Differential Learning for Neural Networks

May 12, 2023

Abstract

In this paper, we introduce a novel framework to study the interpretability and control of learning in feedforward neural networks (NN). In this context, interpretability refers to comprehending the behavior of NN parameters in the learning phase whereas control refers to the ability of influencing this behavior. To address these issues, we adopt a system theoretic approach where we model each hidden layer as a dynamical system, a system whose dynamics are governed by differential equations. By employing tools from control theory, we analyze the NN, and present some practical insights that aid in choosing the network structure, and its design parameters. Within this context, we also introduce novel weight tuning laws, and derive sufficient conditions to guarantee stable learning behavior in neural networks with any number of layers. In other words, the framework is demonstrated to be efficient for both shallow and deep NN. In addition, we show that our framework provides insights into well-known methods such as back-propagation, weight perturbation and hidden layer perturbations. Finally, through numerical examples we demonstrate the efficacy of the proposed framework, and the resulting weight tuning rules while highlighting key insights within the context of deep and shallow neural networks.

1 Introduction

Artificial feedforward neural networks (NNs) are a popular tool for dealing with complex decision making problems that require "human-like" perception. Typically, in such problems, NNs are used to approximate an unknown function, referred to as decision making map, using a given data-set. The canonical way to perform this approximation is to define a cost function, and tune the weights of the NN to minimize the cost function. Common approaches used in the literature to tune these weights involve gradient of the cost function, for instance, back-propagation (Werbos 1990; Rumelhart, Hinton, Williams, et al. 1988).

Such gradient-based approaches can suffer from issues such as the vanishing gradients problem (Pascanu, Mikolov, and Bengio 2013), and can lead to local convergence due to weight initialization (Dauphin et al. 2014). Despite the fact that these issues can lead to potentially unsatisfactory solutions (Pascanu, Mikolov, and Bengio 2013; Dauphin et al. 2014; Jaderberg et al. 2016; Schmidhuber 2015), they are still a common choice of weight tuning as there is a lack of structured approaches to design efficient learning regimes.

Even though the process of tuning the weights, and selecting the hyper-parameters for a NN can sometimes be tedious, NNs are quite popular and are ubiquitous in mission-critical tasks such as driverless cars (Tian et al. 2018). However, one of the major criticism associated with the NNs is their opaqueness. For instance, a NN can perform its function by responding correctly to every input asserted to it, however, the user is still not privy to its internal operation. In other words, there is a lack of understanding of how the data is encoded in the hidden layer neurons during the learning process especially for deep NN.

In this paper, we introduce a principled approach to analyze a NN, and design weight tuning rules that are aimed at addressing a few of these issues. Specifically, we provide a framework to analyze and train a NN, both shallow and deep, while facilitating the following: (1) the interpretability - How can we understand the behavior of hidden layer outputs and weights during the learning process; and (2) control- Can we design efficient weight tuning rules with mathematical guarantees to drive the NN parameters to respective targets.

To answer these questions, we first introduce additional variables into the neural network to break the optimization problem across its hidden layers. Subsequently, the modified optimization problem is constructed in such a way that it does not depend on the nesting that is typical of a NN architecture. The modified NN structure is then utilized to derive a differential equation representation for each layer output which results in a system of cascaded differential equations, representing the working of an NN. Therefore, in the proposed framework, the overall NN is not in a nested form. Such a setup allows the possibility of understanding the inner-working of the NN (interpretability of NN) and enables the tractable design of learning rules (control of NN learning).

In particular, we demonstrate the following: (1) **Interpretability:** we first draw parallels between NN learning and ideas from system theory. Using this setup, we also provide insights into a few of the well known methods such as L_2 norm regularization (Sysoev and Burdakov 2019), noise input (Bishop 1995), adversarial regularization (Ororbia II, Kifer, and Giles 2017), batch normalization (Ioffe and Szegedy 2015), flat minima theory (Zhang et al. 2016), vanishing gradients (Pascanu, Mikolov, and Bengio 2013) with weight Wen et al. 2018 and hidden layer perturbations(Zeng and Yeung 2001). (2) **Control**: using Lyapunov analysis, we design learning regimes that guarantees stable learning behavior in NN with any number of layers. We first derive the back-propagation learning rule through our framework and demonstrate the conditions under which back-propagation learning rule is efficient. To demonstrate that the practitioner is not limited to back-propagation learning rule with this framework, and we show efficient learning with an alternate learning rule.

The rest of this paper is organized as follows: in Section 2, we first discuss some of the existing results which address some of the aforementioned challenges, and list out the specific contributions of this paper. We will then provide a brief background on NN learning, and introduce the notations used throughout this paper. In Section 3, our proposed framework to study a NN is presented. Further, in Section **??**, we will derive the learning rules based on Lyapunov stability theory, and discuss several key insights which are revealed by the proposed framework. In Section 5, we present simulation analysis which demonstrates some of the novel features of our proposed design and analysis framework for tuning a feed-forward NN.

2 Related Works and Background

First, we discuss the relevant literature and then provide the problem statement.

2.1 Related Works

Due to the challenges pertaining to the gradient based training methods, and in general, the lack of interpretability with NN learning, there have been a growing interest in understanding the NN learning. For instance, to escape the drawbacks of gradient-based learning, alternate frameworks were proposed in the literature (Lee et al. 2015; Nøkland 2016; Taylor et al. 2016; Carreira-Perpinan and Wang 2014). Despite finding success and eliminating some of the bottlenecks, these approaches (Lee et al. 2015; Nøkland 2016; Taylor et al. 2016; Carreira-Perpinan and Wang 2014) are either computationally expensive (Lee et al. 2015; Taylor et al. 2016) or provide limited applicability Carreira-Perpinan and Wang 2014. Furthermore, in general, these approaches do not provide any theoretical guarantees, and as a consequence, in majority of these approaches, practical applications of NN rely heavily on empirical design and expertise.

On the other hand, the lack of interpretability with NN learning is tightly associated with the non-convexity of the associated optimization problem. This can be attributed to the nested nature of the NN model. To circumvent this problem, there have been efforts to convert the non-convex optimization problem into a convex one (see for example Mobahi and Fisher III 2015; Vese 1999). These approaches involve either using a smoothing parameter (Mobahi and Fisher III 2015) or a predetermined transformation (Vese 1999) which assume *a priori* knowledge of the exact transformation or the smoothing parameter (Mobahi and Fisher III 2015; Vese 1999) that may not be known in many practical situations.

More recently, the work in (Haber and Ruthotto 2017; Chen et al. 2018) proposed to model the residual network using differential equation. Similarly, there have been numerous works that model the behavior of the biological neural network (brain) (Grossberg 1982), as well as artificial NNs, such as the recurrent neural networks (Chang et al. 2019), residual networks (Haber and Ruthotto 2017; Chen et al. 2018), and normalizing flows (Salman et al. 2018) using differential equations. These approaches provide a well-defined premise for analyzing the NN learning using the theory of differential equations. However, the models introduced in (Haber and Ruthotto 2017; Chen et al. 2018) are focused on implementing residual network (He et al. 2016) where the neural network architecture provides a natural structure for differential equation modelling. Since, no such relationships exists in the case feed-forward networks, these models are not generally applicable.

In contrast with (Haber and Ruthotto 2017; Chen et al. 2018), the proposed approach is not confined to a particular NN architecture, and is not application specific. Due to the use of Lyapunov-based approaches to design the learning rules, we can explicitly guarantee the stability of NN learning irrespective of the number of hidden layers, which, to the best of our knowledge, has not been reported. Moreover, in contrast to the works in (Grossberg 1982), and the works in (Chang et al. 2019, Salman et al. 2018), we model the behavior of feed-forward network, a static map, using cascaded differential equations where the internal structure does not automatically render itself to analysis as a dynamical system.

In summary, the contributions of the paper include (1.) Alternate differential equation-based representation of the neural network learning; (2.) Lyapunov techniques based design of learning procedures; and (3.) Mathematical and simulation analysis.

2.2 Background

In this paper, we will present our ideas in the context of supervised learning, wherein we will use the NN as a classifier. However, the proposed framework is not constrained to this application, and can be easily extended to applications such as unsupervised learning, regression, and function approximation in system identification and control, etc. First, we will fix the notations used in the paper.

Let $\mathbb{N}$ and $\mathbb{R}$ denote the set of natural numbers and real numbers, respectively. We will use $\|.\|$ to denote the Euclidean norm for vectors, and Frobenius norm for matrices. We use $\boldsymbol{x} \in R^{p\times n}$ to denote the data sample, where p refers to the number of dimension in the data-set and n refers to the number of data-points. The classical problem in supervised learning is to determine whether a given data sample, say $\boldsymbol{x}$, belongs to one of the $\mathcal{F}$ categories or not. To make this prediction, one of the common approaches is to first estimate a map $\psi(.)$, that transforms the data-point $\boldsymbol{x}$ into a probability space where the magnitude of $\psi(\boldsymbol{x})$ indicates the membership of $\boldsymbol{x}$ to different categories. The membership values signify the probability of $\boldsymbol{x}$ belonging to any particular category. To learn $\psi(.)$ in the context of supervised learning, one usually assumes that a data-set $\boldsymbol{\mathcal{X}}$ with labels $\boldsymbol{\mathcal{Y}}$ is available, and it represents all the necessary information. The overall learning then involves approximating $\psi(.)$ through a parametric map such as neural networks, spline functions, etc (Bishop 2006).

In this paper, NNs are employed as the parametric map of choice. Let d denote the total number of layers in the NN. The learning process is composed of the learning phase and the testing phase. Let $t \in [0, T] \subseteq \mathbb{R}$ denote a time instant (or sampling instant) in the learning phase, where the interval $[0, T]$ represents the duration of the learning phase. Let the NN output be denoted as $\hat{\boldsymbol{y}}(\boldsymbol{x})$. with estimated weights denoted as $\hat{\boldsymbol{\theta}} = [\hat{\boldsymbol{W}}^{(1)} \cdots \hat{\boldsymbol{W}}^{(d)}]^T$, such that

$$\hat{\boldsymbol{y}}_t = \hat{\boldsymbol{y}}(\boldsymbol{x}_t; \hat{\boldsymbol{\theta}}) = \phi^{(d)}(\hat{\boldsymbol{W}}^{(d)T} \cdots (\phi^{(1)}(\hat{\boldsymbol{W}}^{(1)T}\boldsymbol{x}_t)))), \tag{1}$$

where, the superscripts denote the layer index, for instance, $\phi^{(d)}$ is the activation function of the d^{th} layer in the NN, $\hat{W}^{(d)}$ is the estimated weights of the d^{th} layer and $\boldsymbol{x}$ denotes the input data while $\phi^i(.), \forall i = 1, 2, 3, \cdots$ denote the activation functions of choice.

To estimate $\psi(.)$ using the NN, one would usually minimize the difference between the NN output (estimated parametric map), and the labels $\boldsymbol{y}_t$ at each time instant (or sampling instant) t such that this difference goes to zero as $t \to \infty$. Let us assume that for each t, we are provided a data-sample $\boldsymbol{x}_t$ and the corresponding labels $\boldsymbol{y}_t$ from the training data-set. In this context, we can define a loss function $\ell(\boldsymbol{y}_t, \hat{\boldsymbol{y}}(t))$ such that it represents the difference between the known labels and the estimated NN output. The overall cost $\acute{J}$ is then written as $\acute{J} = E[\ell(\boldsymbol{y}_t, \hat{\boldsymbol{y}})]$. The term $E[\ell(\boldsymbol{y}_t, \hat{\boldsymbol{y}}(t))]$ can denote any generic loss function, for example, a quadratic (L_2 norm-based) loss or a cross-entropy loss function. Furthermore, E refers to the expected value operator.

