Working Memory Augmentation for Improved Learning in Neural Adaptive Control

Deepan Muthirayan[a] and Pramod P. Khargonekar [a]

***Abstract*— In this paper, we propose a novel control architecture, inspired from neuroscience, for adaptive control of continuous time systems. The objective here is to design control architectures and algorithms that can *learn and adapt quickly to changes that are even abrupt*. The proposed architecture, in the setting of standard neural network (NN) based adaptive control, augments an external working memory to the NN. The learning system stores, in its external working memory, recently observed feature vectors from the hidden layer of the NN that are relevant and forgets the older irrelevant values. It retrieves relevant vectors from the working memory to modify the final control signal generated by the controller. The use of external working memory improves the context inducing the learning system to search in a particular direction. This directed learning allows the learning system to find a good approximation of the unknown function even after abrupt changes quickly. We consider two classes of controllers for illustration of our ideas (i) a model reference NN adaptive controller for linear systems with matched uncertainty (ii) backstepping NN controller for strict feedback systems. Through extensive simulations and specific metrics we show that memory augmentation improves learning significantly even when the system undergoes sudden changes. Importantly, we also provide evidence for the proposed mechanism by which this specific memory augmentation improves learning.**

I. Introduction

Human brain is arguably the best learning system known to date. This fact has inspired many different branches of research in science, engineering and computing. To achieve goals such as autonomy, judgment, and common sense in artificial systems, it will likely be necessary to take inspiration from the rapidly expanding knowledge in neuroscience and cognitive science. In control systems, an example of such efforts is neural network based adaptive control.

Modern machine learning approaches such as deep neural networks require large amounts of data for their training. This is in contrast to humans who can learn from relatively small number of examples, sometimes even one example. This is potentially highly relevant in adaptive control systems. In this paper, we explore one of the central control challenges for such adaptive control systems, which is *the ability to learn and adapt to changes in conditions or contexts quickly*.

Memory plays a central role in such learning and cognition tasks for humans. In neuroscience, there are three memory systems namely semantic, episodic and non-declarative (of which procedural memory is a subset) [1], [2]. Each system stores or encodes information differently and has its strengths and weaknesses. Authors in [3] suggest that humans derive their incredible ability to learn from just few examples by an effective combination of these different kinds of memory systems.

Inspired by these insights in neuroscience and motivated by the opportunity in adaptive control, we focus on the following questions: can control algorithms improve learning by incorporating such memory structures? If so what is the appropriate architecture? Is there an universal architecture or is it problem dependent? And in what scenarios does it improve learning? These are hard questions and they have, so far, been relatively under-explored. They form the essential basis for our research agenda under the theme of *learning for control.* In the context of traditional thinking in dynamic systems and control, state of the nonlinear controller constitutes the "memory" in the controller. However, here we are using knowledge about human memory systems to posit specific memory modules that will complement the state of the dynamic nonlinear controller and potentially lead to new learning and control capabilities. Specifically, we draw on some very recent developments in machine learning, Neural Turing machines (NTM), that have taken a similar approach, and incorporate these in adaptive control.

As an initial step towards these larger questions and goals, we consider a well-studied adaptive control setting. The base architecture is the standard adaptive control architecture that uses a neural network as the function approximator of the nonlinear uncertainty [4]–[9]. In addition, the nonlinear uncertainties can vary with time including variations that are *abrupt* or *sudden*. The objective for the controller is to adapt quickly even after such abrupt changes. We make the assumptions that the controller can observe the state of the system and the abrupt changes are not large for this initial exploration. The question then is, can memory structures inspired from neuroscience improve these adaptive control algorithms? And if so what is the architecture? And what are the algorithms?

A traditional approach to improve speed of learning is to increase the learning rate. But in closed loop control applications, this can give rise to high frequency oscillations [10]. Modifications have been proposed to deal with such high frequency oscillations [10]–[12]. What is to be noted is that the increase in the learning speed in these methods is a consequence of the increase in the learning rate. So these methods do not fundamentally address the challenge of improving the speed of learning and response of adaptive controllers. By contrast, here we propose a new control architecture that is novel in terms of how the controller

[a] Electrical Engineering and Computer Sciences, University of California, Irvine, CA 92697. Email: {dmuthira, pramod.khargonekar}@uci.edu

Supported in part by the National Science Foundation under Grant Number ECCS-1839429.

uses information from past learning episodes to learn and respond.

In section II we propose the *Memory Augmented Neural Network* (MANN) adaptive controller and give as an example the extension of the standard *Model Reference Adaptive Controller* (MRAC). We then give some background from neuroscience and provide examples of such augmentation in machine learning literature as well. In section IV-A we discuss the design of the memory interface. In particular we introduce the central idea behind the design which is *induced learning*. Finally in section V we provide a detailed set of simulation results and discussion substantiating the improvements in learning obtained by memory augmentation. We have not included any results on stability analysis in this paper and that is a topic of our current research. But our initial explorations indicate that it may be possible to generalize existing results on stability of NN based adaptive control to our setting.

II. Memory Augmented Control Architecture

In this section, we introduce a novel memory augmented control architecture for adaptive control of continuous time systems. The intuitive idea here is to leverage the capabilities offered by the combination of an implicit memory like a NN and an external working memory that can store relevant information from previous learning episodes.

