Online-Learning Deep Neuro-Adaptive Dynamic Inversion Controller for Model Free Control

1st Nathan Lutes
Mechanical and Aerospace Dept.
Missouri University of Science and Technology
Rolla, MO, USA
nalmrb@mst.edu

2nd K. Krishnamurthy
Mechanical and Aerospace Dept.
Missouri University of Science and Technology
Rolla, MO, USA
kkrishna@mst.edu

3rd Venkata Sriram Siddhardh Nadendla
Computer Science Dept.
Missouri University of Science and Technology
Rolla, MO, USA
nadendla@mst.edu

4th S. N. Balakrishnan
Mechanical and Aerospace Dept.
Missouri University of Science and Technology
Rolla, MO, USA

***Abstract*—Adaptive methods are popular within the control literature due to the flexibility and forgiveness they offer in the area of modelling. Neural network adaptive control is favorable specifically for the powerful nature of the machine learning algorithm to approximate unknown functions and for the ability to relax certain constraints within traditional adaptive control. Deep neural networks are large framework networks with vastly superior approximation characteristics than their shallow counterparts. However, implementing a deep neural network can be difficult due to size specific complications such as vanishing/exploding gradients in training. In this paper, a neuro-adaptive controller is implemented featuring a deep neural network trained on a new weight update law that escapes the vanishing/exploding gradient problem by only incorporating the sign of the gradient. The type of controller designed is an adaptive dynamic inversion controller utilizing a modified state observer in a secondary estimation loop to train the network. The deep neural network learns the entire plant model on-line, creating a controller that is completely model free. The controller design is tested in simulation on a 2 link planar robot arm. The controller is able to learn the nonlinear plant quickly and displays good performance in the tracking control problem.**

***Index Terms*—Nonlinear, adaptive, control, deep neural network, data-driven, model free**

I. Introduction

Adaptive control methods have been on the rise in recent years within the control theory literature. Their ability to correct errors in modelling or to capture and cancel unknown disturbances have proven to be quite desirable as more 'intelligent' and resilient control methods are sought. A quite popular and very powerful form of adaptive control has been found in controllers featuring artificial neural networks (ANNs). ANNs are machine learning algorithms with the extremely useful ability to model, or 'learn', most any function within some degree of error and have been used extensively for other tasks such as learning large and complex data sets in the realm of data science. Within the scope of control theory, ANNs are primarily utilized as estimation methods within controllers to learn uncertainty in plant dynamics or disturbances online and thus improve the controller performance over time. For example, [14] and [13] created adaptive dynamic inversion controllers by using a neural network state observer, termed modified state observer (MSO), in a separate estimation loop to learn modelling uncertainty or unknown disturbances in non-affine, non-square nonlinear systems. In [5], this type of control scheme was extended to an event triggered control framework. Other examples of neural network control are prevalent throughout the literature for many different applications from aerospace to autonomous vehicles to robotics. In [10], a missile guidance law was created by transforming a model predictive control scheme into a constrained quadratic programming problem and solved online using neural networks in a finite receding horizon. In [18], a tracking control solution for unmanned surface vehicle's was created based on a combination of Deep Deterministic Policy Gradient (DDPG) and neural network control. In [17], a neural network was employed to approximate plant dynamics and external disturbances for use in combining active disturbance attenuation (ADA) with robust dynamic programming (RADP). Recurrent neural networks were used in [12] in the form of long short term memory neural networks (LSTM-NN) for an attitude estimation application to unmanned aerial vehicles (UAV)'s by training the network on attitude time-series data.

Recent advances in machine learning have lead to the development of deep neural networks (DNNs) which are even more powerful, and duely more complex, versions of ANNs, usually with considerably larger structures. In [6], an in-depth look at DNNs is provided, discussing their development, popular models, applications and future research directions.

This paper was funded by the NASA Missouri Space Grant Consortium. We appreciate all of their generous aid.

This paper is dedicated to my late advisor, Dr. S. N. Balakrishnan, who unfortunately passed before its submission. His helpful aid was instrumental in the development of this document.

Within the realm of control theory, DNNs are commonly used for reinforcement learning. The deep reinforcement learning (DRL) framework is presented in [9] and then its applications and some current implementations are discussed. A DRL based neural combinatorial optimization strategy for online vehicle route generation is developed in [20]. Liang *et al* [11] created a novel guidance scheme based on model-based DRL where a DNN is trained as a guidance dynamics predictive model and incorporated into a model predictive path integral control framework. Xin *et al* introduces a hybrid learning strategy for DRL control of commercial aircraft where a PID baseline controller is utilized during the preliminary online training of the DNN's and then the DNN calculated control policy is utilized once it produces better performance than the baseline controller in [19]. In [21], model predictive control is used to train a DRL controller for control of a quadcopter drone. Sarabakha *et al* [16] uses DRL for UAV trajectory tracking by employing a baseline controller that simultaneously performs initial control and trains the DNN-based controller. Once the DNN controller produces sufficient policies, control switches to the neural network controller however online training continues for different sets of trajectories not used in the training phase.

