Memory Capacity of Neural Turing Machines with Matrix Representation

Animesh Renanse *
Department of Electronics & Electrical Engineering
Indian Institute of Technology Guwahati
Assam, India
renanse@iitg.ac.in

Rohitash Chandra *
School of Mathematics and Statistics
University of New South Wales
Sydney, NSW 2052, Australia
rohitash.chandra@unsw.edu.au

Alok Sharma
Laboratory for Medical Science Mathematics
RIKEN Center for Integrative Medical Sciences
Yokohama, Japan
alokanand.sharma@riken.jp

*Equal contributions

Editor:

Abstract

It is well known that recurrent neural networks (RNNs) faced limitations in learning long-term dependencies that have been addressed by memory structures in long short-term memory (LSTM) networks. Matrix neural networks feature matrix representation which inherently preserves the spatial structure of data and has the potential to provide better memory structures when compared to canonical neural networks that use vector representation. Neural Turing machines (NTMs) are novel RNNs that implement notion of programmable computers with neural network controllers to feature algorithms that have copying, sorting, and associative recall tasks. In this paper, we study augmentation of memory capacity with matrix representation of RNNs and NTMs (MatNTMs). We investigate if matrix representation has a better memory capacity than the vector representations in conventional neural networks. We use a probabilistic model of the memory capacity using Fisher information and investigate how the memory capacity for matrix representation networks are limited under various constraints, and in general, without any constraints. In the case of memory capacity without any constraints, we found that the upper bound on memory capacity to be $N^2$ for an $N \times N$ state matrix. The results from our experiments using synthetic algorithmic tasks show that MatNTMs have a better learning capacity when compared to its counterparts.

Keywords: Recurrent Neural Networks, Representation Learning, Memory Networks, Memory Capacity.
1. Introduction

Several successful attempts have been made to explain learning and knowledge representation from perspective of kernel methods (Jacot et al., 2018) to dynamical systems viewpoint of recurrent neural networks (RNNs) (Omlin and Giles, 1992; Omlin et al., 1998; Chang et al., 2019). A crucial element in knowledge representation studies of RNNs is that they deal only with finite dimensional vector representations. Thus implementing such neural networks in any architecture would eventually lead to flattening of the input data along any mode, given the input is multi-modal to begin with. It can thus be argued that such flattening layer can cause loss of structural and spatial information in the input, especially in data arising in natural settings like tri-modal images/sequence of images in longitudinal analysis and higher modal datasets arising in the study of human behavior (Busso et al., 2008; Ringeval et al., 2013).

The introduction of even richer inductive biases (Mitchell, 1980) in the past few years can be seen through recent deep learning architectures such as convolutional neural networks (CNNs) (Krizhevsky et al., 2012), graph structured representations (Teney et al., 2016; Kipf et al., 2019), and more recently proposed hierarchy of constituents (Shen et al., 2018). These methods try to preserve the spatial information by providing fewer constraints on the structure of the representation that the model learns in comparison to a traditional neural network (Battaglia et al., 2018). A parallel branch, which provides more generalized representations, which can be introduced by learning higher order tensors ($\geq 2$) for multi-modal data, preserves the number of dimensions and the modes at the output (Bai et al., 2017; Su et al., 2018; Nguyen et al., 2015). This avoids the potential loss of the spatial coherence (Bengio et al., 2013) in the input data which can be incurred due to flattening operation of the higher order tensors and instead process the multi-modal higher order tensor input data as it is.

A particularly simple use of learning higher order tensors as representations is in the field of symmetric positive definite (SPD) matrix (an order 2 tensor) learning, which arises largely in covariance estimation tasks like heteroscedastic multivariate regression (Muller et al., 1987). In this task, one wishes to learn fundamentally a matrix representation from the input data, thus several extensions of conventional neural networks with vector representations have been introduced to accept matrix inputs as well, like learning the SPD representations throughout the layers via Mercer’s kernel (Taghia et al., 2019) to incorporating Riemannian structure of the SPD matrices directly into the layers itself (Huang and Gool, 2016). The matrix representation in such proposals are generated through the bilinear map of the form,

\[ Y = \sigma(U^T X V + B) \]  

(1)

where $U$, $V$ and $B$ are the parameters to be learned of appropriate shapes (Gao et al., 2016). Do et al. (2017), with the help of aforementioned bilinear map, develops the notion of Recurrent Matrix Networks. One of the other benefits argued by the authors of both the studies is the reduction of number of trainable parameters of the model, as the total parameters depends now linearly on any of the input dimensions. This is in contrast to vector-based conventional neural networks where the number of parameters grows quadratically with the dimensions of input vector. Matrix neural networks have been successfully used in the past for seemingly difficult real life tasks like cyclone track prediction (Zhang et al., 2018) and
in high frequency time series analysis (Tran et al., 2019), where the preservation of spatial
data amongst the modes of input proved to be beneficiary as these networks gave better
accuracy than their vector counterparts.

Neural Turing machines (NTMs) are RNN models that combine combine the fuzzy pattern
matching capabilities of neural networks with the notion of programmable computers.
NTMs feature a neural network controller coupled to external memory that interacts with
attention mechanisms. NTMs that feature long short-term memory (LSTM) (Hochreiter
and Schmidhuber, 1997) for network controller can infer simple algorithms that have copying,
sorting, and associative recall tasks (Graves et al., 2014). With their potential benefits,
it’s natural to investigate properties of the matrix representation as generated by the bi-
linear mapping in Eq. 1. Out of the various possible ways in which this can be checked
(such as convergence guarantee and training behaviour etc.), we particularly focus on the
asymptotic memory capacity of the matrix representation by introducing the dynamical
systems viewpoint of generic matrix RNNs (Do et al., 2017).

In this paper, we study memory capacity using matrix representation as used by matrix
RNNs and extend the notion to NTMs. We investigate if the matrix (second order tensor)
representation of data has a better memory capacity than the vector representation in
conventional neural networks. Intuitively, one might argue that such matrix representations
might always be better than vector representations in terms of memory capacity. However,
we show theoretically that it isn’t the case under some constraints, even though in general,
the capacity might be better than vector representations with a bound which is analogous to
the one found for vector representations (Ganguli et al., 2008). Hence, it is vital to discover
ways towards increasing the memory capacity, especially given the memory architectures
that have been proposed in the past (Graves et al., 2014, 2016; Santoro et al., 2016).

In order to provide theoretical evaluation, we take into consideration a generic memory
augmented RNN which simply stores the finite amount of past states. With this archi-
tecture, we thus calculate and simulate it’s capacity to show that it is greater than the
capacity of a simple matrix recurrent architecture, hence showing the increase in the notion
of mathematical memory capacity introduced by adding an extremely simple memory archi-
tecture. However, for more practical purposes, we extend the idea of memory networks
(Weston et al., 2014) to also include the second order tensors, thus introducing the matrix
representation stored - neural memory. We introduce matrix representation stored neural
memory for matrix NTM (MatNTM). We report the results of simulation of the governing
equations of memory capacity of matrix RNNs both with and without external memory
and also shows results of synthetic algorithmic experiments using MatNTMs.