1) Proposed Architecture: Our envisioned general architecture for combining traditional dynamic feedback control with an external memory is depicted in Fig. 1. There are numerous possible ways to develop concrete versions of this general architecture. In this paper, we specialize it to Fig. 1 which extends the standard NN adaptive control architecture by augmenting the NN with an external working memory. The output of the *control law* block is the control input u to the plant. The plant output is the system state. The plant output is fed to the error evaluator which calculates either (i) the error between system state and a pre-specified trajectory in the case of a trajectory tracking problem, or (ii) error between a reference model's state and the state of the system in the case of MRAC. The error is used to calculate the update to the neural network parameters. The output of the neural network is fed to the control law block. Typically, the NN output is used to compensate the nonlinear uncertainty in the system dynamics. The proposed architecture introduces an "external working memory" to the NN. The NN can write or read from the memory to modify its output. In a later section, we will provide a detailed description of this memory interface.

2) Example (MRAC): Consider the system given in (1).

$$\dot{x} = Ax + B(u + f(x)) + B_r r \tag{1}$$

This is a linear plant, whose system matrices A and B are known, with a matched nonlinear uncertainty $f(x)$ [10]. Figure 2 gives the memory augmented MRAC architecture for this problem. The variable x is the system state, r is the command signal, and u is the control input. The control law block's output is supposed to ensure that the system

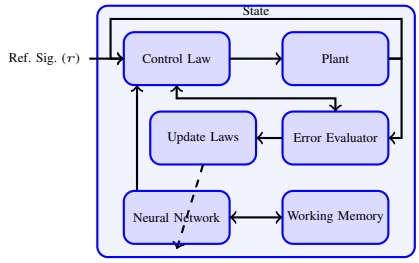

Fig. 1. Memory Augmented NN Direct Adaptive Control Architecture

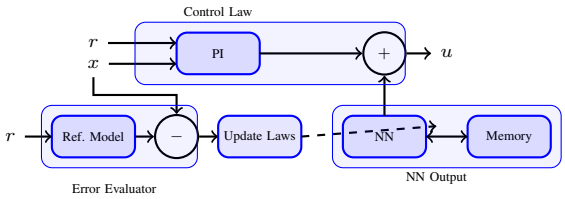

Fig. 2. MRAC Augmented with Memory

state x tracks x_{ref}, which is the state of a given reference model $\dot{x}_{ref} = A_{ref}x_{ref} + B_r r$. The NN output is used by the control law to compensate the unknown $f(x)$. Here, the neural network output is given by $u_{ad} = -\hat{f}$ (refer Notation) and the base control term is PI control. Without the working memory, the architecture is the standard MRAC architecture [10], [13].

Notation: The system state is denoted by, $x \in \mathbb{R}^n$. Throughout the paper, as in standard NN adaptive control, we use a two layer neural network with a sigmoidal activation function in the hidden layer. For a given nonlinear function $f(x)$ and a compact set $\mathcal{C}$ it follows by the universal approximation theorem, that for any c there exists a NN, which includes an extra bias term b_w, such that,

$$f(x) = W^T\sigma(V^T x + b_v) + b_w + \epsilon \ , \ \forall \ x \in \mathcal{C}, \text{ where } \|\epsilon\| \leq c$$

Including the extra term, b_w, allows us to illustrate the learning principle using simple examples. Denote the NN used to approximate the function $f(x)$ by $\hat{f} = \hat{W}^T\sigma(\hat{V}^T x + \hat{b}_v) + \hat{b}_w$. Later, we use a function called softmax(.), whose input is a vector z and output is also a vector of the same length. The ith component of softmax(.) is given by

$$\text{softmax(z)}_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}. \tag{2}$$

We introduce two other vector functions which appear in the NN update laws. We denote these functions by $\hat{\sigma}$ and $\hat{\sigma}'$ which are defined by:

$$\hat{\sigma} = \begin{bmatrix} \sigma(\hat{V}^T x + \hat{b}_v) \\ 1 \end{bmatrix}$$

$$\hat{\sigma}' = \begin{bmatrix} \text{diag}(\sigma(\hat{V}^T x + \hat{b}_v) \odot (1 - \sigma(\hat{V}^T x + \hat{b}_v))) \\ \mathbf{0}^T \end{bmatrix} \tag{3}$$

where $\mathbf{0}$ is a zero vector of dimension equal to the number of hidden layer neurons. The class of controllers that are

NN based adaptive controllers have a base control term and a NN output term. We denote these terms by u_{bl} and u_{ad} respectively. The overall control input is then the summation of the two terms i.e. $u = u_{bl} + u_{ad}$.

III. Memory Systems in the Brain and Neural Turning Machines

In this section we first introduce the predominant types of memory systems in the human brain and how they have motivated our thinking. We then recap some recent developments in machine learning inspired by the idea of working memory in neuroscience. This learning system essentially augments an external working memory to a NN. The primary motivation of this series of work was to build a learning system that can learn from fewer examples. We then give a brief discussion on how this learning system learns from fewer examples.

A. Memory Systems

As mentioned before, neuroscience identifies three kinds of memory systems (i) *semantic* (ii) *non-declarative* (of which procedural memory is a subset) (iii) *episodic memory* [1], [2]. According to the classic work by Tulvig et. al [14], *semantic memory* "is a mental thesaurus, organized knowledge a person possesses about words and other verbal symbols, their meaning and referents, about relations among them, and about rules, formulas, and algorithms for the manipulation of these symbols, concepts and relations". In short, semantic memory refers to the *models*, *concepts* of the environment the brain interacts with and the *algorithms* for manipulating them. From the point of view *reinforcement learning* (RL) [15] or control, semantic memory is related to *model-based* learning [3]. Semantic memory is powerful in the sense that the brain can employ these learned concepts and algorithms in previously unseen environments or contexts. Their disadvantage stems from the computational demands involved in using them, for example: model-based learning in RL [3].