DNN's are sometimes used in other applications besides DRL. For example, [7] used DNN's as approximate controllers by training on data created by model predictive control laws. Sarabakha *et al* [15] combined deep learning and fuzzy logic to create an online learning method for improved control of nonlinear systems. The DNN was pretrained offline on the recorded input-output dataset while a conventional controller controlled the system. Then when the DNN controller produced better results than the conventional, control was switched, with the neural network controller continuing to be trained online. R. Chai *et al* [4] used DNNs for real-time attitude control on a six-DOF hypersonic vehicle reentry flight application. The neural networks were trained offline using pre-generated optimal trajectory data then implemented in real-time. Many of the previous literature utilize some form of offline pre-training or otherwise initially use baseline models whilst the neural network learns the system and uncertainties and/or control policies. This is because unlike shallow neural networks, DNN's are much more difficult to train and typically take substantially more time. They are also more susceptible to problems such as vanishing/exploding gradient, making them difficult to use in purely online roles. However, with a recent advance in the training of neural networks, the power of DNN's can now be harnessed for online learning.

Recently, Bernstein *et al* [2] discovered a new learning algorithm where the weights of a neural network can be updated individually via adding or subtracting a proportion of the weight in the direction that reduces the loss function gradient. In practice, this is approximated by multiplying by the exponential function with the parameter being a learning rate factor. The attribute of this algorithm that makes it especially attractive for training neural networks is the fact that the value of the gradient is not used, only its sign. This allows the algorithm to escape the vanishing/exploding gradient problem that has previously made it very difficult to train neural networks of a significant size in an online fashion. The discovery of the algorithm was found after Bernstein *et al* defined a new distance between neural networks that could be used to facilitate training of one neural network to a theoretical 'optimal' neural network in [3]. Termed *deep latent trust* [3], this new distance definition allows for the relative change in loss function gradient as the result of network weight perturbations to be expressed as a product of the relative perturbations themselves. This in turn allows for definitions of different weight update laws that can be guaranteed to descend the loss function gradient and thus converge to the optimal weights.

This paper presents a novel control framework featuring a dynamic inversion controller combined with the Deep Neural Network Modified State Observer (DNN-MSO) - a model-less observer containing a DNN that trains the neural network online and provides a complete estimate of the plant dynamics. The observer converges to the true system by training the DNN on the error between the estimated system and the true system measurements using the multiplicative weight update tuning law and also uses an estimation error feedback term to improve estimation convergence properties. This smooths the control input and helps to mitigate high frequency chatter in the controlled system while the neural network is converging to the system dynamics. The neural network estimation property combined with the convergence guarantees in the weight tuning law proves that the DNN can adequately learn the system dynamics. This estimation is then used to replace the dynamics with the desired dynamics in the dynamic inversion controller. A Lyapunov analysis reinforces that the closed loop system is uniformly ultimately bounded with the bound being dependent on the controller feedback gain matrix and the estimation capabilities of the neural network. The controller is tested in simulation on two different systems: a highly nonlinear robotic manipulator and a short-term angle of attack trajectory tracking problem for a high performance aircraft.

The body of the paper can be summarized as follows: in section 3, the DNN-MSO is explained and the controller formulation is given ending with the derivation of the tracking error dynamics; in section 4, the multiplicative weight update law and its derivation are presented and the performance guarantees for certain learning rate values are established; in section 5, the stability analysis of the error dynamics is explored using Lyapunov analysis and the concluding results are discussed; in section 6 the controller is tested in simulation via a tracking control problem of a robot manipulator and the results are discussed and finally the paper concludes in section 7.

II. Controller Formulation

Consider the dynamics of an uncertain, nonlinear system:

$$\dot{X}(t) = f(X(t)) + B(u(t) + d(X(t))) \tag{1}$$

where $f(X(t))$ is the nonlinear, known system dynamics, B is the control matrix, $u(t)$ is the control signal and $d(X(t))$ is the state dependent uncertainty. The system can be simplified by distributing B:

$$\dot{X}(t) = f(X(t)) + \check{f}(X(t)) + Bu(t) \tag{2}$$
$$\check{f}(X(t)) = Bd(X(t))$$

The performance goal is to drive the uncertain system towards a desired system having dynamics described by:

$$\dot{X}_d(t) = f^*(X_d(t), u_n) \tag{3}$$

where u_n is some nominal control signal. To accomplish this, we employ a dynamic inversion controller to replace the original system's dynamics with the desired dynamics. However, a significant problem with dynamic inversion controllers is their sensitivity to modelling errors in the system dynamics and thus the uncertainty in the plant poses a significant challenge. To remedy this, past literature [14], [13], [5] have relied on learning the uncertainty using a modified state observer (MSO).

The MSO is a neural network observer that allows estimation of the system uncertainty in a secondary estimation loop. The typical MSO uses the known dynamics as a baseline and builds upon this using a neural network that learns the uncertainty online by comparing the state estimation and the true state measurement. Finally, the MSO contains an estimation error feedback term which helps in diminishing any high frequency dynamics that might be introduced into the controller. The equation for the MSO is given by:

$$\dot{\hat{X}}(t) = f(\hat{X}(t)) + B(u(t) + \hat{d}(X(t)) - K_2(X(t) - \hat{X}(t)) \tag{4}$$

Where $f(\hat{X}(t))$ is the known dynamics baseline, $\hat{d}(X(t))$ is the uncertainty estimate and $K_2(X(t) - \hat{X}(t))$ is the feedback term which helps to filter the control signal. Using the new learning algorithm provided by [2], we can expand upon the MSO concept.