Without loss of generality, in this paper, a quadratic cost function is considered such that $\acute{J}(t) = \frac{1}{2}E[\|\boldsymbol{y}_t - \hat{\boldsymbol{y}}(\boldsymbol{x}_t)\|] = \frac{1}{2}E[r]^T E[r]$, where $\boldsymbol{r} = \boldsymbol{y}_t - \hat{\boldsymbol{y}}(\boldsymbol{x}_t)$. Note the expected value on the cost function is calculated over the complete data-set. The optimization problem then pertains to finding the NN weights $\boldsymbol{\theta}^*$ such that

$$\boldsymbol{\theta}^* = \underset{\hat{\boldsymbol{\theta}} \in \Omega}{\arg\min}\, \acute{J}(t), \tag{2}$$

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

One may observe from (1), that due to the presence of nested nonlinear function, it is difficult to characterize the behavior of hidden layers during the learning process. One of the main contributions of this work is to explicitly derive the dynamics of the neural network that helps in characterizing this behavior, which is discussed next.

3 Differential Learning Approach

In this section, we reformulate the optimization problem described in the previous section such that the nested structure is decomposed into a system of differential equations. First, we introduce a new variable $\boldsymbol{z}^{(d)}$ to represent the output of the top-most hidden layer such that

$$\acute{J}(\hat{\boldsymbol{y}}; \hat{\boldsymbol{\theta}}) = J(\hat{\boldsymbol{y}}(\boldsymbol{z}^{(d)}); \hat{\boldsymbol{\theta}}) = E[\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}(\boldsymbol{z}^{(d)})], \tag{3}$$

where $\hat{\boldsymbol{y}}(\boldsymbol{z}^{(d)}) = \phi^{(d)}(\hat{\boldsymbol{W}}^{(d)}\boldsymbol{z}^{(d)})$ is the output of the NN that depends on $\boldsymbol{z}^{(d)}$. It follows that the loss function depends on $\boldsymbol{z}^{(d)}$, and a constraint $\boldsymbol{z}^{(d)}(\boldsymbol{z}) = \phi^{(d-1)}(\boldsymbol{x})$, is introduced to assert equivalence with the traditional cost function. Next, introduce another variable $\boldsymbol{z}^{(d-1)}$ that is constrained to mimic the penultimate layer in the NN such that the topmost layer depends on $\boldsymbol{z}^{(d-1)}$. Therefore, the constraint on variable $\boldsymbol{z}^{(d)}$ is rewritten as $\boldsymbol{z}^{(d)} = \phi^{(d-1)}(\hat{\boldsymbol{W}}^{(d-1)}\boldsymbol{z}^{(d-1)})$ which introduces an additional constraint as $\boldsymbol{z}^{(d-1)} = \phi^{(d-2)}(\hat{\boldsymbol{W}}^{(d-2)}\boldsymbol{z}^{(d-2)})$. Following this procedure, one may define a variable at each layer of the NN constrained to mimic all the information prior to that particular layer in the NN, and as a result, the overall optimization problem is rewritten as

$$\boldsymbol{\theta}^* = \underset{\hat{\theta} \in \Omega}{\arg\min}\, J(\hat{\boldsymbol{y}}(\boldsymbol{z}^{(d)}); \hat{\boldsymbol{W}}^{(d)})$$

$$\text{subject to} \quad \boldsymbol{z}^{(d)} = \phi^{(d-1)}(\hat{\boldsymbol{W}}^{(d-1)T}\boldsymbol{z}^{(d-1)}), \cdots\cdots, \boldsymbol{z}^{(2)} = \phi^{(1)}(\hat{\boldsymbol{W}}^{(1)T}\boldsymbol{z}^{(1)}), \tag{4}$$

where $\boldsymbol{z}^{(1)} = \boldsymbol{x}_t$.

Remark 3: A natural question that ought to be asked here is whether solving the optimization problem in (2) is equivalent to solving the optimization problem in (4). It can be observed from (4) that the constraints on the optimization are *hard constraints*, and when all these constraints are satisfied, the optimization problem in (2) can be obtained from (4) by replacing $\boldsymbol{z}^{(i)}, i = 1, \cdots, d$ with $\phi^{(i)}(\hat{\boldsymbol{W}}^{(i)}\boldsymbol{z}^{(i-1)})$.

Since solving equation (4) is equivalent to solving (2), we can derive a differential equation representation for the evolution of the cost function from (2) which will characterize the evolution of the optimization described in Eq. (4). The resultant NN learning dynamics are obtained as

$$\begin{aligned}
\dot{J}(\hat{\boldsymbol{y}}(\boldsymbol{z}^{(d)}); \hat{\boldsymbol{\theta}}) &= \boldsymbol{r}^T\dot{\boldsymbol{r}} \\
\dot{\boldsymbol{r}} &= -\delta\phi^{(d)}(\hat{\boldsymbol{W}}^{(d)T}\boldsymbol{z}^{(d)})[\dot{\hat{\boldsymbol{W}}}^{(d)T}\boldsymbol{z}^{(d)} + \hat{\boldsymbol{W}}^{(d)T}\dot{\boldsymbol{z}}^{(d)}] \\
\dot{\boldsymbol{z}}^{(d)} &= \big(\delta\phi^{(d-1)}(\hat{\boldsymbol{W}}^{(d-1)T}\boldsymbol{z}^{(d-1)})\big)[\dot{\hat{\boldsymbol{W}}}^{(d-1)T}\boldsymbol{z}^{(d-1)} + \hat{\boldsymbol{W}}^{(d-1)T}\dot{\boldsymbol{z}}^{(d-1)}] \\
&\vdots \\
\dot{\boldsymbol{z}}^{(2)} &= \big(\delta\phi^{(1)}(\hat{\boldsymbol{W}}^{(1)T}\boldsymbol{z}^{(1)})\big)[\dot{\hat{\boldsymbol{W}}}^{(1)T}\boldsymbol{z}^{(1)} + \hat{\boldsymbol{W}}^{(1)T}\dot{\boldsymbol{z}}^{(1)}].
\end{aligned} \tag{5}$$

where $\dot{\boldsymbol{z}}^{(1)} = \dot{\boldsymbol{x}}$ denotes taking the derivative of the variable (input data) $\boldsymbol{x}$ with respect to time, $\delta\phi^{(i)}, \forall i = 1, \cdots d$ are diagonal matrices of appropriate dimensions which denote the gradient of the activation functions. The term $\boldsymbol{r}$ denotes classification error while $\boldsymbol{z}^{(i)}$ denotes the hidden layer states. Finally, $\hat{\boldsymbol{W}}^{(i)}$ and $\dot{\hat{\boldsymbol{W}}}^{(i)}$ are the weight matrices and their respective derivatives.(for more details and the exact derivation, please refer to Appendix)

In a traditional NN setup, at every learning instant t, the input, i.e., the given dataset, is first transformed through the first layer in the NN. Similarly, within the proposed formulation, one may observe that the hidden layer representation $\dot{\boldsymbol{z}}^{(2)}$ depends on the input $\boldsymbol{z}^{(1)}$.

Similarly, the hidden representation at each layer depends on the layer below. In other words, the nesting that is present in a standard NN representation takes the form of cascaded differential equations with the proposed setup. Thus, the hidden layer representation can be exactly calculated by solving a differential equation using the input $\boldsymbol{z}_t$. However, with the introduction of additional variables, there is an increased computational cost to this approach. It is also worth noting that the variable $\dot{\boldsymbol{z}}_1$ denotes the sampling dynamics of the data (the idea is elaborated further in Section **??**). Furthermore, the system of equation presented here depend on $\boldsymbol{y}_t$ and $\boldsymbol{x}_t$, which denotes the label and the input data at sampling instant t.

In the next section, we will use the tools from dynamical system literature to understand and simplify the two key issues in NN learning - interpretability, and control of NN learning.

Theorem 1. *Check the copied one below and pls fill in the blanks... Let the dynamics of the neural network be given as shown in* (5) *and define* $\Gamma^{(i)} = \big[\prod_{j=i+1}^{d-1} \delta\phi^{(j)T}\hat{\boldsymbol{W}}^{(j+1)}\big]$. *Next, consider the condition that* $\forall t \in [0, T]$, $\Gamma^{(i)T}\Gamma^{(i)} > 0, \forall i = 1, 2, 3, \cdots d$ *and*

$\boldsymbol{z}^{(i)T}\boldsymbol{z}^{(i)} > 0, \forall i = 1, 2, 3, \cdots d$. Consider the learning signals as

$$\dot{\hat{\boldsymbol{W}}}^{(d)} = \alpha^d \boldsymbol{z}^{(d)} \boldsymbol{r}^T \delta\phi^{(d)T},$$
$$\dot{\hat{\boldsymbol{W}}}^{(i)T} = \alpha^i \boldsymbol{z}^{(i)} \boldsymbol{r}^T \delta\phi^{(d)} [\prod_{j=d-1}^{i+1} \hat{\boldsymbol{W}}^{(j+1)T} \delta\phi^{(j)}]$$

where $\boldsymbol{W}^{(i)} \in \mathbb{R}^{\eta^{i-1} \times \eta^i}$ with η^i representing the number of neurons in each hidden layer. Furthermore, $\delta\phi^{(i)} \in \mathbb{R}^{\eta^i \times \eta^i}$ is a diagonal matrix with $\boldsymbol{r} \in \mathbb{R}^{\eta^(d) \times 1}$ and $\boldsymbol{z}^{(i)^T} \in \mathbb{R}^{\eta^i \times 1}$ with $\alpha^i, \forall i$ is the learning rate. It follows that $\dot{V} < 0$ for all t and therefore, $r \to 0$ with $t \to \infty$ and the equilibrium point is locally asymptotically stable. Refer to appendix for proof.