The structure of the paper is given as follows. In Section 2, we briefly introduce and
discuss definitions for memory capacity in neural networks. Section 3 introduces the formal
notion of memory capacity for a matrix recurrent network and presents various bounds on
it under differing constraints on the parameters. Section 4 presents study on the effect
of adding an external memory to evaluate memory capacity. Section 6 presents results
of simulations of the governing equations of memory capacity of matrix recurrent networks
both with and without external memory and Section 7 shows further results of two synthetic
algorithmic experiments. Finally, we conclude the paper with discussion of further research
in Section 8.
2. Memory Capacity in Neural Networks

The broader work on the memory capacity and capacity in general of vector neural networks provides multiple such notions. For example, Baldi and Vershynin (2018) defines $\log_2 |S|$ as the memory capacity of the architecture where $S$ is the set of all functions that it can realize. It has been shown that for a recurrent neural network with $N$ total neurons memory capacity is proportional to $N^3$ (Baldi and Vershynin, 2018). Furthermore, the same definition is used to derive that the capacity of an usual $L$ layered neural network with only threshold activation is upper bounded by a cubic polynomial in size of each layer with bottleneck layers limiting this upper bound (Baldi and Vershynin, 2019). However, the definition of capacity in this case from a memory standpoint is only \textit{expressive} in the sense that it does not refer to any capability of the network to remember the past inputs. A stronger argument presented by the same authors for the definition of memory capacity is that of the existence of a realizable function such that the given architecture can perfectly remember past inputs. That is, the largest value of $K$ such that there exists a function $F$ in the space of all functions realizable by the given architecture $S$, so that for a given set of data $x_i$ and label $y_i$, $\{(x_1,y_1),\ldots,(x_K,y_K)\}$, it’s true that

$$F(x_i) = y_i \text{ for all } i = 1,\ldots,K.$$ 

This definition has been used by Vershynin (2020) to develop memory capacity of multilayered neural networks with threshold and rectifier linear unit (ReLU) activation functions. It was shown that the memory capacity is lower bounded by the number of total weights.

This definition allows deeper mathematical analysis as visible by the remarkable results in Vershynin (2020). However, this definition is not easily portable to higher order representations and their analysis. We gather that most of the analysis done in Vershynin (2020) depends on the vector representations retrieved from the affine transform $Wx + b$, whereas this paper focuses on the bilinear transform $U^TXV + B$ to generate matrix representations of input data. Apart from this, the set of functions realized by the bilinear transforms are fundamentally different from affine transformation. The extension of theory by Baldi and Vershynin (2019); Vershynin (2020) would at the very least need extension of the probabilistic view of the realizable function $F$ to their matrix variate analogues, which further makes the situation unmanageable.

Some of the earlier works on neural networks from a dynamical systems viewpoint gives us a possible alternative definition of the memory capacity purely in terms of a statistic. White et al. (2004); Ganguli et al. (2008) studied discrete time recurrent system

$$x(n) = Wx(n-1) + vs(n) + z(n)$$

where $W$ is the recurrent connectivity matrix and the time dependent scalar signal $s(n)$ is passed through this system via feedforward connection $v$ which contains an additive noise content $z(n)$. Additionally, we can consider an output layer which transforms the state $x(n)$ of length $N$ to an output vector $y(n)$ of length $M$ such that $y_i(n) = x(n)^T u_i$ for $i = 1,\ldots,M$. The objective of the system is to have output $y_j(n)$ equal to the past signal observed $s(n-j)$. With this premise in-place, White et al. (2004) defines the memory trace $m(k)$ as,

$$m(k) = \mathbb{E} [s(n-k)y_k(n)]$$
where expectation is over time. The (asymptotic) memory capacity can be now defined naturally as the following extension of Equation 3.

\[
m_{\text{tot}} = \sum_{k=0}^{\infty} m(k)
\]

Using this definition, we are only shifting our focus on a particular task of past recall, which is orthogonal to our aim of determining the memory capacity of the system itself, invariant of the task it’s performing. To achieve this aim, we clearly need to focus more on the memory capacity induced by the state transition of the system. Ganguli et al. (2008) extended the work to more general notion of memory capacity, with the trace of Fisher Information Matrix (FIM, or equivalently the Fisher Memory Matrix, FMM, depending upon the context of interpretation) of the state \( x(n) \) with the vector of temporal scalar inputs \( s \) as the parameters of the conditional \( p(x(n)|s) \).

Hence, Fisher information matrix of \( p(x(n)|s) \) measures the memory capacity by the capability of past inputs \( s(i) \) for \( i \leq n \) to perturb the current state \( x(n) \), which in turn shows the reluctance or permeability of the state \( x(n) \) to change under a new \( s \). Note that such a measure does not depend explicitly on the structure of the state \( x(n) \) as that information is portrayed by \( p(x(n)|s) \). Thus, as long as we are able to determine the conditional for a given type of state and discrete dynamics (even for matrix representations shown in Equation 5), we can use this definition for obtaining a measure of memory capacity.

However, due to it’s reliance on statistical information about the sensitivity of state rather than explicit measure of the maximum size of the input data which the architecture can map correctly, makes this definition weaker than that of Vershynin (2020) for artificial neural networks. However, due to it’s invariance on the state structure explicitly, and dependence only on conditional distribution of current state given past inputs, it becomes ideal for analysis on a much diverse set of representations in neural networks.

3. Memory Trace in Matrix Representation of Neural Networks

In this section, we define the recurrent dynamical system with matrix representations and then derive it’s Fisher Memory Curve. We then define and derive the memory capacity of matrix networks under certain constraints which further motivates the need of external memory augmentation.

We first study the following dynamical system

\[
X(n) = f(U^T X(n - 1) V + W s(n) + Z(n))
\]

where \( f(.) \) is a pointwise non-linearity, \( s(n) \) is a time-dependent scalar signal, \( X(t) \in \mathbb{R}^{N \times N} \) is the recurrent state of the network, \( W \in \mathbb{R}^{N \times N} \) is the feedforward connection for the scalar signal to enter the state, \( U^T \) and \( V \in \mathbb{R}^{N \times N} \) are the connectivity matrices which transforms current state to the next state and \( Z \sim \mathcal{M}_{N \times N}(0, \varepsilon_1 I, \varepsilon_2 I) \) is the additive white Matrix Gaussian noise with row-wise covariance being \( \varepsilon_1 I \) and column-wise covariance being \( \varepsilon_2 I \) (Gupta and Nagar, 1999).