The DNN-MSO is a special modification of the MSO featuring a deep neural network that is completely data driven. The DNN learns the complete plant dynamics and any disturbances online. The modification of the original MSO to the DNN-MSO is quite simple with the DNN-MSO being described by:

$$\dot{\hat{X}}(t) = \hat{f}(X(t)) + Bu(t) - K_2(X(t) - \hat{X}(t)) \tag{5}$$

One can see from (5), that the known dynamics and uncertainty estimation has been replaced with just one estimation term, $\hat{f}(X(t))$, that represents the entire dynamic model of the plant.

$\hat{f}(X(t))$ is the output of the deep neural network, in this case a deep multi-layered perceptron (MLP), which can be described mathematically as:

$$\hat{f}(X(t)) = W_{n-1}\sigma_{n-1}(W_{n-2}\sigma_{n-2}(W_{n-3}...W_1\sigma_1(W_{in}X(t)))) \tag{6}$$

where W_i is the weights connecting layer i to layer $i+1$ (W_{in} connects the input of the network to layer 1), σ_i is the output of layer i and $X(t)$ is the measured states. Note that for mathematical simplicity and convenience, the biases of the network are assumed implicit and thus omitted from the notation. Further note that σ_n has been omitted because the activation function of the final layer is linear as is standard procedure for a MLP network with a continuous output. Now that the new estimation model has been described, the controller can be formulated.

Since the desired objective of the controller is to drive the true system $X(t)$ to the desired system $X_d(t)$, consider the following equation:

$$\dot{X} - \dot{X}_d + K(X(t) - X_d(t)) = 0 \tag{7}$$

which enforces first order asymptotically stable error dynamics [13]. Note that K is a positive-definite, user-defined gain matrix. Substituting the system dynamics into (7):

$$f(X(t)) + \check{f}(X(t)) + Bu(t) - f^*(X_d(t), u_n) + K(X(t) - X_d(t)) = 0 \tag{8}$$

Rearranging:

$$Bu(t) = f^*(X_d(t), u_n) - (f(X(t)) + \check{f}(X(t))) - K(X(t) - X_d(t)) \tag{9}$$

Assuming that the uncertainty estimate, $\hat{f}(X(t))$, is available and $\hat{f}(X(t)) \approx f(X(t)) + \check{f}(X(t))$, this can be substituted for the system dynamics as:

$$Bu(t) = f^*(X_d(t), u_n) - \hat{f}(X(t)) - K(X(t) - X_d(t)) \tag{10}$$

Then, if B is invertible, the dynamic inversion controller can be completed by multiplying by the inverse as:

$$u(t) = B^{-1}(f^*(X_d(t), u_n) - \hat{f}(X(t)) - K(X(t) - X_d(t))) \tag{11}$$

If B is not invertible, than slack variables can be added in the manner discussed in [14]. This amounts to adding and subtracting the slack matrix B_s and the slack control signal u_s to the formulation to augment the control matrix into an invertible form. The following derivation illustrates this point.

$$Bu(t) = f^*(X_d(t), u_n) - \hat{f}(X(t)) - K(X(t) - X_d(t)) + B_s u_s - B_s u_s$$

$$Bu(t) + B_s u_s = f^*(X_d(t), u_n) - \hat{f}(X(t)) - K(X(t) - X_d(t)) + B_s u_s$$

$$\bar{B} = [B \ B_s], \quad \bar{U} = [u \ u_s]^T$$

$$\bar{B}\bar{U} = f^*(X_d(t), u_n) - \hat{f}(X(t)) - K(X(t) - X_d(t)) + B_s u_s$$

$$\bar{U} = \bar{B}^{-1}(f^*(X_d(t), u_n) - \hat{f}(X(t)) \\ - K(X(t) - X_d(t)) + B_s u_s)$$

Note that the actual control will need to be extracted from $\bar{U}$ and that the choice of slack variables affects the control signal. A methodology for selection of slack variable values is discussed in [13]. Now that the controller has been defined, the error dynamics of the closed loop system can be discussed.

First, the tracking error, estimation error and estimate tracking error are defined as:

$$e_r \triangleq X(t) - X_d(t) \tag{12}$$
$$e_a \triangleq X(t) - \hat{X}(t) \tag{13}$$
$$\hat{e}_r \triangleq \hat{X}(t) - X_d(t) \tag{14}$$

Now the tracking error can be rewritten as a combination of the estimation error and the estimate tracking error:

$$e_r = X(t) - \hat{X}(t) + \hat{X}(t) - X_d(t)$$
$$e_r = e_a + \hat{e}_r \tag{15}$$

Taking the derivative of (15):

$$\dot{e_r} = \dot{e_a} + \dot{\hat{e}}_r \tag{16}$$

Expanding the estimation error derivative:

$$\begin{aligned}\dot{e}_a &= \dot{X}(t) - \dot{\hat{X}}(t)\\ &= f(X(t)) + \check{f}(X(t)) + Bu(t) - \\ &\qquad (\hat{f}(X(t)) + Bu - K_2(X(t) - \hat{X}(t)))\\ &= f(X(t)) + \check{f}(X(t)) - \hat{f}(X(t))\\ &\qquad + K_2(X(t) - \hat{X}(t))\end{aligned} \tag{17}$$

Expanding the estimate tracking error derivative:

$$\begin{aligned}\dot{\hat{e}}_r &= \dot{\hat{X}}(t) - \dot{X}_d(t)\\ &= \hat{f}(X(t)) + Bu(t) - K_2(X(t) - \hat{X}(t))\\ &\qquad - f^*(X_d(t), u_n)\end{aligned}$$