Theorem 2. *Consider the training process of the neural network represented as a constrained nonlinear program as in ..... Let the sampling rate of the input training data of the neural network be represented as ..$\dot{x}$...?, inducing a learning dynamics given by ..... Define a scalar function, V....., such that $V(\boldsymbol{r}) \geq 0$ with $V(\boldsymbol{r}) = 0$ only when $\boldsymbol{r} = 0$, where $\boldsymbol{r} \in \mathbb{R}^{\eta^{(d)} \times 1}$ is the training error. Let the weights of the neural network, initialized from a finite interval (can be specific), and updated as*

$$\dot{\hat{\boldsymbol{W}}}^{(d)} = \alpha^d \boldsymbol{z}^{(d)} \boldsymbol{r}^T \delta\phi^{(d)T},$$
$$\dot{\hat{\boldsymbol{W}}}^{(i)T} = \alpha^i \boldsymbol{z}^{(i)} \boldsymbol{r}^T \delta\phi^{(d)} [\prod_{j=d-1}^{i+1} \hat{\boldsymbol{W}}^{(j+1)T} \delta\phi^{(j)}],$$

where $\boldsymbol{z}^{(i)^T} \in \mathbb{R}^{\eta^i \times 1}$ with α^i, $i = 1, 2, \ldots$ is the learning rate and $\delta\phi^{(i)} \in \mathbb{R}^{\eta^i \times \eta^i}$ is a diagonal matrix. Then, $r \to 0$ as $t \to \infty$ (what about $\tilde{W}^{(d)}$), rendering the equilibrium point at origin locally asymptotically stable, provided that the condition $\Gamma^{(i)T}\Gamma^{(i)} > 0, \forall i = 1, 2, 3, \cdots d$ and $\boldsymbol{z}^{(i)T}\boldsymbol{z}^{(i)} > 0, \forall i = 1, 2, 3, \cdots d$, holds for $t \in [0, T]$, where $\Gamma^{(i)} = \left[\prod_{j=i+1}^{d-1} \delta\phi^{(j)T} \hat{\boldsymbol{W}}^{(j+1)}\right]$, where $\boldsymbol{W}^{(i)} \in \mathbb{R}^{\eta^{i-1} \times \eta^i}$ with η^i representing the number of neurons in each hidden layer.

Refer to appendix for proof.

Remark 3: Theorem 2 is valid for any number of layers in the network. NNs have been used as function approximators as part of closed loop control, in both discrete and continuous time domain. In continuous time domain, Lyapunov stability has been shown when the number of hidden layers is two (Lewis, Jagannathan, and Yesildirak 1998) whereas for discrete time systems (Jagannathan and Lewis 1996), it has been shown that the closed loop is stable for any number of layers. The primary distinction is that in (Lewis, Jagannathan, and Yesildirak 1998; Jagannathan and Lewis 1996), the NN was utilized as part of closed loop control of a dynamical system whereas in this paper, the NN itself is treated as a dynamical system. From Theorem 2, many interesting insights can be derived about well known heuristics in the literature.

Weight and Hidden Layer Perturbation There are two conditions under which Theorem 2 is valid. The first condition is that the layer-wise inputs must have a non-zero norm. This means that the hidden layer representations must consistently provide information (non-zero value of the hidden layer output) to the learning process. The idea can be tied to persistence of excitation condition (PE condition) from systems theory, where PE condition refers to the positive-definiteness of the covariance function of the data at each hidden layer (Moore 1983) . PE condition is typically satisfied through noise perturbations in the input. (Zeng and Yeung 2001) showed that hidden layer perturbation would improve the convergence, which would satisfy the PE condition.

The second condition, $\|\hat{\boldsymbol{W}}^{(i)}\| > 0, \forall i = 1, 2, 3.$ implies that the weight matrices in the learning phase must be sufficiently rich too, which again refers to the idea of PE condition and can be satisfied using weight perturbation (Wen et al. 2018).

Vanishing Gradients: The case $\|\delta\phi^{(i)}\| = 0, \forall i = 1, 2, \cdots d$ implies that the gradients vanish from the learning learning phase. Furthermore, it is also possible that $[\prod_{j=d}^{i+1} \hat{\boldsymbol{W}}^{(j)T} \delta\phi^{(j)T}]$ goes to zero as the number of layers in the network increase, which is the issue of vanishing gradients. Theorem 1 demonstrates that convergence for back-propagation learning rule only holds under the condition that $\|\delta\phi^{(i)}\| \neq 0$ and the singular values of $[\prod_{j=d}^{i+1} \hat{\boldsymbol{W}}^{(j)T} \delta\phi^{(j)T}]$ are all zero.

We have shown that commonly known issues with back-propagation are actually the conditions under which the convergence of back-propagation is guaranteed Pascanu, Mikolov, and Bengio 2013. Another important advantage of the proposed approach is that, the practitioner is not limited by the back-propagation update rule. Below, we demonstrate another learning rule.

3.0.1 Other Learning Rules

First, we will state the theorem where the learning rule is derived.

Theorem 3. *Let the dynamics of the neural network be given as shown in* (5). *Consider the learning signal for NN weight tuning given by*

$$\dot{\hat{\boldsymbol{W}}}^{(d)T} = \alpha^d \delta\phi^{(d)-T} \boldsymbol{r}\boldsymbol{z}^{(d)T}, \tag{6}$$
$$\dot{\hat{\boldsymbol{W}}}^{(i)T} = \alpha^i \delta\phi^{(i)-T} [\prod_{j=d}^{i+1} \hat{\boldsymbol{W}}^{(j)T} \delta\phi^{(j)-T}] \boldsymbol{r}\boldsymbol{z}^{(i)^T}$$

Next, consider the condition such that $\|\boldsymbol{z}^{(i)}\| > 0, \forall i = 1, 2, 3$ *and* $\|\hat{\boldsymbol{W}}^{(i)}\| > 0, \forall i = 1, 2, 3.$ *It can be observed that* $\dot{V} < 0$ *for all* t *and therefore,* $r \to 0$ *with* $t \to \infty$. *Refer to Appendix for Proof*

Theorem 3 is valid when $\|\boldsymbol{z}\| > 0, \forall i = 1, 2, \cdots d$ which implies that each neuron in the hidden layer provides a non-zero output. The condition can be satisfied using noise perturbations (Daneshmand et al. 2018). Theorem 3 is also valid for any number of hidden layers in the NN. As demonstrated earlier, the second condition that is $\|\hat{\boldsymbol{W}}^{(i)}\| > 0, \forall i = 1, 2, 3.$ can be satisfied through weight perturbation (Wen et al. 2018).

In this section we have introduced a structured approach that can be used to design learning rules depending on the application and the information that is available on the data. To use the proposed approach in practical scenarios, we provide some results next.

4 Robot Application

In this paper, standard mathematical notations are used. For a vector x, $\|x\|$ denotes the Euclidean-norm while for a matrix A, $\|A\|$ denotes the Frobenius norm. The smallest singular value is denoted by λ_{min}, and the right limit operator is denoted as x^+, i.e., $x^+ = \lim_{r\downarrow t} x(r)$. We use t to denote the time instants when the controller has access to the sensor samples and has to generate a control input for the system.

The $n-$link robot manipulator dynamics with rigid links Lewis, Jagannathan, and Yesildirak 1998 are expressed as

$$M(q)\ddot{q}(t) + V_m(q,\dot{q})\dot{q}(t) + G(q) + F(\dot{q}) + \tau_d(t) = \tau(t) \tag{7}$$

where $q(t) \in \mathbb{R}^n$ is the joint variable vector, $V_m(q,\dot{q})$ is the Coriolis/centripetal matrix, $M(q)$ is the inertia matrix, $F(\dot{q})$ models the effects of friction, $G(q)$ is the gravity vector, and $\tau_d(t)$ represents disturbances. The control input vector $\tau(t)$ has components of torque for revolute joints and force for prismatic joints. With the following facts, the tracking control problem for the robot dynamics as in (7) is introduced.

Facts Lewis, Jagannathan, and Yesildirak 1998: The matrix $M(q)$ satisfies $B_{m_1}I \leq M(q) \leq B_{m_2}I$, for some known positive constants B_{m_1}, B_{m_2}, where I is the identity matrix of appropriate dimension, and the Coriolis/centripetal matrix $V_m(q,\dot{q})$ is bounded such that $\|V_m(q,\dot{q})\| \leq V_b \|\dot{q}\|$ with $V_b > 0$. There exists constants $g_B, B_F, B_f > 0$ such that $\|G(q)\| \leq g_B$, and $\|F(\dot{q})\| \leq B_F \|\dot{q}\| + B_f$. The matrix $\dot{M} - 2V_m$ is skew- symmetric, and finally, there exists a positive constant, $\tau_{dM} > 0$, such that $\|\tau_d(t)\| \leq \tau_{dM}$, uniformly for all $t \in \mathbb{R}^+$.

By defining the joint positions and velocities as the state vector, the robot manipulator system dynamics can be represented in a compact form Lewis, Jagannathan, and Yesildirak 1998 as

$$\begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ -M^{-1}(q)(V_m(q,\dot{q})\dot{q} + F(\dot{q}) + G(q)) \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1}(q) \end{bmatrix} \tau(t) + \begin{bmatrix} 0 \\ -M^{-1}(q) \end{bmatrix} \tau_d(t). \tag{8}$$

The objective in a tracking control problem is to design the torque input $\tau(t)$ such that the joint positions track a desired trajectory $q_d(t)$. The desired trajectory is restricted to satisfy the following standard assumption.

Assumption 1. kim1999neural; *Lewis, Jagannathan, and Yesildirak 1998 Define $q_d(t) \in \mathbb{R}^n$ such that $\|Q_d\| \leq q_B$, with $Q_d(t) = [q_d(t)\ \dot{q}_d(t)\ \ddot{q}_d(t)]^T$, and $q_B > 0$.*

In real-world applications, for example in manufacturing plants, the robotic manipulator is expected to perform tasks which are characterized by position, velocity and acceleration that are bounded. Therefore, it is reasonable to make the assumption stated above in the design and analysis.

To develop a controller, that enables the robot manipulator to track a desired trajectory, begin by defining the tracking error as

$$e(t) = q_d(t) - q(t), \tag{9}$$

and the filtered tracking error Lewis, Jagannathan, and Yesildirak 1998 as

$$r(t) = \dot{e}(t) + \lambda e(t), \tag{10}$$

with a symmetric matrix $\lambda > 0$ of compatible dimension.

Observe from (10) that a positive choice of λ ensures the stability of $\dot{e}(t)$ if the filtered tracking error is finite and bounded. To design the torque input and to ensure boundedness of the filtered tracking error, differentiate (10) with respect to time to reveal, $\dot{r}(t) = \ddot{e}(t) + \lambda \dot{e}(t)$. By incorporating the dynamics of the robotic manipulator (7) and utilizing (9), (10), the dynamics of the filtered tracking error is obtained Lewis, Jagannathan, and Yesildirak 1998 as

$$M\dot{r}(t) = -V_m r(t) - \tau(t) + f(x) + \tau_d(t) \tag{11}$$

with $f(z) = M(q)(\ddot{q}_d + \lambda\dot{e}) + V_m(q,\dot{q})(\dot{q}_d + \lambda e) + F(\dot{q}) + G(q)$ and $x = [e^T \; \dot{e}^T \; q_d^T \; \dot{q}_d^T \; \ddot{q}_d^T]^T$. If the function $f(z)$ is known, it is straight-forward to generate a control input, $\tau(t) = f(z) + k_v r(t)$, where $k_v \in \mathbb{R}^{n\times n}$ is a positive definite design matrix, such that the filtered tracking error dynamics in (11) is stable. However, the nonlinear function $f(z)$ is unknown due to the nonlinearities such as friction, inertia matrix, and so on and we would like to approximate this nonlinear function.

The computed torque is designated as

$$\tau = \hat{f}(x) + k_v r \tag{12}$$

where $\hat{f}(x)$ denotes an estimate of $f(x)$. By using (12) in (11), we get

$$M\dot{r} = -V_m r - \hat{f}(x) - k_v r + f(x) + \tau_d. \tag{13}$$

The filtered tracking error dynamics (31) reveals that the tracking performance of the robotic manipulator due to the mismatch between $f(x)$ and its estimate.