A recurrent matrix representation system similar to Equation 5 has been developed recently by Do et al. (2017) for dealing with temporal matrix sequences and successfully
used it for sequence-to-sequence learning tasks (Sutskever et al., 2014). In particular, the authors used the following bilinear transformation,

$$H_t = f(U_1^TX_1V_1 + U_2^TH_{t-1}V_2 + B)$$  \hspace{1cm} (6)

where it's clear on comparison with Eq. 5 the main difference that Eq. 5 deals with scalar input $s(n)$ whereas Eq. 6 with more general matrix inputs $X_t$. However, for the ease of analysis of matrix representations, we continue with Eq. 5.

### 3.1 The Fisher Memory Curve

We adopt similar framework as introduced in Ganguli et al. (2008) to develop the performance measure to formalize the efficiency of recurrent network state matrix $X$ to encode the past input signals. Since the information contained in the input signal $s(t)$ is transferred to the recurrent state via the connection matrix $W$, therefore, after large enough $t$, the past signal history induces a conditional distribution on network state $X$; denoted as $p(X(n) | s)$, where $s$ is the vector of all past input signals, with $s_k = s(n-k)$.

We wish to focus on the amount of information that is contained in $X(n)$ about an input signal which appeared $k$ time steps in the past, i.e. $s_k$. We would thus need to know how much the conditional distribution on network state ($p(X(n) | s)$) changes with change in $s$. We note that the Kullback–Leibler (KL) divergence which is also known as relative entropy is a measure of how one probability distribution differs from another probability distribution which is useful in comparing statistical models (Kullback and Leibler, 1951). For a small change $\delta s$, the KL-divergence between $p(X(n) | s)$ and $p(X(n) | s + \delta s)$ can be shown via second-order Taylor series expansion to be approximately\(^1\) equal to $\frac{1}{2}\delta s^T J(s) \delta s$; where $J(s)$ is the Fisher Memory Matrix whose element $J_{i,j}(s)$ identifies the sensitivity of $p(X(n) | s)$ to interference between $s_i$ and $s_j$ which are the input signals appearing $i$ and $j$ timesteps in the past respectively and is given as

$$J_{i,j}(s) = \mathbb{E}_{p(X(n) | s)} \left[ -\frac{\partial^2}{\partial s_i \partial s_j} \log p(X(n) | s) \right]$$  \hspace{1cm} (7)

which is parameterized by all the past inputs $s$. It’s trivial to see now that the diagonal elements of Fisher Memory Matrix (FMM) $J_{i,i}$ represents the Fisher information that the current state $X(n)$ contains about the input signal that appeared $i$ time steps in the past, i.e. $s_i$. Hence, $J_{i,i}$ can be considered as the curve denoting the temporal decay of past information provided to the current state by $s_i$ $i$ timesteps into the past, hence the name Fisher Memory Curve (FMC). Deriving the FMC for Equation 5 will thus provide us with a quantitative measure for the memory decay (and thus, the total capacity) in it’s dynamics. We first state and prove the FMM for the dynamical system defined by Equation 5 as the theorem which follows and then use it to derive it’s FMC and then the consequent bounds on memory capacity under various assumptions.

\(^{1}\) Note that this approximation becomes an equality if the mean $M(s) = \mathbb{E}[X(n) | s]$ is only linearly dependent on $s$, as is the case in 9 (Martens, 2020).
Theorem 1. (FMM for system in Equation 5) Consider the dynamical system defined by 5, the Fisher Memory Matrix defined in 7 for such a system is

\[
J_{i,j}(s) = \text{Tr} \left( \Sigma^{-1} \frac{\partial M(s)}{\partial s_i} \Psi^{-1} \frac{\partial M(s)}{\partial s_j} \right) \tag{8}
\]

where,

\[
M(s) = \sum_{k=0}^{\infty} U^k W s_k V^k
\]

\[
\Psi = \varepsilon_1 \sum_{k=0}^{\infty} U^k U^T
\]

\[
\Sigma = \varepsilon_2 \sum_{k=0}^{\infty} V^k V^T
\tag{9}
\]

Corollary 1. (FMC for system in Equation 5) The Fisher Memory Curve for the dynamical system 5 can hence be shown to be equal to the following,

\[
J(i) = J_{i,i} = \text{Tr} \left( \Sigma^{-1} V_i^T W^T U_i \Psi^{-1} U_i^T W V_i \right) \tag{10}
\]

For simplicity, we focus only on the linear dynamics, i.e. \( f(X) = X \). However, we later argue that saturating non-linearities can have deteriorating effect on the capacity as given later in Definition 1. This assumption which focuses on linear dynamics simplifies the analysis by a big margin as we will see in later sections. With this assumption of linear dynamics the canonical solution of the linear form of Equation 5 can be derived easily, and the final result is stated below:

\[
X(n) = \sum_{k=0}^{\infty} U^k W s_k V^k + \sum_{k=0}^{\infty} U^k Z(n-k) V^k \tag{11}
\]

Since the noise is an additive Gaussian matrix \( Z \), this implies that \( p(X(n)|s) \) will also be a Gaussian distribution; though with different mean and covariance matrices, one for rows as random vectors and another for columns. Since the noise entering the system at different times are also independent, we can write the mean matrix \( M \), the row-wise covariance matrix \( \Psi \) and the column-wise covariance matrix \( \Sigma \) for \( X(n) \) as shown in Equation 9. Note that the mean matrix \( M \) is parameterized linearly by input signal \( s \) while the covariance matrices are independent of \( s \).

We have seen in Section 3.1 that the KL-divergence can be approximated via quadratic Taylor series approximation. Calculating the KL-divergence of \( p(X(n)|s) \) will thus allow us to write the Fisher Memory Matrix in terms of \( M, \Psi \) and \( \Sigma \). Hence, we now state the form of KL-divergence between two matrix Gaussian distributions as the following lemma,
Lemma 1. (\(D_{\text{KL}}\) for \(\mathcal{MN}\)) The KL-divergence between \(p(X_1)\) and \(p(X_2)\) where \(X_1 \sim \mathcal{MN}_{n \times p}(M_1, \Sigma_1, \Psi_1)\) and \(X_2 \sim \mathcal{MN}_{n \times p}(M_2, \Sigma_2, \Psi_2)\) is given by

\[
D_{\text{KL}}(p(X_1) \| p(X_2)) = \frac{1}{2} \left[ \log \frac{\Sigma_2^{|p|} \Psi_2^n}{\Sigma_1^{|p|} \Psi_1^n} - np + \text{Tr} \left( \Psi_2^{-1} \Sigma_1 \right) \right. \\
+ \left. \text{Tr} \left( (M_2 - M_1)^T \Psi_2^{-1} (M_2 - M_1) \right) \right]
\]

where \(M_1, \Psi_1, \Sigma_1\) and \(M_2, \Psi_2, \Sigma_2\) are the mean and covariance matrices for \(X_1\) and \(X_2\), respectively.