Procedural memory is a memory that is gathered over several episodes of learning while performing a task. It is a memory that learns the procedure of performing a task. It is a slow learning memory and can generalize to different context within the same task. For example, in the game of chess expert chess players learn complex algorithms (not precisely expressible as a sequence or set of rules) through experience, leading them to superior performance. This is an example of procedural memory. From the point of view reinforcement learning procedural memory is related to *model-free* learning [3]. There is evidence that the brain employs both semantic and procedural memory systems for RL tasks [3].

Episodic memory, in contrast, stores raw history of past experiences. Loosely, episodic memory refers to memory of specific events that were personally experienced and remembered. For example, the memory of the experience of dinner on one's 18th birthday (e.g., the taste of the food, the location etc.) falls under the umbrella of episodic memory. There is evidence that episodic memory can improve learning with sparse data for RL tasks [16]. Their disadvantage is that they generalize very poorly unlike semantic or procedural memory [3].

Working memory system models are combinations of memory systems and a central executive which can manipulate the information in these systems like recalling from past experience [17]. Working memory systems are clearly demonstrated in a chess player. An expert chess player can recall moves from an old game or the position of the pieces from an earlier part of the game. This enables the player to quickly understand the game situation and make a better decision for the next move. It follows that an expert player is able to make better decisions by using a combination of working memory, with episodic memory as its memory system, and other memory systems such as semantic or procedural memory. It is this combined benefit of two memory systems, through a working memory, that is exploited in many learning systems proposed in the machine learning literature [18]–[20] and the architecture we propose.

B. Use of Memory Systems in Machine Learning

It was proposed as early as 2001 that neural networks with memory capacities [21] could be quite capable of learning from sparse data. For example authors in [22] showed that, "LSTMs (Long Short Term Memory) which have an inherent memory can quickly learn never-before-seen quadratic functions with a low number of data samples". Starting in 2014, Neural Turing Machines (NTMs) [18] being among the first, architectures with external working memory were proposed [19], [20]. The interface of the external working memory was designed such that the process of reading and writing from the memory is *differentiable* or learnable via *error backpropagation* algorithm [23]. It was shown in these works that the addition of the external working memory improved the performance of LSTMs across many problems.

In the standard implementation of NTMs, the interface between NN and the working memory (as in Fig. 1 of the proposed architecture) has two operations (i) *Memory Write* and (ii) *Memory Read*. The working memory stores $\{$*key, value*$\}$ pairs, denoted by $\{k_i, v_i\}$ where i takes values in some (finite) set.

The Memory Write equation in NTM has a *forget term* and an *update term*. The forget term removes contents from the memory that are irrelevant and the update term updates the contents of the memory with the new information provided it is relevant. This gives NTMs the capability to retain information and also update with the new information, depending on their relevance.

The Memory Read operation uses an addressing mechanism to read from the memory. In a typical addressing mechanism, a NN is used to generate a *query vector* q based on the current context. This query is then matched with the keys to determine the values to be read from the memory. Provided the NN is trained to generate an appropriate query, this operation should then retrieve useful information from the memory. This information that is read from the memory is used by NTMs to produce their final output.

What enables NTMs to learn from fewer examples? An external working memory with its interface allows the learning system, such as NTMs, to store and retrieve relevant information to enrich the context for its operation. With an external source that provides a better context, the learning system may not need to learn from scratch. Instead it can learn by accounting for the context provided by the external memory, potentially increasing its speed of learning. NTMs have this capability because (i) they can learn to store and retrieve context enriching information using the write and read operation, and (ii) their learning algorithm, which is based on backpropagation, is effective at *credit assignment* [24].

IV. Working Memory Interface and Control Algorithm

In this section, we provide a detailed description of the interface of the external working memory that augments the NN in Fig. 1. We also motivate and justify, using an example, how memory augmentation improves learning by an *induced learning mechanism*. Finally, we specify the NN update laws and the control algorithms.

A. Memory Interface

The working memory proposed here is based on the ideas discussed in section III-B. We emphasize that the key innovation lies in the specification of the query vector q, write vector a and how the NN output is modified. We denote the memory state by matrix μ, output of Memory Read by M_r and the augmented NN output by u_{ad}. The matrix μ is of size $n_s \times N$. Denote the i-th column vector of matrix μ by μ_i. First, we give the equations for Memory Write, Memory Read and NN output and then discuss them in detail.

$$\text{Memory Write: } \dot{\mu}_i = -z_i \mu_i + c_w z_i a + z_i \hat{W} q_\mu^T \tag{4}$$

$$\text{Memory Read: } M_r = \mu z,\ z = \text{softmax}(\mu^T q) \tag{5}$$

$$\text{NN Output: } u_{ad} = -\hat{W}^T \left(\sigma(\hat{V}^T x + \hat{b}_v) + M_r\right) - \hat{b}_w \tag{6}$$

A.1. Memory Write: The Memory Write equation (4) is similar to NTM and consists of three terms: (i) a forget term (the first term), (ii) an update term using the new information (the second term) and (iii) an additional update term (the third term), which arises from the interaction of the NN learning algorithm with the memory contents.

The main novelty here arises from what we call the *write vector*, signifying *new information,* denoted by a in the Memory Write equation (4). We propose to set a to be the output of the hidden layer:

$$a = h = \sigma(\hat{V}^T x + \hat{b}_v) \tag{7}$$

The middle term in equation (4) updates the memory content using the write vector a at a rate proportional to the weight z_i. This weight, discussed in more detail below, reflects the relevance of a in modifying memory content μ_i.