Now using $Bu(t) = f^*(X_d(t), u_n) - \hat{f}(X(t)) - K(X(t) - X_d(t))$:

$$\begin{aligned}\dot{\hat{e}}_r &= \hat{f}(X(t)) + f^*(X_d(t), u_n) - \hat{f}(X(t))\\ &\qquad - K(X(t) - X_d(t)) - K_2(X(t)\\ &\qquad - \hat{X}(t)) - f^*(X_d(t), u_n) \qquad (18)\\ &= -(K(X(t) - X_d(t)) + K_2(X(t) - \hat{X}(t)))\end{aligned}$$

Finally, inputting (17) and (18) into (16):

$$\dot{e}_r = f(X(t)) + \check{f}(X(t)) - \hat{f}(X(t)) + K_2(X(t) - \hat{X}(t)) \\ + (-(K(X(t) - X_d(t)) + K_2(X(t) - \hat{X}(t))))$$

which simplifies to:

$$\dot{e}_r = \tilde{f}(X(t)) - K(X(t) - X_d(t)) \tag{19}$$
$$\tilde{f}(X(t)) = f(X(t)) + \check{f}(X(t)) - \hat{f}(X(t))$$

III. Multiplicative Weight Update

The backbone of any online neuro-adaptive controller is its ability to learn quickly and robustly. In this case, the driving force behind the DNN-MSO is the novel multiplicative weight update law [2]. As its name suggests, this update law is based on the multiplicative weight update algorithm discussed in [1]. The equation for this tuning law is given simply as:

$$W_{i,l} = W_{i,l} e^{\pm\eta} \tag{20}$$

More precisely, weight i of layer l is updated by multiplying by an exponential term raised to the power of the learning rate η. In this manner, each weight is updated individually with the sign of η to be determined later. The local updating property of the algorithm allows it to be invariant to complex architectures and vaguely defined layers. For the purpose of online neuro-adaptive control, the fundamental advantage of this type of update law is that it does not depend on gradient information and thus escapes the exploding/vanishing gradient problem making online training of deep neural networks feasible. The exact algorithm, the multiplicative adaptive moments based optimizer, provided by [2] is given below:

Algorithm 1: Multiplicative Adaptive Moments Algorithm

```
Select Algorithm Hyperparameters: $\eta, \eta^*, \sigma^*, \beta$
Initialize Weights
$\bar{g} \leftarrow 0$    Initialise second moment estimate
repeat
    $g \leftarrow StochasticGradient()$
    $\bar{g}^2 \leftarrow (1-\beta)g^2 + \beta\bar{g}^2$
    W $\leftarrow W \odot exp[-\eta\ sign\ W \odot clamp_{\eta^*/\eta}(g/\bar{g})]$
    W $\leftarrow clamp_{\sigma^*}(W)$
until converged;
```

The hyperparameters of the algorithm are: the learning rate (η), the maximum weight perturbation factor (η^*), the maximum weight value (σ^*) and the convex combination factor (β). Note that the algorithm has a hard restriction on the size of the weights, set by the user, which enforces a hard upper and lower bound on the network weights. Although a learning rate is specified, the actual weight perturbation factor magnitude may be more, up to a user-defined maximum, so the maximum weight change for one update is hard-bounded as well. The actual update mechanism is very close to the vanilla multiplicative update rule shown in (20) with a few changes for increased practicality. For example, $g/\bar{g}$ is $O(1)$ and therefore is just a higher precision version of $sign(g)$ which captures more information about the gradient without subjecting the algorithm to the exploding/vanishing gradient problems. It is also obvious that the sign of the update factor is determined based off of $sign(W)$ and $sign(g)$ with the reasoning behind this discussed in the following lemma and theorem.

The theory behind the mutliplicative weight update tuning law is fundamentally based on a novel definition of functional distance for neural networks termed *deep relative trust*, introduced in [3], which is described as follows. Consider a neural network having L layers and weight parameters $W = (W_1, W_2, ..., W_L)$ with weight perturbations $\Delta W = (\Delta W_1, \Delta W_2, ..., \Delta W_L)$. Consider a loss function over the network parameters: $\mathcal{L}(W)$. Let $g_k(W) = \nabla_{w_k}\mathcal{L}(W)$ denote the gradient of the loss. Then the gradient breakdown is bounded by:

$$\frac{\|g_k(W + \Delta W) - g_k(W)\|_F}{\|g_k(W)\|_F} \leq \prod_{l=1}^{L}\left(1 + \frac{\|\Delta W_l\|_F}{\|W_l\|_F}\right) - 1 \tag{21}$$

for all $k = 1, ..., L$. As can be seen from (21), the relative change in Loss function gradient with respect to each weight layer is upper bounded by a product of the relative perturbation of the weights over all layers. Thus, we have a tractable model for gradient breakdown with which we can build a descent lemma which is the cornerstone of the Multiplicative Adaptive Moments optimiser theorem.

Next, we must introduce a general lemma for guaranteeing gradient descent.