We provide proof for the above lemma in Appendix A. Notice that from Equation 9, the independence of \(\Psi\) and \(\Sigma\) from signal history \(s_k\), thus the required \(D_{\text{KL}}(p(X(n) | s^1) \| p(X(n) | s^2))\) for different histories \(s^1\) and \(s^2\) simplifies to:

\[
D_{\text{KL}}(p(X(n) | s^1) \| p(X(n) | s^2)) = \frac{1}{2} \left[ \text{Tr} \left( (M_2 - M_1)^T \Psi^{-1} (M_2 - M_1) \right) \right]
\]

(12)

Thus, for the small change in input signal \(s_1 = s\) by \(\delta s\) (hence \(s_2 = s + \delta s\)), the 12 would be approximately equal to \(\frac{1}{2} \delta s^T J(s) \delta s\) as discussed earlier which would directly yield us the FMM as defined in Equation 8, hence proving Theorem 1.

To obtain an even simpler version of the FMM 8, one can further assume that the type of dependency of the mean \(M(s)\) is linear with \(s\) as is the case in 9, which directly implies that Equation 8 is independent of \(s\), leading us to the following form:

\[
J_{i,j} = \text{Tr} \left( \Sigma^{-1} W^{iT} W^{T} U^{i} \Psi^{-1} U^{jT} W^{j} \right)
\]

(13)

One should note that the form in Equation 13 is not as trivial to analyze as the \(J_{i,j}\) has been in Ganguli et al. (2008) for linear dynamical system with vector representations. The FMM equivalent for Equation 13 for such (vector representation) networks takes a very simple form as

\[
J_{i,j} = v^T W^{iT} C_n^{-1} W^{j} v
\]

(14)

where \(v\) is the feedforward connection from input to state, \(W\) is the recurrent connectivity matrix and \(C_n\) is the covariance matrix for the state given by \(C_n = \sum_{k=0}^{\infty} W^k W^{kT}\).

Extensive analysis is possible with Equation 14 as done in Ganguli et al. (2008) and further extension of the theory to other neural architectures like Echo State Networks (Tino, 2017). However, there’s been little to no work in other domains of representations and their capabilities from a memory standpoint. The break in symmetry of Equation 13 is clear in comparison to Equation 14, which is the main cause of complicated interaction terms between the recurrent and the feedforward connections which in-turn decreases the capacity as we shall see in the following sections.

We are mostly concerned about the information retained in \(p(X(n) | s)\) about a signal \(s_k\) entering the network \(k\) timesteps into the past. Considering Fisher memory matrix 13, this is expressed by the diagonal elements of \(J\) which signifies the Fisher information that the state \(X(n)\) retains about a pulse entering the network \(k\) time steps in the past. Thus, using this concept, one can define the memory decay of previous inputs by considering the set \(\{J(i) = J_{i,i} | 0 \leq i < \infty\}\) of all diagonal elements of \(J\), hence the FMC as described in Equation 10.
3.2 The Memory Capacity

The FMC shown in Equation 10 identifies the decay of information in the state $X(n)$ about the past signals. However, to capture the memory capacity of the system, we would need to measure the amount of information remaining in $X(n)$ for all the previous inputs as this will tell us the exact amount of Fisher information that the system has encoded about all the prior inputs. This can be represented by summing over all $i$ in Equation 10, thus yielding us the following definition of the memory capacity for any dynamical system,

\[ J_{\text{tot}} = \sum_{i=0}^{\infty} J(i) \]  

With this definition, the memory capacity of the matrix representation networks is,

\[ J_{\text{tot}} = \sum_{i=0}^{\infty} \text{Tr} \left( \Sigma^{-1} V^T W^T U^i \Psi^{-1} U^{iT} W V^i \right) \]  

This definition of memory capacity is general, however doesn’t give much insights on to the limit or bounds on the capacity directly. Motivated from the present study of vector representation on the similar assumptions as in Ganguli et al. (2008), we now discuss two cases. Firstly, we consider case when the recurrent connectivity matrices are assumed to be normal and secondly the case when they are not. We discuss the latter with a reformed view of Equation 10.

3.2.1 Capacity of Normal Networks

The form of $J(i)$ in Equation 10 doesn’t allow much room for direct analysis, thus, we make certain assumptions to dissect it further. One important assumption which we will continue to deal with is that of normal recurrent connectivity i.e. matrices $U^T$ and $V$ are assumed to be normal, that is they commute with their transpose. This assumption simplifies the FMC and provides insights on to the interaction between the connectivity matrices. Hence, we can write Equation 10 as the eigen-decomposition of $U$ and $V$

\[ J(i) = \frac{1}{\varepsilon_1 \varepsilon_2} \text{Tr} \left( (I - \Lambda_V^2) \Lambda_V^i B^H \Lambda_U^i (I - \Lambda_U^2) \Lambda_U^i B \Lambda_V^i \right) \]  

where $E_V$, $\Lambda_V$ and $E_U$, $\Lambda_U$ are the corresponding orthogonal eigenvector and diagonal eigenvalue matrices for $V$ and $U$ respectively and $B = E_U^H W E_V$. The form in Equation 17 can be achieved by substituting Equation 9 in Equation 10 and taking all matrices inside the inverse introduced by $\Sigma$ and $\Psi$.

Deriving the capacity $J_{\text{tot}}$ is straightforward from Equation 17, the calculations of which is done in Appendix E. We here state the final form of $J_{\text{tot}}$ as the following theorem,

**Theorem 2.** (Capacity of Normal Matrix Networks) Given the dynamical system 5 and
that both the connection matrices $U$ and $V$ are normal, the memory capacity of the system is given as the following.

$$J_{\text{tot}} = \frac{1}{\varepsilon_1 \varepsilon_2} \sum_{j=1}^{N} \sum_{k=1}^{N} \left( 1 - |\lambda_{V_k}|^2 \right) \left( 1 - |\lambda_{U_j}|^2 \right) \frac{|b_{jk}|^2}{1 - |\lambda_{V_k}|^2 |\lambda_{U_j}|^2}$$  (18)

where $\lambda_{V_k}$, $\lambda_{U_j}$ are the respective eigenvalues of $V$ and $U$ and $b_{jk}$ is the $(j,k)^{th}$ element of $B$.

As visible from 18, the total capacity is dependent on the mixture of eigenvalues of the connectivity matrices unlike the capacity in normal vector representation networks where the eigenvalues of the recurrent connectivity all add up to 1, thus yielding $J_{\text{tot,rel}} = 1$ relative to the input Fisher information. On the other hand, in the case of dynamical system 5, we have the input Fisher information given through the following lemma,

**Lemma 2. (Input Fisher Information)** The Fisher information received by the state of the dynamical system in 5 at each timestep is given by,

$$I(W) = \frac{1}{\varepsilon_1 \varepsilon_2} \text{Tr}(W^T W) = \frac{1}{\varepsilon_1 \varepsilon_2} \|W\|_F^2$$  (19)

where $\|\cdot\|_F$ is the Frobenius norm.

