. Rethinking the stability-plasticity trade-off in continual learning from an architectural perspective. *arXiv preprint arXiv:2506.03951*, 2025.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 2025.

Scott Mahan, Emily J. King, and Alex Cloninger. Nonclosedness of sets of neural networks in Sobolev spaces. 137:85–96. ISSN 0893-6080. doi: 10.1016/j.neunet.2021.01.007. URL https://www.sciencedirect.com/science/article/pii/S0893608021000150.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Shishir Muralidhara, Didier Stricker, and René Schuster. Clora: Parameter-efficient continual learning with low-rank adaptation. *arXiv preprint arXiv:2507.19887*, 2025.

Ramakanth Pasunuru and Mohit Bansal. Continual and multi-task architecture search. *arXiv preprint arXiv:1906.05226*, 2019.

Philipp Petersen, Mones Raslan, and Felix Voigtlaender. Topological Properties of the Set of Functions Generated by Neural Networks of Fixed Size. 21(2):375–444. ISSN 1615-3383. doi: 10.1007/s10208-020-09461-0. URL https://doi.org/10.1007/s10208-020-09461-0.

Krishnan Raghavan and Prasanna Balaprakash. Formalizing the generalization-forgetting trade-off in continual learning. *Advances in Neural Information Processing Systems*, 34:17284–17297, 2021.

Walter Rudin. Principles of mathematical analysis. *3rd ed.*, 1976.

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.

Hwijae Son, Jin Woo Jang, Woo Jin Han, and Hyung Ju Hwang. Sobolev training for physics informed neural networks. *arXiv preprint arXiv:2101.08932*, 2021.

Nik Weaver. *Measure theory and functional analysis*. World Scientific Publishing Company, 2013.

Martin Wistuba, Prabhu Teja Sivaprasad, Lukas Balles, and Giovanni Zapper. Continual learning with low rank adaptation. *arXiv preprint arXiv:2311.17601*, 2023.

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

A Appendix

You may include other additional sections here.