The first term in equation (4) forgets the memory's content at the ith location at the same rate z_i. Thus, the Memory Write operation updates and forgets at a rate proportional to the relevance of the write vector. This gives memory the ability to retain information and also update its contents with the new information. The forget term also ensures that the memory contents are stable. The third term in equation (4) is the regular parameter update that arises from the memory vectors μ_is being perceived as additional parameters of the NN by the NN learning algorithm. Here q_μ depends on the problem and c_w is a design constant. If the middle term reflecting new information is left out, the Memory Write equation becomes much more like the NN update equation, and thus we would expect the performance of the controller to not be different from the controller without memory. We illustrate this intuition in the simulation results later.

We emphasize here that, in the long term, the memory does not remember information from earlier learning episodes because its contents are gradually overwritten with information from later episodes. This follows from the Memory Write equation (4). We refer the reader to Fig. 5, which plots the trajectory of the Memory Read output, to illustrate this observation further. In this sense the memory is short term. It is a topic of further research to design a memory with an interface that can store and exploit relevant information from earlier learning episodes.

What is the rationale for the above definition of z_is? The weights z_is are defined in (5). Here we take the dot product of the write vector (which follows from the definition of query vector as given in (8)) and the respective memory vectors, as given in (5), and pass it through the softmax function to generate the weights z_is. It follows that the weight z_i is largest for the memory vector that is most similar to the write vector and smallest for the memory vector that is least similar. Thus z_i measures relevance by similarity.

We want the new information from the write vector to be considered as an update that is relevant to a memory vector j only when it is consistent with the information already stored in this location in terms of what it represents. Here the new write vector, which is given by (7), represents the *learned hidden layer output value for the current state*. Hence the interface should consider the new information as an update to that location j whose content is the last write vector that was equal to the learned hidden layer output value for the same state. This new write vector is likely to be similar to this earlier write vector and so the memory vector at location j. Also, the weight z_i, as defined in (5), is determined by similarity of the write vector and the memory vector at location i. Consequently, the weight z_i is likely to be higher for this location j, i.e., when $i = j$. Hence z_is are effective in determining relevance as we wanted.

A.2. Memory Read: It is defined by (5) and we need to specify the *query vector* q. We propose to set the query vector to be the write vector itself:

$$q = h = \sigma(\hat{V}^T x + \hat{b}_v) \tag{8}$$

The key is defined to be the memory vector itself. Below we justify these choices. The addressing mechanism for this interface matches the query and key by their respective dot product. These dot products are then normalized using the

softmax function to produce a set weights z_i's such that $\sum z_i = 1$, as given in (5). The final output M_r is the weighted linear combination of the memory vectors with z_is as the weights (5). Note that the weights are the largest for those memory vectors that are most similar to the query.

What is the rationale for the above definitions of query and key vectors? We first note that the Memory Read M_r is used to modify the control input as in (6). Hence, the query vector should be a function of hidden layer output which is exactly as in (8). The key should be matchable with the query vector and also contain information about the corresponding memory vector. The memory vector itself satisfies this criterion. Hence we select the memory vector itself as the corresponding key.

We want the addressing mechanism to be able to retrieve relevant values for the current scenario. In this work (i) we assume that abrupt changes in function $f(x)$, specified in terms of the change in value for each x, are not large and (ii) the memory remembers information only from recent learning episodes. Hence the new scenario after an abrupt change is not very different from the scenario that the memory contents correspond to. In such a case, the information that is likely to be relevant is the memory vector that is similar to the query vector (which is the current hidden layer output). This is exactly the output of the Memory Read operation (5), suggesting that the addressing mechanism is effective in retrieving relevant values for the current scenario.

A.3. NN Output: The learning system (NN) *modifies its output* using the information M_r retrieved from the memory. For this memory interface, the NN output is modified by adding the output of the Memory Read to the output of the hidden layer as in (6). Below we give a suggestive explanation using an example on how this modification improves the context *inducing the learner to learn quickly*.

How does the modified NN output induce the learner to learn quickly? While a full analysis of this remains a topic for future research, here we provide a suggestive explanation using an example. Let's consider the scenario where the learning system had settled and remained in steady state for a long time after the initial learning phase. Assume that the NN weights had converged to their correct values, i. e., $\hat{W} = W, \hat{b}_w = b_w, \hat{b}_v = b_v$ and $\hat{V} = V$. We refer to the corresponding NN as the *first network*. Since this is a good approximation, the bound on the error, which is c, is small. Suppose that $f(.)$ changes as follows:

$$f \leftarrow c_1 f(.) + c_2 \text{ where } 1 + c > c_1 > 1 - c \tag{9}$$

The new f is approximately equal to a *second neural network* whose weights are given by $\hat{W} = c_1 W, \hat{b}_w = c_1 b_w + c_2, \hat{b}_v = b_v$ and $\hat{V} = V$. It is easy to show that the error for this approximation is bounded by $c + c^2$, and is small because c is small. Note that the hidden layers of the correct approximations are identical for the times before (first network) and after the abrupt change (second network).

Before the abrupt change the contents of the memory are the hidden layer values of the first (and so second) network. This follows from the discussion on Write operation and the fact that the Memory Write equation is reduced to the first two terms after the initial learning phase. This is so because q_μ becomes zero once the learning phase is over (refer section IV-B). Just after the abrupt change the Memory Read outputs a value that is approximately equal to the hidden layer output of the first (and so second) network. This follows from the discussion on Memory Read operation.