We prove Lemma 2 in Appendix B. With the input Fisher information in hand, we now see an interesting case arising from Theorem 2 when $U$ and $V$ are further assumed to be convergent apart from being normal. In particular, we observe that $g(|\lambda_{V_k}|, |\lambda_{U_j}|) = \frac{(1-|\lambda_{V_k}|^2)(1-|\lambda_{U_j}|^2)}{1-|\lambda_{V_k}|^2 |\lambda_{U_j}|^2} \in (0,1)$ as $0 < |\lambda_{V_k}|, |\lambda_{U_j}| < 1$ due to convergent condition and $\frac{1}{\varepsilon_1 \varepsilon_2} \sum_{j=1}^{N} \sum_{k=1}^{N} |b_{jk}|^2 = \frac{1}{\varepsilon_1 \varepsilon_2} \text{Tr}(B^H B) = \frac{1}{\varepsilon_1 \varepsilon_2} \text{Tr}(W^T W)$ due to normal condition. Hence, we arrive at the following corollary of Theorem 2,

**Corollary 2. (Capacity of Normal Convergent Matrix Networks)** Given the dynamical system 5 and that both the connection matrices $U$ and $V$ are normal as well as convergent, the memory capacity of the system is limited by the following inequality

$$J_{\text{tot}} < \frac{1}{\varepsilon_1 \varepsilon_2} \text{Tr}(W^T W)$$  (20)

which is equivalent to stating that the memory capacity of dynamical system in Equation 5 with respect to instantaneous input Fisher information is

$$J_{\text{tot,rel}} = \frac{J_{\text{tot}}}{I(W)} < 1$$  (21)

The bound in Equation 20 implies that the amount of information stored in the state of Equation 5 about all the previous input signals is less than the amount of input information at the current instance. Equation 21 further implies that the normal convergent...
matrix representation in recurrent neural networks cannot efficiently redistribute the total past information with respect to the information just observed. Thus, the memory capacity of such a network would be sub-optimal and would not perform well on short/long term recall task and also will be likely to fail in generalizing over those respective tasks. This fact is loosely shown by Figure 5 and 7.

One final thing to note about the assumption of convergence is that it is not required in the case of vector representation networks, as magnitude of all eigenvalues have to be less than 1 due to asymptotic stability criterion (Ganguli et al., 2008).

3.2.2 Capacity of General Non-Normal Networks

So far, we have seen the inability of normal convergent connectivity in matrix recurrent neural networks in storing information about past signals. However, we have only dealt under the constraint of normal connectivity matrices. It is not yet clear how the performance might change if we relax this constraint (whether $J_{tot}$’s upper bound increases or decreases even further). Relaxing this constraint, however, is not straightforward as can be seen from difficulty in direct analysis of 10 and 15. Under similar challenges, Ganguli et al. (2008) defined a broader form of Fisher memory matrix, known as space-time Fisher memory matrix. Here, the temporal signal $s_k$ was assumed to be supported also by the spatial dimension introduced by the feedforward connectivity vector for each signal at each time. This formulation depends on the fact that the form of $J_{i,j}$ turns out to be simpler in the case of vector representations as seen in Equation 14, unlike Equation 13. Specifically, the vector feedforward connection $v$ in 14 acts on $W_i^T C_n^{-1} W_j$ which can be seen as a matrix storing the information regarding all the elements of $v$ (which can be regarded as the spatial dimension) for each time step, which leads to the formulation of $J_{st}^{(i,m),(j,n)} = [W_i^T C_n^{-1} W_j]_{(m,n)}$ (Note the different context of $W$ for vector dynamics). The introduction of $J_{st}$ makes the analysis of $J(i)$ for vector representation dynamics for non-normal connectivity much more accessible.

The formulation of an equivalent paradigm in matrix representation dynamics seems to be a non-trivial task however, especially due to the fact that we can’t separate the feedforward connection matrix $W$ from the matrix products inside the trace operator in Equation 13. The presence of $W$ in between the products indicates the presence of complicated interaction terms between the feedforward connection and the recurrent connections. This interaction might be a result of using structured matrix representations which we explain further.

The spatial dimension in matrix representations should correspond to the signal $s_i = s(n-i)$ reaching each neuron in the state matrix $X(n)$. Note that the input signal is a scalar $s_k$, thus for vector representations, the $k^{th}$ neuron receives $v_k s(n-i)$ as the signal. Similarly in matrix representations, the $(k,l)^{th}$ neuron will receive $W_{k,l} s(n-i)$, thus $W$ provides structure to input signal to reach the neuron. Without this structural support, the signal can clearly not reach the state neuron. Thus, it seems logical that the capacity

---

2. Loosely in the sense that the topology of the network studied here and the one used in the experiments differ substantially, yet the results are same, that is the Matrix RNN fails to learn in the experiments discussed later on.
of matrix representations would directly depend upon the matrix that stratifies the input signal to each neuron.

We can see this effect if we assume that each recurrent connectivity matrix $U$ and $V$ induces their respective spatio-temporal Fisher memory matrix $J_{st}^{U}$ and $J_{st}^{V}$, which are similar to structure as in the vector representations as discussed in the beginning. They, $J_{st}^{U}$ and $J_{st}^{V}$ store the Fisher information that $X(n)$ contains about the interference of the input signals which appears $i$ and $j$ time steps in the past along all their respective spatial dimensions for the respective connection matrices $U$ and $V$, as shown below

\[ J_{st}^{V}(i,j) = V^j \Sigma^{-1}V^iT \]
\[ J_{st}^{U}(i,j) = U^i \Psi^{-1}U^jT \]  

Using 22, we can reform the 13 in a much simpler form which shows the effect of each of these matrices on the $J_{i,j}$,

\[ J_{i,j} = Tr \left( W J_{st}^{V}(i,j) W^T J_{st}^{U}(i,j) \right) \] 

and thus, the $J_{tot}$,

\[ J_{tot} = \sum_{i=0}^{\infty} Tr \left( W J_{st}^{V}(i,i) W^T J_{st}^{U}(i,i) \right) \]  

Thus, the Fisher memory matrix $J_{i,j}$ depends upon the interaction between each of the spatio-temporal memory matrix along with the feedforward connection which connects input signal to the state neurons. Now, since each of the spatio-temporal matrices are positive definite and $\varepsilon_2 \sum_{i=0}^{\infty} Tr J_{st}^{V}(i,i) = \varepsilon_1 \sum_{i=0}^{\infty} Tr J_{st}^{U}(i,i) = N$, we can retrieve a fundamental bound on the memory capacity $J_{tot}$ which we state as the following theorem (Appendix C)

**Theorem 3.** (Capacity of General Networks) Given the dynamical system 5, the memory capacity of the system is fundamentally limited by the following inequality

\[ J_{tot} \leq \frac{N^2}{\varepsilon_1 \varepsilon_2} Tr \left( W^T W \right) \]  

which in terms of instantaneous relative capacity is,

\[ J_{tot,rel} \leq N^2 \]  

Therefore, the memory capacity of a matrix representation recurrent networks is fundamentally bounded by the size of the state matrix. It’s interesting to note that besides the apparent flaw in capacity under normal connectivity constraints for matrix representation networks, the capacity in general case at least seems to be in-line with the trend first observed with vector representation recurrent networks, where the $J_{tot,rel} \leq N$ for $N$ neurons. However, as visible from the case of normal connectivity matrices, the question that whether the system’s capacity would be close to this bound is not trivially answered. Clearly, the
capacity couldn’t reach the bound by optimizing only $W$, as $J_{tot,rel}$ doesn’t depend on $W$. However, one can argue for the case of high $\|U\|_F$ and $\|V\|_F$ which becomes apparent from 24 and Appendix C.