To estimate, $f(x)$, deep NNs are employed. Consider d denote the total number of layers in the NN. The learning process is composed of the learning phase and the testing phase. Let $t \in [0, T] \subseteq \mathbb{R}$ denote a time instant (or sampling instant) in the learning phase, where the interval $[0, T]$ represents the duration of the learning phase. Let the ideal NN output be denoted as $\boldsymbol{y}(\boldsymbol{x})$. with estimated weights denoted as $\boldsymbol{\theta} = [\boldsymbol{W}^{(1)} \cdots \boldsymbol{W}^{(d)}]^T$, such that

$$\hat{f} \approx \hat{f}(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{W}^{(d)T}\phi^{(d-1)}(\cdots(\phi^{(1)}(\boldsymbol{W}^{(1)T}\boldsymbol{x})))) + \varepsilon(t), \tag{14}$$

where, the superscripts denote the layer index, for instance, $\phi^{(d)}$ is the activation function of the d^{th} layer in the NN, $\hat{W}^{(d)}$ is the estimated weights of the d^{th} layer and $\boldsymbol{x}$ denotes the input data while $\phi^i(.), \forall i = 1, 2, 3, \cdots$ denote the activation functions of choice. Furthermore, $\varepsilon(t)$ denotes the approximation error. Let the estimated NN output be denoted as $\hat{\boldsymbol{y}}(\boldsymbol{x})$. with estimated weights denoted as $\hat{\boldsymbol{\theta}} = [\hat{\boldsymbol{W}}^{(1)} \cdots \hat{\boldsymbol{W}}^{(d)}]^T$, such that

$$\hat{f} \approx \hat{f}(\boldsymbol{x}; \hat{\boldsymbol{\theta}}) = \phi^{(d)}(\hat{\boldsymbol{W}}^{(d)T} \cdots (\phi^{(1)}(\hat{\boldsymbol{W}}^{(1)T} \boldsymbol{x}_t)))), \tag{15}$$

where, the superscripts denote the layer index, for instance, $\phi^{(d)}$ is the activation function of the d^{th} layer in the NN, $\hat{W}^{(d)}$ is the estimated weights of the d^{th} layer and $\boldsymbol{x}$ denotes the input data while $\phi^i(.), \forall i = 1, 2, 3, \cdots$ denote the activation functions of choice.

First, we introduce a new variable $\boldsymbol{z}^{(d)}$ to represent the output of the top-most hidden layer such that

$$\hat{f} = \hat{f}(\boldsymbol{z}^{(d)}) \tag{16}$$

where

$$\hat{f}(\boldsymbol{z}^{(d)}) = \hat{\boldsymbol{W}}^{(d)} \boldsymbol{z}^{(d)}$$

is the output of the NN that depends on $\boldsymbol{z}^{(d)}$ and a constraint

$$\boldsymbol{z}^{(d)} = \phi^{(d-1)}(\hat{\boldsymbol{W}}^{(d-1)T} \cdots (\phi^{(1)}(\boldsymbol{W}^{(1)T} \boldsymbol{x}))))$$

, is introduced to assert equivalence with the traditional structure of the NN. Next, introduce another variable $\boldsymbol{z}^{(d-1)}$ that is constrained to mimic the penultimate layer in the NN such that the topmost layer depends on $\boldsymbol{z}^{(d-1)}$. Therefore, the constraint on variable $\boldsymbol{z}^{(d)}$ is rewritten as

$$\boldsymbol{z}^{(d)} = \phi^{d-1}(\hat{\boldsymbol{W}}^{(d-1)T} \boldsymbol{z}^{(d-1)})$$

which introduces an additional constraint as

$$\boldsymbol{z}^{(d-1)} = \phi^{(d-2)}(\hat{\boldsymbol{W}}^{(d-2)T} \boldsymbol{z}^{(d-2)})$$

. Following this procedure, one may define an auxiliary variable at each layer of the NN constrained to mimic all the information prior to that particular layer in the NN, and as a result, the overall optimization problem is rewritten as

$$\begin{aligned} &\hat{f}(\boldsymbol{z}^{(d)}) = \hat{\boldsymbol{W}}^{(d)} \boldsymbol{z}^{(d)} \\ \text{subject to} \quad &\boldsymbol{z}^{(d)} = \phi^{d-1}(\hat{\boldsymbol{W}}^{(d-1)T} \boldsymbol{z}^{(d-1)}), \cdots\cdots, \boldsymbol{z}^{(3)} = \phi^{(2)}(\hat{\boldsymbol{W}}^{(2)T} \boldsymbol{z}^{(2)}), \end{aligned} \tag{17}$$

where $\boldsymbol{z}^{(2)} = \phi^{(1)}(\hat{\boldsymbol{W}}^{(1)T} x)$.

A natural question that ought to be asked here is whether solving the optimization problem in (17) is equivalent to solving the optimization problem in (15). It can be observed from (17) that the constraints on the optimization are *hard constraints*, and when all these constraints are satisfied, the optimization problem in (15) can be obtained from (17) by replacing $\boldsymbol{z}^{(i)}, i = 1, \cdots, d$ with $\phi^{(i)}(\hat{\boldsymbol{W}}^{(i)} \boldsymbol{z}^{(i-1)})$.

With this context, it must be understood the estimated neural network and the original neural network are only equivalent when the constraints are satisfied. Therefore, any algorithm with the introduction of z variables must have two loops, first loop to satisfy the constraints and the second loop to generate control input. Therefore, at each t, we must go through an iterative update to satisfy the constraints on $\boldsymbol{z}$. We may write the converged version of z as $z_\infty$and the network is then given as

$$\begin{aligned} &\hat{f}(\boldsymbol{z}^{(d)}_\infty) = \hat{\boldsymbol{W}}^{(d)} \boldsymbol{z}^{(d)}_\infty \\ \text{subject to} \quad &\boldsymbol{z}^{(d)}_\infty = \phi^{(d-1)}(\hat{\boldsymbol{W}}^{(d-1)T} \boldsymbol{z}^{(d-1)}_\infty), \cdots\cdots, \boldsymbol{z}^{(3)}_\infty = \phi^{(2)}(\hat{\boldsymbol{W}}^{(2)T} \boldsymbol{z}^{(2)}_\infty), \end{aligned} \tag{18}$$

where $\boldsymbol{z}^{(2)}_\infty = \phi^{(1)}(\hat{\boldsymbol{W}}^{(1)T} \boldsymbol{x})$. The estimated version of the network is then given as

$$\begin{aligned} &\hat{f}(\boldsymbol{z}^{(d)}) = \hat{\boldsymbol{W}}^{(d)} \hat{\boldsymbol{z}}^{(d)} \\ \text{subject to} \quad &\hat{\boldsymbol{z}}^{(d)} = \boldsymbol{z}^{(d)}_\infty, \cdots\cdots, \hat{\boldsymbol{z}}^{(3)}_j = \hat{\boldsymbol{z}}^{(3)}_\infty, \end{aligned} \tag{19}$$

Let us now define the estimation errors for the z variables as

$$r^{(i)}_{\boldsymbol{z}} = (\boldsymbol{z}^{(i)}_\infty - \hat{\boldsymbol{z}}^{(i)}_j) \tag{20}$$

where the constraints $\forall i = 2, \cdots d$ as

$$\hat{\boldsymbol{z}}^{(i)}_\infty = \phi^{(i-1)}(\hat{\boldsymbol{W}}^{(i-1)T} \hat{\boldsymbol{z}}^{(i-1)}_\infty) \tag{21}$$

then

$$\dot{\hat{\boldsymbol{z}}}^{(i)}_\infty = \Sigma^{(i-1)}[\dot{\hat{\boldsymbol{W}}}^{(i-1)T} \hat{\boldsymbol{z}}^{(i-1)}_\infty)] \tag{22}$$

where with $\dot{\hat{\boldsymbol{W}}}^{(i)} = -\left[\boldsymbol{z}^{(i)}_\infty r^{(i)T}_z \Sigma^{(i-1)}\right]$ we write

$$\dot{\hat{\boldsymbol{z}}}^{(i)}_\infty = -\Sigma^{(i-1)}\Sigma^{(i-1)T} r^{(i)}_z \boldsymbol{z}^{(i-1)T}_\infty \boldsymbol{z}^{(i-1)}_\infty \tag{23}$$

$$= -\Sigma^{(i-1)}\Sigma^{(i-1)T} r^{(i)}_z \boldsymbol{z}^{(i-1)T}_\infty \boldsymbol{z}^{(i-1)}_\infty] \tag{24}$$

Since, $\dot{r}^{(i)}_{\boldsymbol{z}} = (\boldsymbol{z}^{(i)}_\infty - \hat{\boldsymbol{z}}^{(i)}_j)$, we get

$$\dot{r}^{(i)}_{\boldsymbol{z}} = [-\Sigma^{(i-1)}\Sigma^{(i-1)T} r^{(i)}_z \boldsymbol{z}^{(i-1)T}_\infty \boldsymbol{z}^{(i-1)}_\infty - \dot{\hat{\boldsymbol{z}}}^{(i)}_j)] \tag{25}$$

Let $\dot{\hat{\boldsymbol{z}}}^{(i)}_j = [-\Sigma^{(i-1)}\Sigma^{(i-1)T} r^{(i)}_z \boldsymbol{z}^{(i-1)T}_\infty \boldsymbol{z}^{(i-1)}_\infty + K^{(i)}_z \times r^{(i)}_{\boldsymbol{z}}]$ whence substitution and cancellation of common terms provides

$$\dot{r}^{(i)}_{\boldsymbol{z}} = -[K^{(i)}_z \times r^{(i)}_{\boldsymbol{z}}]] \tag{26}$$

where $\Sigma^{(d-1)}$ is a diagonal matrix of the first derivatives of the layer wise activation functions. Let us now define the weight estimation error as

$$\tilde{\boldsymbol{W}}^{(i)} = (\boldsymbol{W}^{(i)} - \hat{\boldsymbol{W}}^{(i)}) \tag{27}$$

thus providing the derivative as

$$\dot{\tilde{\boldsymbol{W}}}^{(i)} = -\dot{\hat{\boldsymbol{W}}}^{(i)} \tag{28}$$

Next, we will write the updates for the weights as

$$\begin{aligned}\dot{\tilde{\boldsymbol{W}}}^{(d)} &= -\dot{\hat{\boldsymbol{W}}}^{(d)} = -z_{\infty}^{(d)} r^T + \sigma \times \hat{\boldsymbol{W}}^{(i)} \\ \dot{\tilde{\boldsymbol{W}}}^{(i)} &= -\dot{\hat{\boldsymbol{W}}}^{(i)} = \left[\boldsymbol{z}_{\infty}^{(i)} r_z^{(i)T} \Sigma^{(i-1)}\right] + \sigma \times \hat{\boldsymbol{W}}^{(i)}\end{aligned} \tag{29}$$

Similarly, the system dynamics may be described as (17) we obtain

$$M\dot{r} = -V_m r - k_v r + \left[\boldsymbol{W}^{(d)} - \hat{\boldsymbol{W}}^{(d)}\right]^T \boldsymbol{z}_{\infty}^{(d)} + \tau_d + \epsilon. \tag{30}$$