The modification of the NN output using this Memory Read appears to the learner as if the hidden layer is partially fixed at the hidden layer of the second network. In addition, the second network is a correct approximation of the new function and so is a valid point for the learning to converge to after the abrupt change. As a result, the learner is induced to update its outer layer weights in a direction that converges to the outer layer weights of the second network, further inducing the learning of the whole network to proceed in a direction towards the second network. This directed learning makes the learning quicker.

A.4. Learning Mechanism Principle: Motivated by the above discussion, we believe that, for our proposed learning controller that uses an external working memory, *learning is accelerated through an induced learning mechanism which facilitates quick convergence to a neural network that is a good approximation of the new function*. Interestingly, this is also the idea behind transfer learning where a pre-trained network on a different problem is reused for the new problem by fine tuning only the final layers while fixing the earlier layers.

B. Control Algorithm

First, we provide the complete set of equations for the general control architecture in Fig. 1 and then provide their specific forms for (i) the MRAC controller of a linear plant with matched uncertainty and (ii) the backstepping controller for strict feedback systems.

B.1. General Control and Update Law: The Memory Write equation, the Memory Read equation and the NN output are same as the equations (4), (5) and (6) respectively. The control input for NN adaptive control is a combination of base controller u_{bl}, which is problem specific, the NN output u_{ad} and a "robustifying term" v [6], [25]. The final control input is given by,

$$u = u_{bl} + u_{ad} + v, \tag{10}$$

The variable q_μ in (4) is problem specific and depends on the Lyapunov function (without the NN error term). The NN update law, which constitutes the learning algorithm for the proposed architecture, is the regular update law for a two layer NN [25],

$$\begin{bmatrix} \dot{\hat{W}} \\ \dot{\hat{b}}_w^T \end{bmatrix} = C_w \left(\hat{\sigma} - \hat{\sigma}' \left(\hat{V}^T x + \hat{b}_v \right) \right) q_\mu$$

$$\begin{bmatrix} \dot{\hat{V}} \\ \dot{\hat{b}}_v^T \end{bmatrix} = C_v \begin{bmatrix} x \\ 1 \end{bmatrix} q_\mu \begin{bmatrix} \hat{W} \\ \hat{b}_w^T \end{bmatrix}^T \hat{\sigma}' \tag{11}$$

B.2. MRAC Controller: Here we specify the control equa-

tions of the MRAC controller for the system in (1). The control input and the update laws are same as (10) and (11). If the base controller input u_{bl} is the standard LQR controller and the lyapunov function is $e^T Pe$, then the vector $q_\mu = e^T PB$, where $e = x - x_{ref}$, P is the matrix solution to the lyapunov equation $A_{ref}^T P + PA_{ref} = -Q$, and Q is a positive definite matrix (Refer [26]). The robustifying term is given by $v = -k_z \left(\|\hat{W}\|_F + \|\hat{V}\|_F + \|\mu\|_F \right) \|e\|_2$.

B.3. Backstepping Controller for Strict Feedback Systems: A strict feedback system is given by the set of equations as given in (12). The function $g(x) > 0 \; \forall \; x$. The problem setup is that y has to track the desired trajectory x_{1d}. In addition the system is either partially or completely unknown if either $f(x)$ is unknown or both $f(x)$ and $g(x)$ are unknown. The control can be designed via the backstepping method [8], [27]. The backstepping method entails defining a sequence of auxiliary control inputs, one for each state x_i, in a recursive manner. For lack of space we don't discuss the derivation of the backstepping controller.

$$\begin{aligned} \dot{x}_1 &= f_1(x_1) + g_1(x_1)x_2 \\ &\vdots \\ \dot{x}_n &= f_n(x_n) + g_n(x_n)u, \; y = x_1 \end{aligned} \tag{12}$$

Here we provide the architecture for robust NN backstepping controller of partially known systems. This controller uses a separate NN to approximate each of the unknown function $f_i(.)$. We use subscript i to denote the weights of each neural network. In the extended architecture a separate memory is augmented to each neural network. Two subscripts are used to denote the memory and its specific vector respectively. The auxiliary control inputs for $i = 1$ is just the desired trajectory x_{1d}. For $1 \leq i \leq n-1$ the auxiliary control inputs $x_{i+1,d}$s are a combination of the base auxiliary controller, a NN output and a robustifying term,

$$\begin{aligned} x_{i+1,d} &= x_{i+1,bl} + x_{i+1,ad} + v_{i+1} \\ x_{i+1,bl} &= g_i^{-1} \left(\dot{x}_{i,d} - K_i e_i - g_{i-1}^T e_{i-1} \right) \\ x_{i+1,ad} &= g_i^{-1} \left(-\hat{W}_i^T \left(\sigma \left(\hat{V}_i^T x_i + \hat{b}_{i,v} \right) + M_{i,r} \right) - \hat{b}_{i,w} \right) \\ v_{i+1} &= -g_i^{-1} k_z (\|\hat{W}_i\|_F + \|\hat{V}_i\|_F + \|\mu_i\|_F) e_i \end{aligned} \tag{13}$$

where $e_i = x_i - x_{i,d}$. The control input u also has three terms but with few differences,

$$\begin{aligned} u &= u_{bl} + u_{ad} + v_{n+1}, \; u_{bl} = g_n^{-1} \left(\dot{x}_{n,d} - K_n e_n \right) \\ u_{ad} &= g_n^{-1} \left(-\hat{W}_n^T \left(\sigma \left(\hat{V}_n^T x_n + \hat{b}_{n,v} \right) + M_{n,r} \right) - \hat{b}_{n,w} \right) \\ v_{n+1} &= -g_n^{-1} k_z (\|\hat{W}_n\|_F + \|\hat{V}_n\|_F + \|\mu_n\|_F) e_n \end{aligned} \tag{14}$$

The NN update laws for the network i is the same as (11). The $q_{i,\mu}$ vector is e_i if the lyapunov function for each auxiliary control is e_i^2.