In our analysis of memory capacity of matrix representation networks so far, we assumed that Equation 5 works under linear dynamics; that is, $f(X) = X$. However, in practice some type of non-linearity is used for learning efficient and diverse representations. The same can be said for biological neurons as their activation’s are not unbounded, hence the finite dynamic range. We next show that adding a saturating non-linearity like $\sigma(x)$ or $\tanh(x)$ causes the capacity to also be bounded by a multiple of the same range.

### 3.3 Effects of finite dynamic range

So far, we’ve seen the asymptotic effects on the memory capacity of the linear dynamics matrix representation networks. However, it still remains to be answered the effects on the capacity of the system when the dynamics is restricted to some finite dynamic range as in the biological neurons through other saturating non-linearities commonly used in machine learning community.

#### 3.3.1 Memory Capacity is upper-bounded by the norm of state

To see whether such non-linearities increase or decrease the memory capacity, we first assume that the network architecture is such that neural activity of each neuron in $X(n)$ is limited between $-\sqrt{R}$ and $\sqrt{R}$; hence, restricting $\text{Tr}(X^T X) < N^2 R$.

Looking at the average of the state norm of Equation 5 which is further shown below in Equation 27, we see that $\text{Tr}(X^T X) < N^2 R$ implies that each of it’s components are also bounded by $N^2 R$

\[
\mathbb{E} [\text{Tr}(X^T X)] = \sum_{k=0}^{\infty} \|U^k W V^k\|_F^2 + \varepsilon_1 \sum_{k=0}^{\infty} \text{Tr}(V^k V^k) \text{Tr}(U^k U^k) \tag{27}
\]

which implies that $\sum_{k=0}^{\infty} \|U^k W V^k\|_F^2 < N^2 R.$
On the other hand, since $\Sigma^{-1}$ and $\Psi^{-1}$ are positive definite, we see that the capacity of the matrix network is

$$J_{tot} = \text{Tr} \left( \sum_{i=0}^{\infty} \Sigma^{-1} V^{iT} W^{T} U^{i} \Psi^{-1} U^{iT} W V^{i} \right)$$

$$= \text{Tr} \left( \Sigma^{-1} \sum_{i=0}^{\infty} V^{iT} W^{T} U^{i} \Psi^{-1} U^{iT} W V^{i} \right)$$

$$\leq \text{Tr} \left( \Sigma^{-1} \right) \text{Tr} \left( \Psi^{-1} \right) \left( \sum_{i=0}^{\infty} \text{Tr} \left( V^{iT} W^{T} U^{i} U^{iT} W V^{i} \right) \right)$$

$$\leq \text{Tr} \left( \Sigma^{-1} \right) \text{Tr} \left( \Psi^{-1} \right) \sum_{i=0}^{\infty} \| U^{iT} W V^{i} \|_F^2$$

$$\leq \text{Tr} \left( \Sigma^{-1} \right) \text{Tr} \left( \Psi^{-1} \right) E \left[ \text{Tr} \left( X^{T} X \right) \right]$$

$$< \text{Tr} \left( \Sigma^{-1} \right) \text{Tr} \left( \Psi^{-1} \right) N^2 R$$

Hence, when we restrict the neuronal dynamic range of activations between certain thresholds, the memory capacity of the network is also limited by that same threshold with appropriate constants. This implies that in the case of saturating non-linearities like sigmoid or tanh, the memory capacity $J_{tot}$ may decrease whereas for non-saturating non-linearities like ReLU and its derivatives, this bound doesn’t create a problem from a memory standpoint.

So far, we’ve seen an analysis on the memory capacity for matrix representation recurrent network as described by the linear dynamics of Equation 5. We first considered memory capacity under the constraint of normal connectivity matrices, in which we proved the limited capability of such a network. We did this under further convergence assumption by storing information about past signals which turns out to be worse than vector representation recurrent network under same constraints.

We then further discussed the general case and showed the information stored in the state relative to input information cannot exceed the number of neurons, which thus seems to generalize from the similar results obtained for vector representation recurrent neural network. The question that now naturally arise is whether there are ways in which $J_{tot}$ can be increased. One obvious way of achieving that is through addition of an external memory resource to the recurrent neural network which on an intuitive level does increases the memory capacity of the network. Even though there has been very in-depth work on the memory of the non-linear vector neural networks, there hasn’t been any which extends those ideas to architectures which have external memory available to exploit, let alone the higher order representations that we deal with in this paper. We thus explore the idea of quantifying the memory capacity of an exceedingly simple memory network with matrix representations through the definitions introduced in previous sections and provide the absolute minimum increase in memory that one would expect from such a memory architecture.
4. The Effects of External State Memory on State Dynamics

We earlier saw the memory capacity shown in Equation 18 of the dynamical system Equation 5. In this section, we explore one possible path to increase the aforementioned memory capacity.

4.1 Direct ways to increase the capacity

We start by first noting the FMM in Theorem 1 of matrix networks. In order to increase this notion of memory capacity, we should clearly either (a) increase the mean $M$ or (b) decrease the covariances $\Sigma$ and $\Psi$. However, doing either of them is not clear at first sight. To increase the mean $M$, we can try adding more terms to it, which can be done easily if we note the state solution in Equation 11 is obtained via solving the recurrent relation in Equation 5 in the linear case. Hence, if we add any term in the state dynamics 5, then we would get the resultant in Equation 11. Now on the question of what we should add to the RNN state in Equation 5 to have a larger mean, can be answered by noting that if we give the state at time step $n$ the access to past states, then the state solution would be expected to contain the solutions of each of those past states too, thus increasing the terms in the state solution. Note that such a structure can be termed under the well-explored Memory Networks (Weston et al., 2014). In memory networks, the state at each time step is also supplemented through a term read from an external memory in the previous time step and the term which is written to memory is usually some transformation of the current state itself (Graves et al., 2014, 2016; Santoro et al., 2016). Here, we assign the external memory to store certain past states, thus assuming no specific structure that determines what should be stored in the memory at the current time step. Note that such a memory should have a queue-like behavior. We explore this idea in detail now and derive the corresponding capacity of such an architecture.