Substituting the neural network into the system equation provides

$$M\dot{r} = -V_m r - k_v r + \tilde{\boldsymbol{W}}^{(d)T} \boldsymbol{z}_{\infty}^{(d)} + \epsilon + \tau_d. \tag{31}$$

Now, let the Lyapunov function be now given as

$$V = r^T M r^{(d)} + \sum_{i=1}^{d-1} r_{\boldsymbol{z}}^{(i)T} r_{\boldsymbol{z}}^{(i)} + tr(\tilde{\boldsymbol{W}}^{(d)T} \tilde{\boldsymbol{W}}^{(d)}) + \sum_{i=1}^{d-1} tr(\tilde{\boldsymbol{W}}^{(i)T} \tilde{\boldsymbol{W}}^{(i)}) \tag{32}$$

Next, we write the first derivative as

$$\dot{V} = \frac{1}{2} r^T [\dot{M} r + M\dot{r}] + \frac{1}{2} \sum_{i=1}^{d-1} r_{\boldsymbol{z}}^{(i)T} \dot{r}_{\boldsymbol{z}}^{(i)} + tr(\tilde{\boldsymbol{W}}^{(d)T} \dot{\tilde{\boldsymbol{W}}}^{(d)}) + \sum_{i=1}^{d-1} tr(\tilde{\boldsymbol{W}}^{(i)T} \dot{\tilde{\boldsymbol{W}}}^{(i)}) \tag{33}$$

Let us now substitute the dynamics and updates.

$$\dot{V} = \frac{1}{2} r^T [2\dot{M} r + M M^{-1} [-V_m r - k_v r \tag{34}$$

$$+\tilde{\boldsymbol{W}}^{(d)T} \boldsymbol{z}_{\infty}^{(d)} + \epsilon + \tau_d]] - \sum_{i=1}^{d-1} r_{\boldsymbol{z}}^{(i)T} K_z^{(i)} \times r_{\boldsymbol{z}}^{(i)} + \frac{1}{2} tr(\tilde{\boldsymbol{W}}^{(d)T} [-\boldsymbol{z}_{\infty}^{(d)} r^T + \sigma \times \hat{\boldsymbol{W}}^{(d)}]) \tag{35}$$

$$+\sum_{i=1}^{d-1} tr(\tilde{\boldsymbol{W}}^{(i)T} \left[\boldsymbol{z}_{\infty}^{(i-1)} r_{\boldsymbol{z}}^{(i)T} \Sigma^{(i-1)} + \sigma \times \hat{\boldsymbol{W}}^{(i)}\right]) \tag{36}$$

Simplification provides

$$\dot{V} = -r^T k_v r - \sum_{i=1}^{d-1} r_{\boldsymbol{z}}^{(i)T} K_z^{(i)} \times r_{\boldsymbol{z}}^{(i)}$$

$$-r^T[\dot{M} - 2V_m]r + r^T[\tilde{\boldsymbol{W}}^{(d)T} \boldsymbol{z}_\infty^{(d)} + \tau_d + \epsilon] - tr(r^T \tilde{\boldsymbol{W}}^{(d)T} \boldsymbol{z}_\infty^{(d)})$$

$$\sum_{i=1}^{d-1} tr(\tilde{\boldsymbol{W}}^{(i)T} \sigma \times \hat{\boldsymbol{W}}^{(i)}]) + \frac{1}{2} tr(\tilde{\boldsymbol{W}}^{(d)T} \sigma \times \hat{\boldsymbol{W}}^{(d)}]) \tag{37}$$

$$+\sum_{i=1}^{d-1} tr(\tilde{\boldsymbol{W}}^{(i)T} \big[\boldsymbol{z}_\infty^{(i-1)} r_{\boldsymbol{z}}^{(i)T} \Sigma^{(i-1)}]) \tag{38}$$

Considering $\tilde{\boldsymbol{W}}^{(i)} = \boldsymbol{W}^{(i)} - \hat{\boldsymbol{W}}^{(i)}$ and the skew symmetric property then gives $[\dot{M} - 2V_m] = 0$, the following ensues

$$\dot{V} = -r^T k_v r - \sum_{i=1}^{d-1} r_{\boldsymbol{z}}^{(i)T} K_z^{(i)} r_{\boldsymbol{z}}^{(i)} \tag{39}$$

$$+r^T[\tilde{\boldsymbol{W}}^{(d)T} \boldsymbol{z}_\infty^{(d)} + \tau_d + \epsilon - \tilde{\boldsymbol{W}}^{(d)T} z_\infty^{(d)}] \tag{40}$$

$$\sum_{i=1}^{d-1} tr(\tilde{\boldsymbol{W}}^{(i)T} \sigma \times [\boldsymbol{W}^{(i)} - \tilde{\boldsymbol{W}}^{(i)}]) \tag{41}$$

$$+\frac{1}{2} tr(\tilde{\boldsymbol{W}}^{(d)T} \sigma \times \hat{\boldsymbol{W}}^{(d)} [\boldsymbol{W}^{(i)} - \tilde{\boldsymbol{W}}^{(i)}]) \tag{42}$$

$$+\sum_{i=1}^{d-1} tr(\tilde{\boldsymbol{W}}^{(i)T} \big[\boldsymbol{z}_\infty^{(i-1)} r_{\boldsymbol{z}}^{(i)T} \Sigma^{(i-1)}]) \tag{43}$$

which provides

$$\dot{V} = -r^T k_v r - \sum_{i=1}^{d-1} \frac{1}{2} \Big(\sigma tr(\tilde{\boldsymbol{W}}^{(i)T} \tilde{\boldsymbol{W}}^{(i)})\Big) \tag{44}$$

$$-\frac{1}{2} \sigma tr(\tilde{\boldsymbol{W}}^{(d)T} \tilde{\boldsymbol{W}}^{(d)}]) - \sum_{i=1}^{d-1} r_{\boldsymbol{z}}^{(i)T} K_z^{(i)} r_{\boldsymbol{z}}^{(i)} \tag{45}$$

$$+r^T[\tilde{\boldsymbol{W}}^{(i)T} \boldsymbol{z}_\infty^{(d)} + \tau_d + \epsilon - \tilde{\boldsymbol{W}}^{(d)T} z_\infty^{(d)}] \tag{46}$$

$$\sum_{i=1}^{d-1} \Big(\sigma \times tr(\tilde{\boldsymbol{W}}^{(i)T} \boldsymbol{W}^{(i)})\Big) \tag{47}$$

$$+\frac{1}{2} \sigma tr(\tilde{\boldsymbol{W}}^{(d)T} \boldsymbol{W}^{(d)})) \tag{48}$$

$$+\sum_{i=1}^{d-1} tr(\tilde{\boldsymbol{W}}^{(i)T} \big[\boldsymbol{z}_\infty^{(i-1)} r_{\boldsymbol{z}}^{(i)T} \Sigma^{(i-1)}]) \tag{49}$$

Applying Young's inequality

$$\dot{V} \leq -r^T k_v r - \sum_{i=1}^{d-1} \frac{1}{2}\Big(\sigma tr(\tilde{\boldsymbol{W}}^{(i)T}\tilde{\boldsymbol{W}}^{(i)})\Big) \tag{50}$$

$$-\frac{1}{2}\sigma tr(\tilde{\boldsymbol{W}}^{(d)T}\tilde{\boldsymbol{W}}^{(d)}]) - \sum_{i=1}^{d-1} r_{\boldsymbol{z}}^{(i)T} K_z^{(i)} r_{\boldsymbol{z}}^{(i)} \tag{51}$$

$$+\frac{\|r\|^2}{2} + \frac{\|\tau_d\|^2}{2} + \frac{\|\epsilon\|^2}{2} + \tag{52}$$

$$\sum_{i=1}^{d-1} \frac{1}{2}\sigma\Big(\frac{\|\tilde{\boldsymbol{W}}^{(i)}\|^2}{2} + \frac{\|\boldsymbol{W}^{(i)}\|^2}{2}\Big) \tag{53}$$

$$+\frac{1}{2}\sigma\Big(\frac{\|\tilde{\boldsymbol{W}}^{(d)}\|^2}{2} + \frac{\|\boldsymbol{W}^{(d)}\|^2}{2}\Big) \tag{54}$$

$$+\sum_{i=1}^{d-1}\Big[\frac{\sigma||\tilde{\boldsymbol{W}}^{(i)}\|^2}{8} + \frac{(8-\sigma)\|\Sigma^{(i-1)}\boldsymbol{z}_{\infty}^{(i-1)} r_{\boldsymbol{z}}^{(i)T}\|^2}{8}\Big] \tag{55}$$

Rearranging provides

$$\dot{V} \leq -(k_v - 1/2)\|r\|^2 + \frac{\|\tau_d\|^2}{2} + \frac{\|\epsilon\|^2}{2} - \sum_{i=1}^{d-1}\Big(\frac{\sigma}{8}\Big)||\tilde{\boldsymbol{W}}^{(i)}\|^2 \tag{56}$$

$$-\frac{\sigma}{2}\|\tilde{\boldsymbol{W}}^{(d)}||^2 - \sum_{i=1}^{d-1}[K_z^{(i)} - (8-\sigma)\|\Sigma^{(i-1)}\boldsymbol{z}_{\infty}^{(i-1)}\|^2] \times \|r_{\boldsymbol{z}}^{(i)}\|^2 \tag{57}$$

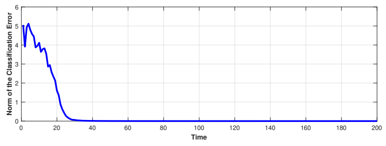
$$+\sum_{i=1}^{d}\frac{1}{2}\sigma\Big(\frac{\|\boldsymbol{W}^{(i)}\|^2}{2}\Big) \tag{58}$$

5 Results and Discussion

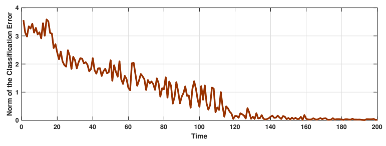
In this paper, the proposed framework is applied to the problem of classification. All simulations are performed in MATLAB. For the purpose of simulation, ode45 solvers are used to achieve the outputs at each layer in the NN. Refer to the appendix for implementation details and details on the XOR data-set. The hyper parameter details and the details of the data-set are provided in the appendix too.

5.1 XOR Problem

XOR is a classification problem where the expected outputs are known in advance and is a classic problem in the supervised learning setup. On the surface, XOR appears to be a very straight forward problem, however, the authors in (Yanling, Bimin, and Zhanrong 2002) reported that the problem involves learning a nonlinear function. With this objective the use of a NN is justified.



(a) Norm of the error for XOR classification



(b) Norm of the Error for MNIST Classification

Figure 1

Table 1: % Accuracies for the various data-sets with the proposed framework.

Datasets	Update rule(Theorem 3)	Update rule(Theorem 4)
Rolling	99	99
Sensorless	94	94
MNIST	92	94
Dexter	71	77

The results are in this case demonstrated with the back-propagation update rules. The resulting plots of the cost function are described in Fig. 1a. It is observed that the cost function converges and the end result is satisfactory with the accuracy given at 100 % for the XOR problem.

Next, we will provide results on the MNIST data-set.