V. Discussion and Simulation Results

In this section we provide simulation results that illustrate the improvements in learning by inclusion of an external working memory. We illustrate this using examples for each of the controllers in section IV-B. In addition we also provide (i) comparison with response of the MANN controller when $c_w = 0$ in (4) (ii) comparison of performance of a NN which has the same number of parameters as the MANN controller (number for the latter includes the size of the memory) (iii) illustration of induced learning.

A. Flight Control Problem

In this sub-section we consider the longitudinal dynamics of a flight. The system model is a linear plant with a matched uncertainty. Denote the flight's angle of attack by α, pitch by q, the elevator control input by u. Then the system equations appended with an integrator, as in [26], is given by,

$$\begin{aligned} \begin{bmatrix} \dot{e}_I \\ \dot{\alpha} \\ \dot{q} \end{bmatrix} &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & \frac{Z_\alpha}{mU} & 1 + \frac{Z_q}{mU} \\ 0 & \frac{M_\alpha}{I_y} & \frac{M_q}{I_y} \end{pmatrix} \begin{bmatrix} e_I \\ \alpha \\ q \end{bmatrix} \\ &+ \begin{pmatrix} 0 \\ \frac{Z_\delta}{mU} \\ \frac{M_\delta}{I_y} \end{pmatrix} (u + f(x)) + \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} y_{cmd} \end{aligned} \tag{15}$$

where y_{cmd} is the command signal that the angle of attack has to track and $e_I = \int \alpha - y_{cmd}$. Here $y_{cmd} = 0.1$ rad. The system parameters are that of B-747 flight. The flight's mass is $m = 288773$ Kg, moment of inertia $I_y = 44877574$ Kgm^2. The flight is assumed to be at an altitude of $h = 6000$ m and travelling at a speed of $U = 274$ m/s (0.8 Mach). The base controller is a LQR controller whose cost matrices are given by $Q = I$ and $R = 1$. The values of the terms in the matrices for B-747 are given by,

$$\begin{aligned} &\frac{Z_\alpha}{mU} = -0.32, 1 + \frac{Z_q}{mU} = 0.86, \frac{M_\alpha}{I_y} = -0.93, \\ &\frac{M_q}{I_y} = -0.43, \frac{Z_\delta}{mU} = -0.02, \frac{M_\delta}{I_y} = -1.16 \end{aligned} \tag{16}$$

The memory interface parameters are set as $c_w = 3/4$, $k_z = 0$ and the NN learning rates are set as $C_w = C_v = 10$. The outer layer NN weights are initialized to value 0 and the hidden layer weights and the memory vectors are randomly initialized to a number between 0 and 1. In the two examples we consider here, the number of hidden layer neurons (N) is set as 4 and 5 respectively and the number of memory vectors (n_s) is set as 1. Number of hidden layers is chosen to be the smallest number such that the NN controller (without memory) does not show large oscillations. The uncertainty jumps in both the examples are set such that the uncertainty term is at most of the same order as the linear terms in the system equations. In example 1, $f(x)$ is given by,

$$\begin{aligned} &f(x) = C_f(t)x^2, \text{ where } C_f = 0.1 \text{ at } t = 0, \\ &C_f \to 50C_f \text{ at } t = 5 \text{ and } C_f \to 2C_f \text{ at } t = 25 \end{aligned} \tag{17}$$

In example 2, $f(x)$ is given by,

$$\begin{aligned} &f(x) = x^2 + 0.1C_f, \text{ where } C_f = 0.1 \text{ at } t = 0, \\ &C_f \to 10C_f \text{ at } t = 5 \text{ and } C_f \to 2C_f \text{ at } t = 25 \end{aligned} \tag{18}$$

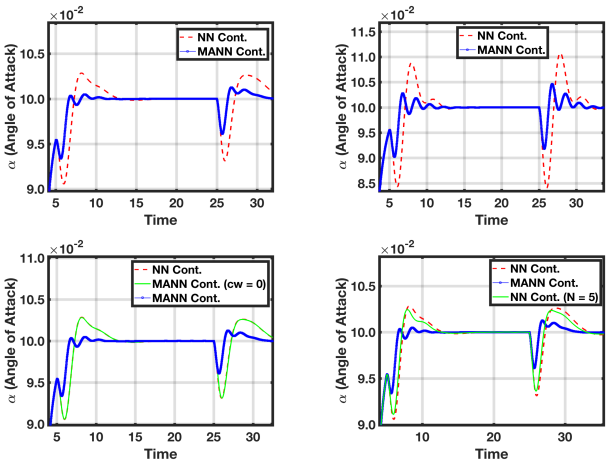

Fig. 3. Comparison of MRAC flight controllers with and without Memory. Left above: angle of attack response α (example 1), right above: angle of attack response α (example 2), left below: comparison with MANN controller without the first update term (example 1), right below: comparison with $N = 5$ NN controller (example 1)