4.2 Addition of State Memory to State Dynamics

In this section, we modify Equation 5 by adding a generic finite queue-memory which stores the past state representations for the current state as an additional input. The motivation behind this is to access past representations for the current state in order to simplify the task of encoding each past input signal for that state. The main goal now is to find the best representation only for a small previous neighborhood of current input signal instead of all past signals, while encoding minimal to no information for those past signals as they are readily available through this type of memory. We further show that even with a bounded memory size of just one slot, the memory capacity becomes an infinite matrix series sum which is at the very least is upper bounded by four times the capacity of the matrix representation networks without memory access at each time step as given in Equation 10. Hence, we mathematically prove the increase in capacity such a memory structure will have for matrix representation in RNNs.
4.2.1 The change in dynamics

Consider the following dynamical system

\[ X(n) = U^T X(n - 1) + W s(n) + Q_{READ}(n) + Z(n) \]  

(29)

where \( U, V, W \) and \( Z \) have the same meaning as earlier. The \( Q_{READ}(n) \) represents the matrix read from memory at time \( n \), which in general can be a function on the set of all the past elements of memory \( \{Q(t) : 0 \leq t < n - 1\} \). In practice, most of the memory based neural architectures only consist of finite memory span while the read operation mapping from the memory \( Q \) to an extracted read element happens through a key-value retrieval mechanism. In most cases, this map is linear where a scalar strength is attributed to each slot of memory based on a key usually generated through an RNN (Graves et al., 2014; Santoro et al., 2016).

In the same motivation, one can define a queue-like memory \( Q_n \) of size \( p \), where \( Q_n[i] \in \mathbb{R}^{N \times N} \) for all \( i \in \{1, \cdots, p\} \) stores the state \( X(n - 1 - i) \); i.e. \( Q_n[i] = X(n - 1 - i) \). The update to memory happens as an \texttt{enqueue()} operation which adds \( X(n - 1) \) as a new slot on the front and at the same time the operation \texttt{dequeue()} removes the slot \( Q_n[p] \) containing the state \( X(n - 1 - p) \). To read from \( Q_n \), we describe a sequence of scalars \( \{\alpha_k\} \) for \( k \in 1, \cdots, p \) which describes the strength of each of the memory locations in the current time step \( n \), such that \( \sum_{k=1}^{p} \alpha_k = 1 \). The sequence \( \{\alpha_k\} \) can be functions of the previous state, can be dependent on the input signal or just be constant (perhaps, they may give equal strength to all slots for time \( n \)). With this construct, we can now define the \( Q_{READ}(n) \) formally as

\[ Q_{READ}(n) = \sum_{t=1}^{p} \alpha_t Q_n[t] = \sum_{t=1}^{p} \alpha_t X(n - 1 - t) \]  

(30)

We now see how memory capacity of system in Equation 29 is altered in comparison to the system in Equation 5 which will be further discussed in next section.

4.2.2 Calculating the altered FMC in a simple case

With the memory structure in place, we now state the FMC of Equation 29 and then prove it in the discussion that follows.

**Theorem 4.** (FMC of Matrix Memory Network) The Fisher Memory Curve of the dynamical system 29 and 30 with \( p = 1 \) is

\[ J(k)' = \text{Tr} \left( \Sigma_{\text{MEM}^{-1}} \Sigma_{\text{STATE}^{-1}} \left( \partial s_k M^T_{\text{MEM}} \right) \Psi_{\text{MF}^{-1}} \Psi_{\text{STATE}^{-1}} \left( \partial s_k M \right) ight) + \Sigma_{\text{MF}^{-1}} \Sigma_{\text{STATE}^{-1}} \left( \partial s_k M^T \right) \Psi_{\text{MF}^{-1}} \Psi_{\text{STATE}^{-1}} \left( \partial s_k M \right) \]

where,

\[ \Sigma_{\text{MF}^{-1}} = I - \Sigma_{\text{STATE}^{-1}} \left( \Sigma_{\text{MEM}^{-1}} + \Sigma_{\text{STATE}^{-1}} \right)^{-1} \]

\[ \Psi_{\text{MF}^{-1}} = I - \Psi_{\text{STATE}^{-1}} \left( \Psi_{\text{MEM}^{-1}} + \Psi_{\text{STATE}^{-1}} \right)^{-1} \]
where \( \Sigma^{-1}_{\text{STATE}}, \Sigma^{-1}_{\text{MEM}}, \Psi^{-1}_{\text{STATE}} \) and \( \Psi^{-1}_{\text{MEM}} \) are given by 36.

**Corollary 3.** (Worst case upper bound on capacity) From the Theorem 4, the worst case upper bound on the capacity of \( 29 \ J'_\text{tot} \) with respect to the capacity of \( 5 \ J_{\text{tot}} \) (Equation 15) with identical connection matrices can be seen to be

\[
J'_\text{tot} \leq 4J_{\text{tot}}
\]

The importance of the above corollary can be seen through the simulation (Figure 2 E), which raises the question of what can be the minimum capacity of the system in Equation 29?

Addition of memory to Equation 5 comes with obvious benefits as discussed earlier. However, proving these effects mathematically might give us clue about the limits of such effects. To derive the Fisher Memory Curve for Equation 29, we first note that an attempt at finding general solution of Equation 29 yields the following recurrence relation

\[
X(n) = \sum_{k=0}^{\infty} U^{kT} W s_k V^k + \sum_{k=0}^{\infty} U^{kT} Z(n-k) V^k + \sum_{k=0}^{\infty} U^{kT} \left( \sum_{t=1}^{p} \alpha_t X(n-k-1-t) \right) V^k
\]

where, we see an extra term as compared to Equation 11. It’s not difficult to see now the inclusion of sum of \( Q_{\text{READ}}(n-k) \) for infinitely many \( k \) makes the state equation difficult to analyze. We need to recursively substitute \( X(n-k-\cdots) \) infinitely many times, each one being an infinite sum itself. However, we can simplify the analysis by making a very reasonable and widely used assumption that more memory (slots) will lead to much broader access to the past states and thus, will lead to better performance. With this simple assumption, it only becomes necessary for us to show the effectiveness of \( p = 1 \) or just 1 slot of memory which can, due to our assumption, can act as a lower bound on the memory capacity thus derived. Hence, we now only analyze the simple but effective case of \( p = 1 \).

Note that in this case, \( \{\alpha_k\} = \alpha_1 \) and \( \alpha_1 = 1 \).