Lemma 1. *Consider a continuously differentiable function $\mathcal{L} : \mathbb{R}^n \mapsto \mathbb{R}$ that maps $W \mapsto \mathcal{L}(W)$. Define $W \triangleq (W_1, W_2, ..., W_L)$ where L is the total number of parameter groups. Similarly, define $\Delta W \triangleq (\Delta W_1, \Delta W_2, ..., \Delta W_L)$ as the perturbation of W. Let θ_k measure the angle between ΔW_k and the negative gradient $-g_k(W) = -\nabla_{W_k}\mathcal{L}(W)$. Then:*

$$\mathcal{L}(W + \Delta W) - \mathcal{L}(W) \leq -\sum_{k=1}^{L} \|g_k(W)\|_F \|\Delta W_k\|_F \left[cos(\theta_k) - \max_{t\in[0,1]} \frac{\|g_k(W + t\Delta W) - g_k(W)\|_F}{\|g_k(W)\|_F}\right] \tag{22}$$

Proof. By the fundamental theorem of Calculus, we have:

$$\mathcal{L}(W + \Delta W) - \mathcal{L}(W) = \sum_{k=1}^{L}\left[g_k(W)^T \Delta W_k + \int_0^1 [g_k(W + t\Delta W) - g_k(W)]^T \Delta W_k \, dt\right] \tag{23}$$

Replacing the first term on the right side of (23) with the cosine formula for the dot product and using the integral estimation lemma to bound the second term, we arrive at:

$$\mathcal{L}(W + \Delta W) - \mathcal{L}(W) \leq -\sum_{k=1}^{L} \|g_k(W)\|_F \|\Delta W_k\|_F \left[cos\theta_k - \max_{t\in[0,1]} \frac{\|g_k(W + t\Delta W) - g_k(W)\|_F}{\|g_k(W)\|_F}\right] \tag{24}$$

□

Lemma 1 now gives us our formal descent guarantee conditions. More precisely, this is when:

$$\max_{t\in[0,1]} \frac{\|g_k(W + t\Delta W) - g_k(W)\|_F}{\|g_k(W)\|_F} < cos(\theta_k) \quad (for\ k = 1, ..., L) \tag{25}$$

We see that the descent guarantee stems from the gradient breakdown. Now we can use our previously defined notion of *deep relative trust* to derive the condition over the learning rate in which descent is guaranteed. This is expressed in the following theorem.

Theorem 1. *Let L by the continuously differentiable loss function of a neural network of depth L that obeys deep relative trust. For $k = 1, ..., L$, let $0 \leq \gamma_k \leq \frac{\pi}{2}$ denote the angle between $|g_k(W)|$ and $|W_k|$ (where $|\textbf{.}|$ denotes element-wise absolute value). Then the following update law:*

$$W \rightarrow W + \Delta W = W \odot [1 - \eta \ sign \ W \odot sign(g(W))] \tag{26}$$

guarantees a decrease in the loss function provided that:

$$\eta < (1 + cos(\gamma_k)^{\frac{1}{L}} - 1, \quad (for \ all \ k = 1, ..., L) \tag{27}$$

Proof. Using the definition of *deep relative trust* (21), we have:

$$\begin{aligned}\max_{t\in[0,1]} &\frac{\|g_k(W + t\Delta W) - g_k(W)\|_F}{\|g_k(W)\|_F} \\ &\leq \max_{t\in[0,1]} \prod_{l=1}^{L}\left(1 + \frac{\|t\Delta W_l\|_F}{\|W_l\|_F}\right) - 1 \\ &\leq \prod_{l=1}^{L}\left(1 + \frac{\|\Delta W_l\|_F}{\|W_l\|_F}\right) - 1\end{aligned} \tag{28}$$

Now using the results of the descent lemma (1), descent is guaranteed if:

$$\prod_{l=1}^{L}\left(1 + \frac{\|\Delta W_l\|_F}{\|W_l\|_F}\right) < 1 + cos(\theta_k) \quad (for \ all \ k = 1, ..., L) \tag{29}$$

where θ_k measures the angle between ΔW and $-g_k(W)$. Considering (26), the perturbation is defined as $\Delta W = -\eta|W| \odot sign \ g(W)$ where $\|\Delta W_*\|_F / \|W_*\|_F = \eta$ for any possible subset of weights W_*. Let $\measuredangle(\textbf{.}, \textbf{.})$ denote the angle between its operators. Now, θ_k and γ_k are related by:

$$\begin{aligned}\theta_k &= \measuredangle(\Delta W_k, -g_k(W)) \\ &= \measuredangle(-\eta \ |W_k| \odot sign(g_k(W)), -g_k(W)) \\ &= \measuredangle(|W_k| \odot sign(g_k(W)), g_k(W)) \\ &= \measuredangle(|W_k|, |g_k(W)|) \\ &= \gamma_k\end{aligned}$$

Substituting the two previous results into (29), we get:

$$\prod_{l=1}^{L}(1 + \eta) < 1 + cos(\gamma_k)$$

which becomes

$$(1+\eta)^L < 1 + cos(\gamma_k)$$

and finally

$$\eta < (1 + cos(\gamma_k)^{\frac{1}{L}} - 1 \tag{30}$$

□

One final remark for this section is that the update law used in the proof and that used in the algorithm are different. It is worth mentioning that for the practical algorithm we take advantage of the approximation: $w(1 \pm \eta) \approx we^{\pm\eta}$.