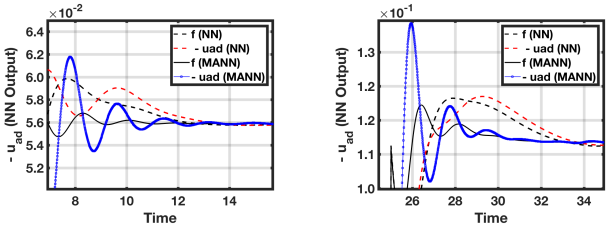

Fig. 4. Illustration of *induced learning* in the flight control problem. Left: - NN output for both controllers at first abrupt change (example 1), right: - NN output for both controllers at second abrupt change (example 1)

B. Control of Strict Feedback System

For illustration we consider the following partially known strict feedback system: $n = 2$, $f_1 = -0.1x_2 + 0.1x_2^2$, $f_2 = -0.1x_2 + 0.1x_2^2$, $g_1 = 1 + 0.1x_1^2$, $g_2 = 1 + 0.1x_2^2$. The controller parameters are set as: $c_w = 3/4$, $K_i = 10\ \forall i$, $k_z = 10$, $C_w = C_v = 10$. The number of hidden layer neurons and the number of memory vectors are set as 3 and 1 respectively. The desired trajectory $x_{1d} = 0.1$. The outer layer NN weights are initialized to 0 and the hidden layer and memory vectors are randomly initialized to a number between 0 and 1. In this example, $f_1 = x_1^2 + 0.001$ at $t = 0$ and $f_1 \to f_1 + 0.25\sin(10x_1) - 0.001$ at $t = 5$; $f_2 = x_2^2 + 0.001$ at $t = 0$ and $f_2 \to f_2 - 0.001$ at $t = 5$.

C. Discussion

From the top two plots of Fig. 3 and the left plot of Fig. 6, it follows that augmenting the NN controller with a working memory improves significantly the system's response to abrupt changes. Improvements are seen in both peak reduction and settling time. And this is observed to hold for the varied scenarios considered in these simulations. Tables I and II provide precise values for the two performance metrics (i) peak deviation (ii) settling time for 10% error. These metrics clearly show that the improvements obtained, by the inclusion of a working memory, are significant.

TABLE I

FLIGHT CONTROL, PEAK DEVIATION, $\max|\alpha - y_{\text{CMD}}|$

Example	1	2
NN cont. (I)	0.54^o	0.89^o
NN cont. (II)	0.51^o ($N = 5$)	0.7^o ($N = 6$)
MANN Cont.	0.38^o	0.47^o
Reduction (from (II))	25.5%	32.6%

TABLE II

FLIGHT CONTROL, SETTLING TIME (10 % ERROR)

Example	1	2
NN cont.	6.61 s	6.55 s
NN cont.	5.91 s ($N = 5$)	5.43 s ($N = 6$)
MANN Cont.	3.45 s	4.1 s

We also compare the MANN controller with a NN controller that has atleast the same number of parameters as the MANN controller. For the flight control problem, this NN controller has $N = 5$ and $N = 6$ number of hidden layer neurons in examples 1 and 2 respectively. In the example for strict feedback system, this NN controller has $N = 4$ number of hidden layer neurons. The simulations reveal that this NN controller, inspite of having the same number of parameters as the MANN controller, is still weaker in its response to abrupt changes. This is illustrated in the bottom right plot of Fig. 3 and the right plot of Fig. 6 and clearly suggested by the performance measures in Tables I and II.

The bottom left plot of Fig. 3 provides comparison of the response of the MANN controller without the first update term. It is clear that the response without this term is only as good as the NN controller without the memory justifying a previous statement that we made. The effect of induced learning is illustrated in Fig. 4. From the plots in Fig. 4 it is evident that the external working memory induces the NN network to converge to a good approximation in quick time.

In Fig. 5 we show the first two components of the hidden layer output of the NN controller and the MANN controller and M_r (the output of Memory Read (5)). In these plots, the Memory Read output M_r is scaled by $1/c_w$ to account for the same factor in the first update term (4). The example considered in the plots is similar to that of the example in section IV-A. From Fig. 5 it is evident that the hidden layer output of the MANN controller converges nearer to the value before the abrupt change. By contrast, it converges to a very different value for the NN controller without memory. This suggests that the memory is inducing the NN to converge to a network with very similar hidden layer weights, which is similar to the outcome of the example discussed in section IV-A. Consequently, this result also serves as evidence for the induced learning mechanism.

The plots also reveal that the contents are updated after every learning phase that follows an abrupt change. This retains the effect of induced learning as observed in the controller's performance after subsequent changes. Finally

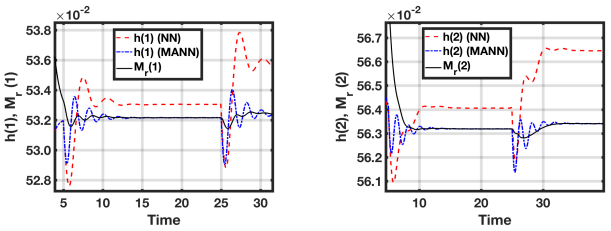

Fig. 5. Hidden layer output, h. Left: first component of hidden layer output (example 2), right: second component of hidden layer output (example 2)

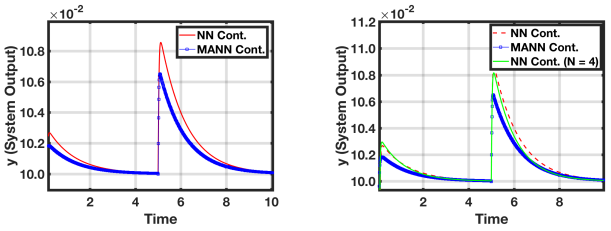

Fig. 6. Comparison of backstepping controllers with and without memory. Left: system output y, right: comparison with $N = 4$ NN controller