5.2 MNIST Classification

Similar to the XOR classification problem, we next consider the problem of classification with MNIST data-set. Mini-batches sized for this problem are kept at 64. The learning rates are chosen as 0.001 while 80% of the data is chosen for training. Furthermore, twenty percent of the data is chosen for use during the testing phase. These splits are constructed randomly. All of the results presented in the section are on the test set. The accuracies are provided in Table. 1.

Satisfactory classification results are observed in the context of this data-sets and the resulting cost converges to zero. In the next section, several well known data-sets are utilized with the two update rules derived in this paper.

5.3 General Classification

A total of four data-sets are used for analysis in this study. The statistics for these data-sets are summarized in Table. 2 (Refer to appendix). Observe that the rolling element bearing, sensorless, and dexter are fault diagnostics data-sets. For this analysis, we use a five layer NN with activation functions chosen as tanh. Moreover, weights are all initialized at random. The accuracies for these data-sets are standard and match the corresponding literature.

6 Conclusions

In this paper, the learning behavior of NN was modeled by a system of differential equations and it was shown that the process provided important and practical insights into NN learning. The efficiency of popular heuristics back-propagation, weight perturbation, hidden layer perturbations was observed through precise analysis. It was shown that the practitioner is not limited to using back-propagation based learning but can use control theoretic tools to derive different update rules. Moreover, the control theoretic approach can help to derive the learning rules for any number of layers of the NN. The main limitation is because, the approach does not take into account the stochastic nature of the data, which could be part of the future work.

7 Acknowledgements

This research was supported in part by an NSF I/UCRC award IIP 1134721 and Intelligent Systems Center.

References

Werbos, Paul J (1990). “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10, pp. 1550–1560.

Rumelhart, David E, Geoffrey E Hinton, Ronald J Williams, et al. (1988). “Learning representations by back-propagating errors”. In: *Cognitive modeling* 5.3, p. 1.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). “On the difficulty of training recurrent neural networks”. In: *International conference on machine learning*, pp. 1310–1318.

Dauphin, Yann N et al. (2014). “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In: *Advances in neural information processing systems*, pp. 2933–2941.

Jaderberg, Max et al. (2016). “Decoupled neural interfaces using synthetic gradients”. In: *arXiv preprint arXiv:1608.05343*.

Schmidhuber, Jürgen (2015). “Deep learning in neural networks: An overview”. In: *Neural networks* 61, pp. 85–117.

Tian, Yuchi et al. (2018). “Deeptest: Automated testing of deep-neural-network-driven autonomous cars”. In: *Proceedings of the 40th international conference on software engineering*. ACM, pp. 303–314.

Sysoev, Oleg and Oleg Burdakov (2019). “A smoothed monotonic regression via l2 regularization”. In: *Knowledge and Information Systems* 59.1, pp. 197–218.

Bishop, Chris M (1995). “Training with noise is equivalent to Tikhonov regularization”. In: *Neural computation* 7.1, pp. 108–116.

Ororbia II, Alexander G, Daniel Kifer, and C Lee Giles (2017). “Unifying adversarial training algorithms with data gradient regularization”. In: *Neural computation* 29.4, pp. 867–887.

Ioffe, Sergey and Christian Szegedy (2015). “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167*.

Zhang, Chiyuan et al. (2016). “Understanding deep learning requires rethinking generalization”. In: *arXiv preprint arXiv:1611.03530*.

Wen, Yeming et al. (2018). “Flipout: Efficient pseudo-independent weight perturbations on mini-batches”. In: *arXiv preprint arXiv:1803.04386*.

Zeng, Xiaoqin and Daniel S Yeung (2001). “Sensitivity analysis of multilayer perceptron to input and weight perturbations”. In: *IEEE Transactions on Neural Networks* 12.6, pp. 1358–1366.

Lee, Dong-Hyun et al. (2015). “Difference target propagation”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 498–515.

Nøkland, Arild (2016). “Direct feedback alignment provides learning in deep neural networks”. In: *Advances in Neural Information Processing Systems*, pp. 1037–1045.

Taylor, Gavin et al. (2016). “Training neural networks without gradients: A scalable admm approach”. In: *International Conference on Machine Learning*, pp. 2722–2731.

Carreira-Perpinan, Miguel and Weiran Wang (2014). “Distributed optimization of deeply nested systems”. In: *Artificial Intelligence and Statistics*, pp. 10–19.

Mobahi, Hossein and John W Fisher III (2015). “A Theoretical Analysis of Optimization by Gaussian Continuation.” In: *AAAI*, pp. 1205–1211.

Vese, Luminita (1999). “A method to convexify functions via curve evolution”. In: *Communications in Partial Differential Equations* 24.9-10, pp. 1573–1591. DOI: `10.1080/03605309908821476`. eprint: `https://doi.org/10.1080/03605309908821476`. URL: `https://doi.org/10.1080/03605309908821476`.

Haber, Eldad and Lars Ruthotto (2017). “Stable architectures for deep neural networks”. In: *Inverse Problems* 34.1, p. 014004.

Chen, Tian Qi et al. (2018). “Neural ordinary differential equations”. In: *Advances in Neural Information Processing Systems*, pp. 6571–6583.

Grossberg, Stephen (1982). “How does a brain build a cognitive code?” In: *Studies of mind and brain*. Springer, pp. 1–52.

Chang, Bo et al. (2019). “AntisymmetricRNN: A Dynamical System View on Recurrent Neural Networks”. In: *arXiv preprint arXiv:1902.09689*.

Salman, Hadi et al. (2018). “Deep Diffeomorphic Normalizing Flows”. In: *arXiv preprint arXiv:1810.03256*.

He, Kaiming et al. (2016). “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Bishop, Christopher M (2006). *Pattern recognition and machine learning*. springer.

Lewis, FW, Suresh Jagannathan, and A Yesildirak (1998). *Neural network control of robot manipulators and non-linear systems*. CRC Press.

Jagannathan, Sarangapani and Frank L Lewis (1996). “Multilayer discrete-time neural-net controller with guaranteed performance”. In: *IEEE Transactions on Neural Networks* 7.1, pp. 107–130.

Moore, J (1983). “Persistence of excitation in extended least squares”. In: *IEEE Transactions on Automatic Control* 28.1, pp. 60–68.

Daneshmand, Hadi et al. (2018). “Escaping saddles with stochastic gradients”. In: *arXiv preprint arXiv:1803.05999*.

Yanling, Zhao, Deng Bimin, and Wang Zhanrong (2002). “Analysis and study of perceptron to solve XOR problem”. In: *The 2nd International Workshop on Autonomous Decentralized System, 2002*. IEEE, pp. 168–173.

Soylemezoglu, Ahmet, S Jagannathan, and Can Saygin (2010). “Mahalanobis Taguchi system (MTS) as a prognostics tool for rolling element bearing failures”. In: *Journal of Manufacturing Science and Engineering* 132.5, p. 051014.

Lichman, M. (2013). *UCI Machine Learning Repository*. URL: `http://archive.ics.uci.edu/ml`.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep learning”. In: *nature* 521.7553, p. 436.

Guyon, Isabelle et al. (2005). “Result analysis of the NIPS 2003 feature selection challenge”. In: *Advances in neural information processing systems*, pp. 545–552.

Murray, Richard M (2017). *A mathematical introduction to robotic manipulation*. CRC press.

8 Appendix

8.1 Implementation Details and the XOR problem

8.1.1 Data-set

The details of all the data-set utilized for analysis is this work is provided in Table. 2

Table 2: Summary descriptions of the different data-sets used in this paper

Data-set	Dimensions	Data points	Classes
Rolling (Soylemezoglu, Jagannathan, and Saygin 2010)	11	35000	4
Sensorless (Lichman 2013)	48	78000	11
MNIST (LeCun, Bengio, and Hinton 2015)	784	72000	10
Dexter (Guyon et al. 2005)	20000	300	2

8.1.2 Implementation Details

One of the primary issues with the ODE based approaches introduced in (Haber and Ruthotto 2017; Chen et al. 2018) is due to the scenario that batch-wise updates are not possible. However, the update rules presented in this paper, guaranteed to be efficient when batch wise learning is considered. An algorithm for such an extension is provided in Algorithm 1.

Specifically, for each batch of data, all the ODE equations are allowed to converge. Then the process is repeated for other batches in the data.

As described in (Haber and Ruthotto 2017; Chen et al. 2018)) unique solution of the differential equation can be observed when the neural network has finite weights and uses Lipshitz nonlinearities, such as tanh or relu. The finiteness of weights are guaranteed by Theorem 2 and 3.

For the results mentioned below, we choose tanh nonlinearities. First we would demonstrate efficiency on the XOR problem, where the whole data-set is considered at once. Next, we would show results on the MNIST data-set. Finally, we will provide some general classification results with common benchmarking data-sets. For all the test, the experiments are executed 100 times and the mean is reported. For batch-wise analysis, the batch size is considered to be 64.

Before getting into the details of the neural network implementation, we will start with a description of the XOR problem.

8.1.3 XOR classification

The XOR, or "exclusive or" is the problem of using a neural network to predict the outputs of XOR logic gates given two binary inputs. An XOR function should return a true value if the two inputs are not equal and a false value if they are equal. The truth table for an XOR function can be given as follows in Table 3 We consider a three layer neural network for this analysis. The size of the input $\boldsymbol{z}^{(1)}$ and its hidden representation $\boldsymbol{z}^{(2)}$ can be achieved as $\boldsymbol{z}^{(1)}, \in \mathbb{R}^{2\times 4}$ and $\boldsymbol{z}^{(2)}, \dot{\boldsymbol{z}}^{(2)} \in \mathbb{R}^{5\times 4}$ respectively.

Table 3: Truth table for XOR gates

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

With this setup, the sizes of respective weights are given as $\hat{\boldsymbol{W}}^{(1)}, \dot{\hat{\boldsymbol{W}}}^{(1)} \in \mathbb{R}^{\eta \times 2}$ and $\hat{\boldsymbol{W}}^{(2)}, \dot{\hat{\boldsymbol{W}}}^{(2)} \in \mathbb{R}^{1 \times \eta}$. Finally, we get $\boldsymbol{r}, \dot{\boldsymbol{r}} \in \mathbb{R}^{1 \times 4}$ with the cost being scalar.

8.2 Lyapunov Principles

Specifically, in this analysis we seek an aggregate summarizing function that explains the behavior of the system at any time instant t. Through this function, one may make conclusions about the first derivative using the following definition.

Definition 1 (Definition of stability in the lyapunov sense (Murray 2017))**.** *Let $V(x,t)$ be a non-negative function with derivative $\dot{V}(x,t)$ along the system. The following is then true*

1. *If $V(x,t)$ is locally positive definite and $\dot{V}(x,t) \leq 0$ locally in x and for all t, then the equilibrium point is locally stable (in the sense of Lyapunov).*
2. *If $V(x,t)$ is locally positive definite and decresent, and $\dot{V}(x,t) \leq 0$ locally in x and for all t, then the equilibrium point is uniformly locally stable (in the sense of Lyapunov).*
3. *If $V(x,t)$ is locally positive definite and decresent, and $\dot{V}(x,t) < 0$ locally in x and for all t, then the equilibrium point is locally asymptotically stable.*
4. *If $V(x,t)$ is locally positive definite and decresent, and $\dot{V}(x,t) < 0$ in x and for all t, then the equilibrium point is globally asymptotically stable.*

Equillibrium points in the context of classification may mean the minima of the cost function. In the context of the problem of classification, local stability refers to the case when the system approaches close to the minima and the classification error is bounded. On the other hand, asymptotic stability refers to the case when the classification error approaches zero.