IV. Stability Analysis

The error dynamics were presented in section 3, culminating in (19). The results of section 4 support the embedded DNN's capacity to learn and approximate a function to a certain degree (proves that the actual weights approach the true weights over time) and exerts hard bounds on the weights. However the error dynamics of the function still require analysis. Consider the Lyapunov candidate function:

$$L = e_r^T e_r \tag{31}$$

Taking the derivative:

$$\dot{L} = e_r^T \dot{e}_r + \dot{e}_r^T e_r$$

$$\dot{L} = e_r^T(\tilde{f}(X(t)) - K(X(t) - X_d(t))) + (\tilde{f}(X(t)) - K(X(t) - X_d(t)))^T e_r \tag{32}$$

Considering $e_r = X(t) - X_d(t)$, (32) becomes:

$$\dot{L} = e_r^T(\tilde{f}(X(t)) - Ke_r) + (\tilde{f}(X(t)) - Ke_r)^T e_r$$

$$\dot{L} = e_r^T \tilde{f}(X(t)) - e_r^T K e_r + \tilde{f}(X(t))^T e_r - e_r^T K^T e_r$$

$$\dot{L} = 2e_r^T \tilde{f}(X(t)) - 2e_r^T K^T e_r \tag{33}$$

By Cauchy-Schwartz inequality:

$$|e_r^T \tilde{f}(X(t))| \leq \|e_r\|_2 \|\tilde{f}(X(t))\|_2 \tag{34}$$

Because of the results of section 4 and thus the stability guarantee of the DNN for appropriate parameter choices, $\|\tilde{f}(X(t))\|_2 \leq \epsilon$, which in turn makes (34):

$$|e_r^T \tilde{f}(X(t))| \leq \|e_r\|_2 \|\tilde{f}(X(t))\|_2 \leq \|e_r\|_2 \epsilon \tag{35}$$

Let the eigenvalues of the gain matrix K be ordered such that $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$. Then, since $K > 0$, the following inequality holds:

$$\lambda_1(K) e_r^T e_r \leq e_r^T K e_r \leq \lambda_n(K) e_r^T e_r \tag{36}$$

Using $e_r^T e_r = \|e_r\|_2^2$ and substituting the upper bound of (35) and the lower bound of (36) into (33), we arrive at:

$$\dot{L} \leq 2\|e_r\|_2 \epsilon - 2\lambda_1(K)\|e_r\|_2^2 \tag{37}$$

Which implies $\dot{L} \leq 0$ when:

$$\|e_r\|_2 \epsilon \leq \lambda_1(K)\|e_r\|_2^2$$

$$\|e_r\|_2 \geq \frac{\epsilon}{\lambda_1(K)} \tag{38}$$

Equation (38) implies the system is uniformly ultimately bounded (UUB) with the bound on the tracking error (e_r) magnitude depending on the upper bound of the neural network approximation error (ϵ) and the smallest eigenvalue of the gain matrix K $(\lambda_1(K))$. Because the weight update law guarantees descent and thus stability in learning and because the controller formulation was based on stable error dynamics, the system is always stable if $\lambda_1(K) > 0$.

V. Robot Manipulator Simulation Results

To test the performance of the proposed controller, a simulation was performed using a 2 link planar robot manipulator. The dynamics of this system are given below as:

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + G(q) = \tau(t) \tag{39}$$

where $q \in \mathbb{R}^2$ are the joint angles, $M(q)$ is the inertia matrix, $V_m(q, \dot{q})$ is the Coriolis/centripetal matrix, $G(q)$ is the gravity vector and τ_t is the control torque. (39) can be expressed in matrix form via:

$$\begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ -M^{-1}(q)(V_m(q,\dot{q})\dot{q} + G(q)) \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1}(q) \end{bmatrix} \tau_t \tag{40}$$

It is desired to have the joint angles follow the reference trajectory of:

$$\begin{bmatrix} q_{1,d} \\ q_{2,d} \end{bmatrix} = \begin{bmatrix} sin(0.5t) \\ cos(0.5t) \end{bmatrix}$$

The manipulator links had length $l_1 = 1m, l_2 = 1m$ and masses $m_1 = 1kg, m_2 = 2.3kg$. The controller PD gains and MSO gains were:

$$K = \begin{bmatrix} 7.5 & 0 & 0 & 0 \\ 0 & 7.5 & 0 & 0 \\ 0 & 0 & 7.5 & 0 \\ 0 & 0 & 0 & 7.5 \end{bmatrix}, \quad K_2 = \begin{bmatrix} 30 & 0 & 0 & 0 \\ 0 & 30 & 0 & 0 \\ 0 & 0 & 30 & 0 \\ 0 & 0 & 0 & 30 \end{bmatrix}$$

The neural network used had four layers, sigmoidal hidden layer activation functions and its weights were initialised as $W \sim N(0, \frac{3.6}{\sqrt{L}})$ according to [8]. The weight update parameters were $\eta = 0.001$, $\eta_{max} = 100\eta$, $\sigma_{max} = 1250$, $B = 0.999$. The simulation time was 10 seconds, the integration time step was 0.001. The initial conditions were:

$$\begin{bmatrix} q_{1,0} \\ q_{2,0} \\ \dot{q}_{1,0} \\ \dot{q}_{2,0} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0 \\ 0 \end{bmatrix}, \quad \hat{q}_0 = q_0$$

Fig. 1-4 display the results of the simulation. Fig. 1 shows the tracking performance of the system with the neural network turned off. While the actual trajectories loosely follow the desired trajectories, it is obvious that there is significant error in the tracking performance. In Fig. 2, the desired and actual state trajectories are displayed, this time using the DNN-MSO to learn the system. The performance is much better when compared to Fig. 1 with the actual states converging to the desired states very quickly and showing excellent tracking throughout the simulation. The trajectory tracking error can be seen in Fig. 3. Similar to the trajectory figure, the error figure displays the fast convergence and that the error stays very close to zero throughout time with some very minor oscillations in the error 1 signal. The torque inputs, Fig. 4, seem to be reasonable with moderate amplitude.