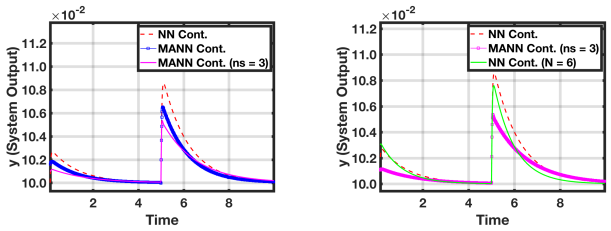

Fig. 7. Backstepping controller with $n_s = 3$ memory states. Left: system output y, right: comparison with $N = 6$ NN controller

we address the aspect of having more than one memory state. In Fig. 7 we provide the response for the backstepping controller with more than one memory state. It clearly reveals that by having more than one memory state the learning performance can be improved.

VI. Conclusion

We proposed a novel control architecture for adaptive control of continuous time systems that is inspired from neuroscience. The proposed architecture augments an external working memory to the neural network that compensates the unknown nonlinear function in the system dynamics. We provided a specific memory interface for this architecture and discussed how this design improves the speed of learning by a mechanism called *induced learning*. Finally, we provided simulation results for two class of controllers (i) a NN MRAC controller for linear systems with matched uncertainty and (ii) backstepping NN controller for strict feedback systems. The simulations and the performance metrics clearly established that the controllers with memory augmentation provide significant improvements in learning and performance over their counterparts without memory.

References

[1] E. Tulving, "How many memory systems are there?" *American psychologist*, vol. 40, no. 4, p. 385, 1985.

[2] H. L. Roediger, F. M. Zaromb, and W. Lin, "A typology of memory terms," 2017.

[3] S. J. Gershman and N. D. Daw, "Reinforcement learning and episodic memory in humans and animals: an integrative framework," *Annual review of psychology*, vol. 68, pp. 101–128, 2017.

[4] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on neural networks*, vol. 1, no. 1, pp. 4–27, 1990.

[5] A. Yeşildirek and F. L. Lewis, "Feedback linearization using neural networks," *Automatica*, vol. 31, no. 11, pp. 1659–1664, 1995.

[6] F. L. Lewis, A. Yesildirek, and K. Liu, "Multilayer neural-net robot controller with guaranteed tracking performance," *IEEE Transactions on Neural Networks*, vol. 7, no. 2, pp. 388–399, 1996.

[7] K. S. Narendra and S. Mukhopadhyay, "Adaptive control using neural networks and approximate models," *IEEE Transactions on neural networks*, vol. 8, no. 3, pp. 475–485, 1997.

[8] C. Kwan and F. L. Lewis, "Robust backstepping control of nonlinear systems using neural networks," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 30, no. 6, pp. 753–766, 2000.

[9] L. Chen and K. S. Narendra, "Nonlinear adaptive control using neural networks and multiple models," *Automatica*, vol. 37, no. 8, pp. 1245–1255, 2001.

[10] T. Yucelen, G. De La Torre, and E. N. Johnson, "Improving transient performance of adaptive control architectures using frequency-limited system error dynamics," *International Journal of Control*, vol. 87, no. 11, pp. 2383–2397, 2014.

[11] V. Stepanyan and K. Krishnakumar, "Adaptive control with reference model modification," *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 4, pp. 1370–1374, 2012.

[12] T. E. Gibson, A. M. Annaswamy, and E. Lavretsky, "Adaptive systems with closed-loop reference models: Stability, robustness and transient performance," *arXiv preprint arXiv:1201.4897*, 2012.

[13] H. Whitaker, J. Yamron, and A. Kezer, "Design of model reference control systems for aircraft," *Cambridge, MA: Instrumentation Laboratory, Massachusetts Institute of Technology*, 1958.

[14] E. Tulving *et al.*, "Episodic and semantic memory," *Organization of memory*, vol. 1, pp. 381–403, 1972.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[16] M. Lengyel and P. Dayan, "Hippocampal contributions to control: the third way," in *Advances in neural information processing systems*, 2008, pp. 889–896.

[17] A. D. Baddeley, "Is working memory still working?" *European psychologist*, vol. 7, no. 2, p. 85, 2002.

[18] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[19] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International conference on machine learning*, 2016, pp. 1842–1850.

[20] E. Parisotto and R. Salakhutdinov, "Neural map: Structured memory for deep reinforcement learning," *arXiv preprint arXiv:1702.08360*, 2017.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *International Conference on Artificial Neural Networks*. Springer, 2001, pp. 87–94.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[24] J. Guerguiev, T. P. Lillicrap, and B. A. Richards, "Towards deep learning with segregated dendrites," *Elife*, vol. 6, p. e22901, 2017.

[25] F. Lewis, S. Jagannathan, and A. Yesildirak, *Neural network control of robot manipulators and non-linear systems*. CRC Press, 1998.

[26] S. A. Nivison and P. Khargonekar, "Improving long-term learning of model reference adaptive controllers for flight applications: A sparse neural network approach," in *AIAA Guidance, Navigation, and Control Conference*, 2017, p. 1249.

[27] M. Krstic, I. Kanellakopoulos, P. V. Kokotovic *et al.*, *Nonlinear and adaptive control design*. Wiley New York, 1995, vol. 222.