8.3 Derivation of the system of equation

To derive the new framework, we will first differentiate the cost function with respect to time and derive a time evolution equation for every layer in the network. It follows that

$$\nabla_t J(\hat{\boldsymbol{y}}(\boldsymbol{z}^{(d)}); \hat{\boldsymbol{\theta}}) = \boldsymbol{r}^T \nabla_t \boldsymbol{r} \tag{59}$$

where ∇_t refers to the derivative with respect to time t. Since $\boldsymbol{r} = \boldsymbol{y}_t - \hat{y}_t$, and $\boldsymbol{y}$ for each batch is constant, therefore $\dot{\boldsymbol{y}}_t = 0$. for $\forall t$. The term $\nabla_t \boldsymbol{r}$ through chain rule can be written as

$$\begin{aligned}
&\nabla_t \boldsymbol{r} = -\nabla_{(\hat{\boldsymbol{W}}^{(d)T}\boldsymbol{z}^{(d)})}\phi^d(\hat{\boldsymbol{W}}^{(d)T}\boldsymbol{z}^{(d)}) \\
&[\nabla_t \hat{\boldsymbol{W}}^{(d)T}\boldsymbol{z}^{(d)} + \hat{\boldsymbol{W}}^{(d)T}\nabla_t \boldsymbol{z}^{(d)}], \qquad (60)
\end{aligned}$$

where $\nabla_{\hat{\boldsymbol{W}}^{(d)T}\boldsymbol{z}^{(d)}}$ denotes derivative with respect to $\hat{\boldsymbol{W}}^{(d)T}\boldsymbol{z}^{(d)}$. Similarly, one may write the dynamics for $\boldsymbol{z}^{(d)}$ through chain rule as

$$\begin{aligned}
&\nabla_t \boldsymbol{z}^{(d)} = \nabla_{(\hat{\boldsymbol{W}}^{(d-1)T}\boldsymbol{z}^{(d-1)})}\phi^{d-1}(\hat{\boldsymbol{W}}^{(d-1)T}\boldsymbol{z}^{(d-1)}) \\
&[\nabla_t \hat{\boldsymbol{W}}^{(d-1)T}\boldsymbol{z}^{(d-1)} + \hat{\boldsymbol{W}}^{(d-1)T}\nabla_t \boldsymbol{z}^{(d-1)}] \qquad (61)
\end{aligned}$$

Next, $\nabla_t \boldsymbol{z}^{(d-1)}$ can be simplified as

$$\begin{aligned}
&\nabla_t \boldsymbol{z}^{(d-1)} = \nabla_{(\hat{\boldsymbol{W}}^{(d-2)T}\boldsymbol{z}^{(d-2)})}\phi^{d-1}(\hat{\boldsymbol{W}}^{(d-2)T}\boldsymbol{z}^{(d-2)}) \\
&[\nabla_t \hat{\boldsymbol{W}}^{(d-2)T}\boldsymbol{z}^{(d-2)} + \hat{\boldsymbol{W}}^{(d-2)T}\nabla_t \boldsymbol{z}^{(d-2)}]. \qquad (62)
\end{aligned}$$

We can generalize these expressions to get a system of equations as

$$\begin{aligned}
&\nabla_t J(\hat{\boldsymbol{y}}(\boldsymbol{z}^{(d)}); \hat{\boldsymbol{\theta}}) = \boldsymbol{r}^T \nabla_t \boldsymbol{r} \\
&\nabla_t \boldsymbol{r} = -\nabla_{\hat{\boldsymbol{W}}^{(d)T}\boldsymbol{z}^{(d)}}\big(\phi^d(\hat{\boldsymbol{W}}^{(d)T}\boldsymbol{z}^{(d)})\big)[\nabla_t \hat{\boldsymbol{W}}^{(d)T}\boldsymbol{z}^{(d)} + \hat{\boldsymbol{W}}^{(d)T}\nabla_t \boldsymbol{z}^{(d)}] \\
&\nabla_t \boldsymbol{z}^{(d)} = \nabla_{\hat{\boldsymbol{W}}^{(d-1)T}\boldsymbol{z}^{(d-1)}}\big(\phi^{d-1}(\hat{\boldsymbol{W}}^{(d-1)T}\boldsymbol{z}^{(d-1)})\big)[\nabla_t \hat{\boldsymbol{W}}^{(d-1)T}\boldsymbol{z}^{(d-1)} + \hat{\boldsymbol{W}}^{(d-1)T}\nabla_t \boldsymbol{z}^{(d-1)}] \\
&\vdots \\
&\nabla_t \boldsymbol{z}^{(2)} = \nabla_{\hat{\boldsymbol{W}}^{(1)T}\boldsymbol{z}^{(1)}}\big(\phi^1(\hat{\boldsymbol{W}}^{(1)T}\boldsymbol{z}^{(1)})\big)[\nabla_t \hat{\boldsymbol{W}}^{(1)T}\boldsymbol{z}^{(1)} + \hat{\boldsymbol{W}}^{(1)T}\nabla_t \boldsymbol{z}^{(1)}], \qquad (63)
\end{aligned}$$

where $\boldsymbol{z}^{(1)} = \boldsymbol{z}_t$. To modify these equations in traditional differential equation notation, one may replace $\nabla_{\hat{\boldsymbol{W}}^{(1)}\boldsymbol{z}^{(1)}}(.)$ with $\delta(.)$ and $\nabla_t(.) = \dot{(.)}$ and rewrite the above equation as

$$\begin{aligned}
&\dot{J}(\hat{\boldsymbol{y}}(\boldsymbol{z}^{(d)}); \hat{\boldsymbol{\theta}}) = \boldsymbol{r}^T \dot{\boldsymbol{r}} \\
&\dot{\boldsymbol{r}} = -\delta\phi^{(d)}(\hat{\boldsymbol{W}}^{(d)T}\boldsymbol{z}^{(d)})[\dot{\hat{\boldsymbol{W}}}^{(d)T}\boldsymbol{z}^{(d)} + \hat{\boldsymbol{W}}^{(d)T}\dot{\boldsymbol{z}}^{(d)}] \\
&\dot{\boldsymbol{z}}^{(d)} = \big(\delta\phi^{(d-1)}(\hat{\boldsymbol{W}}^{(d-1)T}\boldsymbol{z}^{(d-1)})\big)[\dot{\hat{\boldsymbol{W}}}^{(d-1)T}\boldsymbol{z}^{(d-1)} + \hat{\boldsymbol{W}}^{(d-1)T}\dot{\boldsymbol{z}}^{(d-1)}] \\
&\vdots \\
&\dot{\boldsymbol{z}}^{(2)} = \big(\delta\phi^{(1)}(\hat{\boldsymbol{W}}^{(1)T}\boldsymbol{z}^{(1)})\big)[\dot{\hat{\boldsymbol{W}}}^{(1)T}\boldsymbol{z}^{(1)} + \hat{\boldsymbol{W}}^{(1)T}\dot{\boldsymbol{z}}^{(1)}]. \qquad (64)
\end{aligned}$$

We will rewrite the above equation with some notational simplification to achieve the final system as

$$\begin{aligned}
\dot{\boldsymbol{r}} &= -\delta\phi^{(d)}[\dot{\hat{\boldsymbol{W}}}^{(d)T}\boldsymbol{z}^{(d)} + \hat{\boldsymbol{W}}^{(d)T}\dot{\boldsymbol{z}}^{(d)}] \\
\dot{\boldsymbol{z}}^{(d)} &= \delta\phi^{(d-1)}[\dot{\hat{\boldsymbol{W}}}^{(d-1)T}\boldsymbol{z}^{(d-1)} + \hat{\boldsymbol{W}}^{(d-1)T}\dot{\boldsymbol{z}}^{(d-1)}] \\
&\vdots \\
\dot{\boldsymbol{z}}^{(2)} &= \delta\phi^{(1)}[\dot{\hat{\boldsymbol{W}}}^{(1)T}\boldsymbol{z}^{(1)} + \hat{\boldsymbol{W}}^{(1)T}\dot{\boldsymbol{z}}^{(1)}].
\end{aligned}$$

Note that $\delta\phi^{(i)}, \forall i = 1, \cdots d$ are diagonal matrices of appropriate dimensions, $\boldsymbol{r}$ denotes classification error while $\dot{\boldsymbol{z}}^{(i)}$ and $\boldsymbol{z}^{(i)}$ are the hidden layer states and their respective derivatives.

8.4 Theorem Proofs

Proof of Theorem 2. Choose the Lyapunov candidiate function as

$$V = \frac{1}{n}\sum_{j=1}^{n} V_k, \tag{65}$$

How do we verify, this is a proper Lyapunov function? where n refers to the number of data-points in the data. Let V_j be written as

$$V_k = \frac{1}{2}\boldsymbol{r}^T\boldsymbol{r} + tr(\tilde{W}^T\tilde{W}). \tag{66}$$

Each term in V can be understood to denote the contribution of each data-point to the energy of the system, which, in the case of neural networks refers to the cost incurred by it. Since $\boldsymbol{r} = \boldsymbol{y}_t - \hat{y}_t$, and $\boldsymbol{y}$ for each data-point is constant, $\dot{\boldsymbol{y}}_t = 0$. for $\forall t$. With this fact, we take the first derivative of V_k with respect to time and write

$$\dot{V}_k = \boldsymbol{r}^T\dot{\boldsymbol{r}} \tag{67}$$

Substituting the dynamics of $\dot{\boldsymbol{r}}$ to achieve

$$\dot{V}_k = -\boldsymbol{r}^T(\delta\phi^{(d)}[\dot{\hat{\boldsymbol{W}}}^{(d)T}\boldsymbol{z}^{(d)} + \hat{\boldsymbol{W}}^{(d)T}\dot{\boldsymbol{z}}^{(d)}]). \tag{68}$$

Substituting $\dot{\boldsymbol{z}}^{(d)}, \dot{\boldsymbol{z}}^{(d-1)}, \cdots$ from (5) results in

$$\begin{aligned}
\dot{V}_k = -\boldsymbol{r}^T\Bigg(\delta\phi^{(d)}\Bigg[\dot{\hat{\boldsymbol{W}}}^{(d)T}\boldsymbol{z}^{(d)} + \hat{\boldsymbol{W}}^{(d)T}\delta\phi^{(d-1)}\big[\dot{\hat{\boldsymbol{W}}}^{(d-1)T}\boldsymbol{z}^{(d-1)} \\
+ \hat{\boldsymbol{W}}^{(d-1)T}\big(\delta\phi^{(d-2)}[\dot{\hat{\boldsymbol{W}}}^{(d-2)T}\boldsymbol{z}^{(d-2)} + \hat{\boldsymbol{W}}^{(d-1)T}(\cdots)]\big)\big]\Bigg]\Bigg).
\end{aligned} \tag{69}$$