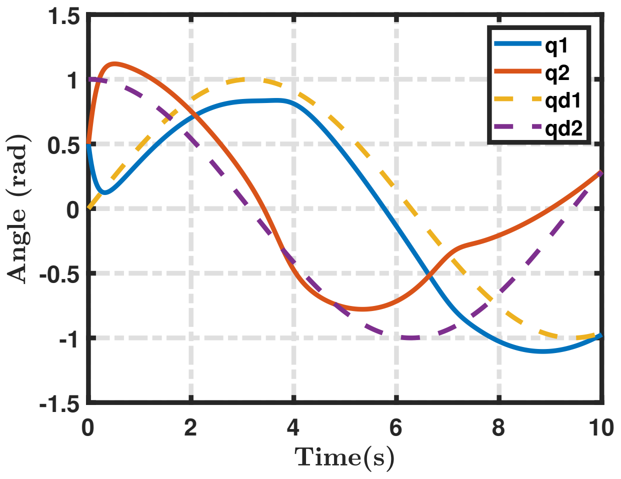

Fig. 1: Trajectory Tracking: No NN

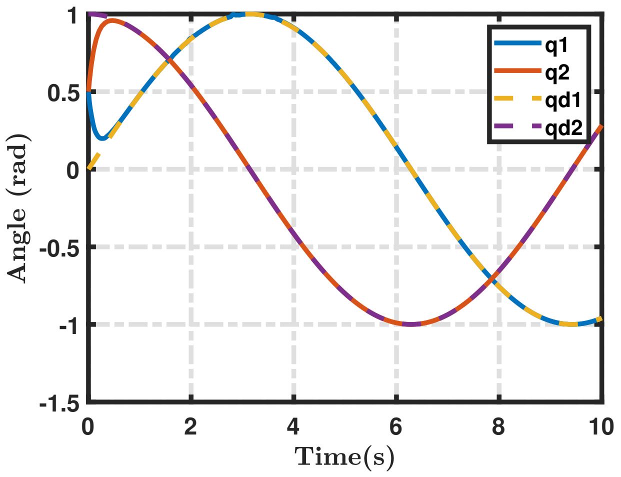

Fig. 2: Trajectory Tracking: with NN

VI. Conclusion

This paper presented a novel model-free dynamic inversion controller featuring a DNN that was trained online using the multiplicative weight update training algorithm. The derivation of the controller was given as well as a stability analysis for a tracking problem and the update law used to train the network was shown and discussed. A simulation showing a joint angle tracking problem with a highly nonlinear robotic manipulator demonstrated the effectiveness of the controller. The potential of a DNN in a nonlinear controller was displayed as the network was able to learn quickly and effectively, leading to

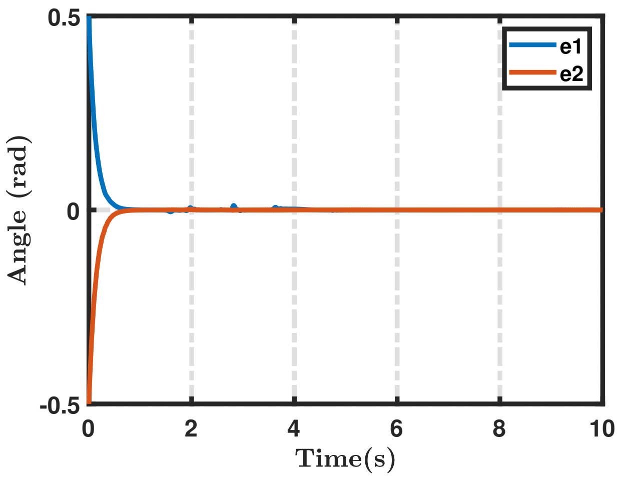

Fig. 3: Error: with NN

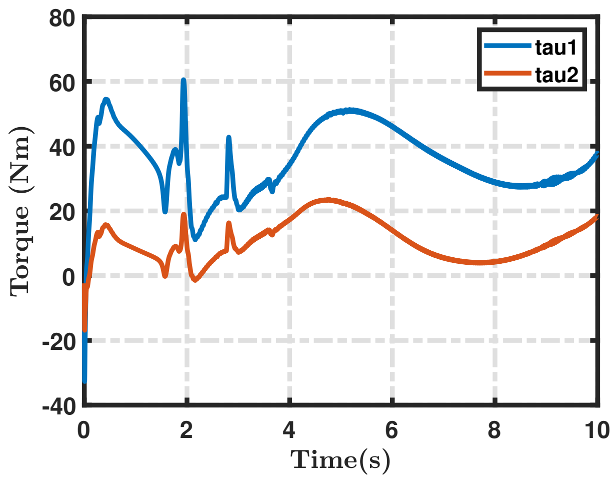

Fig. 4: Torque: with NN

exceptional tracking performance as compared to a standard non-adaptive feedback controller. To the author's knowledge, this is the first work to feature a truly online training deep neural network in a dynamic inversion controller.

Future work includes implementing this controller on a real plant and examining the results. Real world implementation comes with its own set of challenges namely overcoming delay that might be present due to hardware restrictions and thus might affect the controller performance. Another problem to overcome is the algorithm's sensitivity to initial conditions. This was somewhat overcome in this paper by utilizing the initialization strategy outlined by [8] but there exists much room for improvement to achieve consistent results. Finally, the author's would like to extend the theory presented in this paper to backstepping control to be more utilizable in general nonlinear systems.

References

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale. "The Multiplicative Weights Update Method: a Meta-Algorithm and Applications". In: *Theory of Computing* 8.1 (2012), pp. 121–164. DOI: 10.4086/toc.2012.v008a006. URL: https://doi.org/10.4086/toc.2012.v008a006.

[2] Jeremy Bernstein et al. *Learning compositional functions via multiplicative weight updates*. 2021. arXiv: 2006.14560 [cs.NE].

[3] Jeremy Bernstein et al. *On the distance between two neural networks and the stability of learning*. 2021. arXiv: 2002.03432 [cs.LG].

[4] R. Chai et al. "Six-DOF Spacecraft Optimal Trajectory Planning and Real-Time Attitude Control: A Deep Neural Network-Based Approach". In: *IEEE Transactions on Neural Networks and Learning Systems* 31.11 (2020), pp. 5005–5013. DOI: 10.1109/TNNLS.2019.2955400.

[5] A. Ghafoor et al. "Event Triggered Neuro-Adaptive Controller (ETNAC) Design for Uncertain Linear Systems". In: *2018 IEEE Conference on Decision and Control (CDC)* (2018). DOI: 10.1109/cdc.2018.8618962.

[6] Huang Yi et al. "A study on Deep Neural Networks framework". In: *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. 2016, pp. 1519–1522. DOI: 10.1109/IMCEC.2016.7867471.

[7] B. Karg and S. Lucia. "Efficient Representation and Approximation of Model Predictive Control Laws via Deep Learning". In: *IEEE Transactions on Cybernetics* 50.9 (2020), pp. 3866–3878. DOI: 10.1109/TCYB.2020.2999556.

[8] Siddharth Krishna Kumar. "On weight initialization in deep neural networks". In: *CoRR* abs/1704.08863 (2017). arXiv: 1704.08863. URL: http://arxiv.org/abs/1704.08863.

[9] H. Li et al. "Deep reinforcement learning: Framework, applications, and embedded implementations: Invited paper". In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2017, pp. 847–854. DOI: 10.1109/ICCAD.2017.8203866.

[10] Z. Li et al. "Missile Guidance Law Based on Robust Model Predictive Control Using Neural-Network Optimization". In: *IEEE Transactions on Neural Networks and Learning Systems* 26.8 (2015), pp. 1803–1809. DOI: 10.1109/TNNLS.2014.2345734.

[11] C. Liang et al. "Learning to Guide: Guidance Law Based on Deep Meta-Learning and Model Predictive Path Integral Control". In: *IEEE Access* 7 (2019), pp. 47353–47365. DOI: 10.1109/ACCESS.2019.2909579.

[12] Y. Liu, Y. Zhou, and X. Li. "Attitude Estimation of Unmanned Aerial Vehicle Based on LSTM Neural Network". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–6. DOI: 10.1109/IJCNN.2018.8489118.

[13] R. Padhi, S.n. Balakrishnan, and N. Unnikrishnan. "Model-following neuro-adaptive control design for non-square, non-affine nonlinear systems". In: *IET Control Theory Applications* 1.6 (2007), pp. 1650–1661. DOI: 10.1049/iet-cta:20060364.

[14] Karthik Rajagopal et al. "Neuroadaptive Model Following Controller Design for Non-Affine and Non-Square Aircraft Systems". In: *AIAA Guidance, Navigation, and Control Conference* (2009). DOI: 10.2514/6.2009-5737.

[15] A. Sarabakha and E. Kayacan. "Online Deep Fuzzy Learning for Control of Nonlinear Systems Using Expert Knowledge". In: *IEEE Transactions on Fuzzy Systems* 28.7 (2020), pp. 1492–1503. DOI: 10.1109/TFUZZ.2019.2936787.

[16] A. Sarabakha and E. Kayacan. "Online Deep Learning for Improved Trajectory Tracking of Unmanned Aerial Vehicles Using Expert Knowledge". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 7727–7733. DOI: 10.1109/ICRA.2019.8794314.

[17] X. Wang, X. Ye, and W. Liu. "Online Adaptive Critic Learning Control of Unknown Dynamics With Application to Deep Submergence Rescue Vehicle". In: *IEEE Access* 8 (2020), pp. 96565–96580. DOI: 10.1109/ACCESS.2020.2996270.

[18] Y. Wang et al. "Unmanned Surface Vehicle Course Tracking Control Based on Neural Network and Deep Deterministic Policy Gradient Algorithm". In: *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*. 2018, pp. 1–5. DOI: 10.1109/OCEANSKOBE.2018.8559329.

[19] M. Xin et al. "Online Hybrid Learning to Speed Up Deep Reinforcement Learning Method for Commercial Aircraft Control". In: *2019 3rd International Symposium on Autonomous Systems (ISAS)*. 2019, pp. 305–310. DOI: 10.1109/ISASS.2019.8757756.

[20] J. J. Q. Yu, W. Yu, and J. Gu. "Online Vehicle Routing With Neural Combinatorial Optimization and Deep Reinforcement Learning". In: *IEEE Transactions on Intelligent Transportation Systems* 20.10 (2019), pp. 3806–3817. DOI: 10.1109/TITS.2019.2909109.

[21] T. Zhang et al. "Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 528–535. DOI: 10.1109/ICRA.2016.7487175.