Simplification reveals

$$\dot{V}_k = -\boldsymbol{r}^T \bigg[\delta\phi^{(d)} \dot{\hat{\boldsymbol{W}}}^{(d)T} \boldsymbol{z}^{(d)} + \delta\phi^{(d)} \hat{\boldsymbol{W}}^{(d)T} \delta\phi^{(d-1)} \dot{\hat{\boldsymbol{W}}}^{(d-1)T} \boldsymbol{z}^{(d-1)}]\} \\ + \delta\phi^{(d)} \hat{\boldsymbol{W}}^{(d)T} \delta\phi^{(d-1)} \hat{\boldsymbol{W}}^{(d-1)T} \delta\phi^{(d-2)} \dot{\hat{\boldsymbol{W}}}^{(d-2)T} \boldsymbol{z}^{(d-2)} + \cdots \bigg] \tag{70}$$

In Eq. (70), let $A_i = \delta\phi^{(d)}[\prod_{j=d-1}^{j+1} \hat{\boldsymbol{W}}^{(j+1)T} \delta\phi^{(j)}] \dot{\hat{\boldsymbol{W}}}^{(i)T} \boldsymbol{z}^{(i)}$. and rewrite Eq. (70) as

$$\dot{V}_k = -\boldsymbol{r}^T[\delta\phi^{(d)} \dot{\hat{\boldsymbol{W}}}^{(d)T} \boldsymbol{z}^{(d)}\} + \sum_{i=1}^{d-1} \boldsymbol{r}^T[A_i]), \tag{71}$$

Next, consider the second term in Eq. (71). With $\dot{\hat{\boldsymbol{W}}}^{(i)} = \alpha^i \boldsymbol{z}^{(i)} \boldsymbol{r}^T \delta\phi^{(d)} [\prod_{j=d-1}^{i+1} \hat{\boldsymbol{W}}^{(j+1)T} \delta\phi^{(j)}]$and $\alpha^i, \forall i = 1,2,3 > 0$, A_i can be rewritten as

$$A_i = \alpha^i \delta\phi^{(d)} [\prod_{j=d-1}^{i+1} \hat{\boldsymbol{W}}^{(j+1)T} \delta\phi^{(j)} \prod_{j=i+1}^{d-1} \delta\phi^{(j)T} \hat{\boldsymbol{W}}^{(j+1)}] \delta\phi^{(d)T} \boldsymbol{r}(\boldsymbol{z}^{(i)^T} \boldsymbol{z}^{(i)}). \tag{72}$$

Let $\Gamma^{(i)} = [\prod_{j=i+1}^{d-1} \delta\phi^{(j)T} \hat{\boldsymbol{W}}^{(j+1)}]$ and simplify to achieve

$$A_i = \alpha^i \Gamma^{(i)T} \Gamma^{(i)} \boldsymbol{r}(\boldsymbol{z}^{(i)^T} \boldsymbol{z}^{(i)}), \tag{73}$$

Similarly, consider the first term in Eq. (71) and choose $\dot{\hat{\boldsymbol{W}}}^{(d)}$ as $\alpha^d \boldsymbol{z}^{(d)} \boldsymbol{r} \delta\phi^{(d)}$ and simplify to get the first term as

$$-\alpha^d \left[\boldsymbol{r}^T \Gamma^{(d)T} \Gamma^{(d)} \boldsymbol{r}(\boldsymbol{z}^{(d)T} \boldsymbol{z}^{(d)}) \right] \tag{74}$$

Upon substitution into (71) with A_i as given in (73), one may achieve

$$\dot{V}_k = -\alpha^d \left[\boldsymbol{r}^T \Gamma^{(d)T} \Gamma^{(d)} \boldsymbol{r}(\boldsymbol{z}^{(d)T} \boldsymbol{z}^{(d)}) \right] - \sum_{i=1}^{d} \left[\alpha^i \boldsymbol{r}^T \Gamma^{(i)T} \Gamma^{(i)} \boldsymbol{r}(\boldsymbol{z}^{(i)^T} \boldsymbol{z}^{(i)}) \right], \tag{75}$$

Notice that both terms in Eq. (75) are in quadratic forms, therefore $\dot{V}_k \geq 0$ (wrong!!!!). Additionally, with the condition that $(\boldsymbol{z}^{(i)^T} \boldsymbol{z}^{(i)}) > 0$ as well as $\Gamma^{(i)} \Gamma^{(i)T} > 0, \forall i$, $\dot{V}_j$ is strictly greater than zero.

Since $\dot{V} = \sum \dot{V}_j$, $\dot{V}_k > 0$ implies $\dot{V} > 0$ (wrong!!). The conclusion implies $r \to \infty$ as $t \to \infty$, thus completing the proof. □

Proof of Theorem 3. Choose the Lyapunov candidiate function as

$$V = E[V_j], \tag{76}$$

where E denotes the expected value where the expectation is taken over different batches in the data-set. Consider iid sampling on the data and assume uniform distribution. We may rewrite (76) as

$$V = \sum_{j=1}^{n} V_j, \tag{77}$$

where n refers to the number of batches in the data. Specifically,(77) can be understood as choosing a Lyapunov function for each batch in the data.

$$\dot{V}_j = \frac{1}{2} tr\{\boldsymbol{r}^T \boldsymbol{r}\} \tag{78}$$

tr is the trace operator. Each term in V denotes the value of Lyapunov function with respect to one batch in the data.Since $\boldsymbol{r} = \boldsymbol{y}_t - \hat{y}_t$, and $\boldsymbol{y}$ for each batch is constant, therefore $\dot{\boldsymbol{y}}_t = 0$. for $\forall t$.Taking the first derivative with respect to time provides

$$\dot{V}_j = \frac{1}{2} tr\{\boldsymbol{r}^T \dot{\boldsymbol{r}}\} \tag{79}$$

Substituting the dynamics to achieve

$$\dot{V}_j = -tr\{\boldsymbol{r}^T (\delta\phi^{(d)} [\dot{\hat{\boldsymbol{W}}}^{(d)} \boldsymbol{z}^{(d)} + \hat{\boldsymbol{W}}^{(d)} \dot{\boldsymbol{z}}^{(d)}])\}. \tag{80}$$

Substituting $\dot{\boldsymbol{z}}^{(d)}, \dot{\boldsymbol{z}}^{(d-1)}, \cdots$ from (5) results in

$$\begin{aligned} \dot{V}_j = -tr\{\boldsymbol{r}^T \Bigg(\delta\phi^{(d)} \Bigg[\dot{\hat{\boldsymbol{W}}}^{(d)} \boldsymbol{z}^{(d)} + \hat{\boldsymbol{W}}^{(d)} \delta\phi^{(d-1)} \big[\dot{\hat{\boldsymbol{W}}}^{(d-1)} \boldsymbol{z}^{(d-1)} \\ + \hat{\boldsymbol{W}}^{(d-1)} \big(\delta\phi^{(d-2)} [\dot{\hat{\boldsymbol{W}}}^{(d-2)} \boldsymbol{z}^{(d-2)} + \hat{\boldsymbol{W}}^{(d-1)} (\cdots)]) \big] \Bigg] \Bigg) \}. \end{aligned} \tag{81}$$

Simplification reveals

$$\begin{aligned} \dot{V}_j = -tr\{\boldsymbol{r}^T [\delta\phi^{(d)} \dot{\hat{\boldsymbol{W}}}^{(d)} \boldsymbol{z}^{(d)} + \delta\phi^{(d)} \hat{\boldsymbol{W}}^{(d)} \delta\phi^{(d-1)} \dot{\hat{\boldsymbol{W}}}^{(d-1)} \boldsymbol{z}^{(d-1)}\} \\ + \delta\phi^{(d)} \hat{\boldsymbol{W}}^{(d)} \delta\phi^{(d-1)} \hat{\boldsymbol{W}}^{(d-1)} \delta\phi^{(d-2)} \dot{\hat{\boldsymbol{W}}}^{(d-2)} \boldsymbol{z}^{(d-2)} + \cdots]\} \end{aligned} \tag{82}$$

It follows on collecting terms that

$$\dot{V}_j = -tr\{\boldsymbol{r}^T [\delta\phi^{(d)} \dot{\hat{\boldsymbol{W}}}^{(d)} \boldsymbol{z}^{(d)}\} + \sum_{i=1}^{d-1} tr\{\boldsymbol{r}^T [A_i]\}, \tag{83}$$

where $A_i = [\prod_{j=i+1}^{d} \delta\phi^{(j)} \hat{\boldsymbol{W}}^{(j)}] \delta\phi^{(i)} \dot{\hat{\boldsymbol{W}}}^{(i)} \boldsymbol{z}^{(i)}$. With $\dot{\hat{\boldsymbol{W}}}^{(i)} = \alpha^i \delta\phi^{(i)-T} \prod_{j=d}^{i+1} \delta\phi^{(j)-T} \hat{\boldsymbol{W}}^{(j)T} \boldsymbol{r} \boldsymbol{z}^{(i)^T}$ where $\alpha^i, \forall i = 1, 2, 3 > 0$, A_i can be rewritten as

$$A_i = \alpha^i [\prod_{j=i+1}^{d} \delta\phi^{(j)} \hat{\boldsymbol{W}}^{(j)} (\delta\phi^{(i)} \delta\phi^{(i)-T}) \prod_{j=i}^{d} \hat{\boldsymbol{W}}^{(j)T} \delta\phi^{(j)-T}] \boldsymbol{r} (\boldsymbol{z}^{(i)^T} \boldsymbol{z}^{(i)}). \tag{84}$$

Let $\Gamma^{(i)} = \left[\prod_{j=i+1}^{d} \delta\phi^{(j)} \hat{\boldsymbol{W}}^{(j)} \prod_{j=i}^{d} \hat{\boldsymbol{W}}^{(j)T} \delta\phi^{(j)-T}\right]$ and simplify to achieve

$$A_i = \alpha^i \Gamma^{(i)} \boldsymbol{r}(\boldsymbol{z}^{(i)^T} \boldsymbol{z}^{(i)}). \tag{85}$$

Consider the learning signal for $\dot{\hat{\boldsymbol{W}}}^{(d)}$ as $\alpha^d \delta\phi^{(d)-T} \boldsymbol{r}\boldsymbol{z}^{(d)T}$, substitute into (83) with A_i as given in (85) to get

$$\dot{V}_j = -\alpha^d tr\{\boldsymbol{r}^T \Gamma^{(d)} \boldsymbol{r}(\boldsymbol{z}^{(d)T} \boldsymbol{z}^{(d)})\} - \sum_{i=1}^{d} \alpha^i tr\{\boldsymbol{r}^T \Gamma^{(i)} \boldsymbol{r}(\boldsymbol{z}^{(i)^T} \boldsymbol{z}^{(i)})\}, \tag{86}$$

Consider the condition that $\|\boldsymbol{z}^{(i)}\|^2 > 0, \forall i = 1, 2, 3,$ with $\|\hat{\boldsymbol{W}}^{(i)}\|^2 > 0, \forall i = 1, 2, 3.$ and $\|\delta \boldsymbol{f}^{(i)}\|^2 > 0$. With these conditions, it can be seen that $\Gamma^{(i)}$ is positive definite and $(\boldsymbol{z}^{(d)T} \boldsymbol{z}^{(d)})$ is positive definite. Thus V_j is negative definite. The final result from (77) provides $\dot{V}$ is negative definite for all t and therefore, $r \to 0$ with $t \to \infty$. $\square$