With \( p = 1 \), we can re-write 31 as follows

\[
X(n) = \sum_{k=0}^{\infty} U^{kT} W s_k V^k + \sum_{k=0}^{\infty} U^{kT} Z(n-k) V^k + \sum_{i_1=2}^{\infty} U^{(i_1-2)T} X(n-i_1) V^{i_1-2}
\]

For ease of representation, let us define \( A = \sum_{k=0}^{\infty} U^{kT} W s_k V^k \) and \( B^n = \sum_{k=0}^{\infty} U^{kT} Z(n-k) V^k \). Therefore, we can now substitute the value of \( X(n-i_1) \) in the 32 as follows to get the following form

\[
X(n) = A + \sum_{i_1=2}^{\infty} U^{(i_1-2)T} A V^{i_1-2} + \sum_{i_1=2}^{\infty} \sum_{i_2=1}^{\infty} U^{(i_1+i_2-4)T} X(n-i_1-i_2) V^{i_1+i_2-4}
\]

\[
+ \sum_{i_1=2}^{\infty} U^{(i_1-2)T} B^{n-i_1} V^{i_1-2} + B^n
\]
It’s now trivial to see the above’s extension to infinite sums of combinations of \( A \) and \( B^{(n-i_1-i_2-...)} \). Before that, let us define the following

\[
S_A(m) = \sum_{i_m=2}^{\infty} \cdots \sum_{i_1=2}^{\infty} U_{\Sigma_{j=1}^{m} i_j-2m} T A V_{\Sigma_{j=1}^{m} i_j-2m} \\
S_B(m) = \sum_{i_m=2}^{\infty} \cdots \sum_{i_1=2}^{\infty} U_{\Sigma_{j=1}^{m} i_j-2m} T B^{(n-\Sigma_{j=1}^{m} i_j)} V_{\Sigma_{j=1}^{m} i_j-2m}
\]

where \( S_A(0) = A \) and \( S_B(0) = B^n \). With 34, we can now present the general solution of \( X(n) \) as the following

\[
X(n) = \sum_{m=0}^{\infty} [S_A(m) + S_B(m)]
\]

By the same arguments as in Section 3.1, the mean and covariance matrices for \( p(X(n)|s) \) can be easily seen to be (note that both rows and columns of \( Z(n-k) \) are independent)

\[
M(s) = \sum_{m=1}^{\infty} \sum_{i_m=2}^{\infty} \cdots \sum_{i_1=2}^{\infty} U_{\Sigma_{j=1}^{m} i_j-2m+k} T W V_{\Sigma_{j=1}^{m} i_j-2m+k} s_k + \sum_{k=0}^{\infty} U^k T W V^k s_k
\]

\[
\Psi = \varepsilon_1 \sum_{m=1}^{\infty} \sum_{i_m=2}^{\infty} \cdots \sum_{i_1=2}^{\infty} U_{\Sigma_{j=1}^{m} i_j-2m+k} T U_{\Sigma_{j=1}^{m} i_j-2m+k} + \varepsilon_1 \sum_{k=0}^{\infty} U^k T U^k
\]

\[
\Sigma = \varepsilon_2 \sum_{m=1}^{\infty} \sum_{i_m=2}^{\infty} \cdots \sum_{i_1=2}^{\infty} V_{\Sigma_{j=1}^{m} i_j-2m+k} T V_{\Sigma_{j=1}^{m} i_j-2m+k} + \varepsilon_2 \sum_{k=0}^{\infty} V^k T V^k
\]

We refer back to Equation 36 and note that addition of just one slot of state memory to the state changes the memory dynamics hugely. We can further classify the mean \( M \) and covariance matrices \( \Psi, \Sigma \) into two components identified from the origin of the contribution, whether the contribution comes from addition of memory or the memory innately to state itself which we have discussed earlier. Such a classification can help us to understand the scale of contribution to memory capacity of the dynamics 29 by each of the two classes. Note that the case when \( m = 0 \) just corresponds to the case when there’s no state memory attached to the dynamics.

Deriving the FMC is straightforward now, considering Equation 36 as stated below

\[
J'_{k,k} = J(k)' = \text{Tr} \left( \Sigma^{-1} \frac{\partial M(s)^T}{\partial s_k} \Psi^{-1} \frac{\partial M(s)}{\partial s_k} \right)
\]
where
\[
\partial_{s_k}M = \frac{\partial M(s)}{\partial s_k} = \left( \sum_{m=1}^{\infty} \sum_{i_m=2}^{\infty} \sum_{i_1=2}^{\infty} U(\sum_{j=1}^{m} i_j - 2m + k)^T W V \left( \sum_{j=1}^{m} i_j - 2m + k \right) + U^T W V^T \right) \partial_{s_k}M \text{MEM}
\]

Looking at Equation 37 in the context of Equation 38, we can see informally that due to the heavy contribution from the memory towards all of the measures in Equation 36, the \(J_{k,k}'\) seems to be much general and larger than \(J_{k,k}\) encountered earlier in the case without memory structure \(Q\) (Equation 10). More formally, using Equation 36, we get the following long form of FMC \(J(k)\)'

\[
J(k)' = \text{Tr} \left( \Sigma^{-1} \left( \partial_{s_k}M \text{MEM}^T \right) \Psi^{-1} \left( \partial_{s_k}M \text{MEM} \right) + \Sigma^{-1} V^T U^T \Psi^{-1} \left( \partial_{s_k}M \text{MEM} \right) \right)
\]

\[
+ \Sigma^{-1} \left( \partial_{s_k}M \text{MEM}^T \right) \Psi^{-1} U^T W^T V^T + \Sigma^{-1} V^T U^T \Psi^{-1} U^T W^T V^T J_1'
\]

where \(J_1'\) can be simplified further by using Woodbury matrix identity (Lemma 3, Appendix D) on \(\Sigma^{-1}\) and \(\Psi^{-1}\) which yields the following form of covariance matrices

\[
\Sigma^{-1} = \Sigma_{\text{STATE}}^{-1} - \Sigma_{\text{STATE}}^{-1} \left( \Sigma_{\text{MEM}}^{-1} + \Sigma_{\text{STATE}}^{-1} \right)^{-1} \Sigma_{\text{COMB}}^{-1}
\]

\[
\Psi^{-1} = \Psi_{\text{STATE}}^{-1} - \Psi_{\text{STATE}}^{-1} \left( \Psi_{\text{MEM}}^{-1} + \Psi_{\text{STATE}}^{-1} \right)^{-1} \Psi_{\text{COMB}}^{-1}
\]

where \(\Sigma_{\text{COMB}}^{-1}\) describes the part of the inverse apart from \(\Sigma_{\text{STATE}}^{-1}\) where the covariance contributed by addition of memory and the state itself are combined, similarly for \(\Psi^{-1}\).

We can now use Equation 40 to see clearly that the FMC \(J(k)\) discussed earlier for matrix representation networks without the above discussed queue-like state memory \(Q\) is indeed a small part of \(J(k)'\), because \(J_1'\) now decomposes into the following terms

\[
J_1' = \Sigma_{\text{STATE}}^{-1} V^T W^T U^T \Psi_{\text{STATE}}^{-1} U^T W^T V^T - \Sigma_{\text{COMB}}^{-1} V^T W^T U^T \Psi_{\text{STATE}}^{-1} U^T W^T V^T
\]

\[\text{J(k)w/o Tr operator} \]

\[- \Sigma_{\text{STATE}}^{-1} V^T W^T U^T \Psi_{\text{COMB}}^{-1} U^T W^T V^T + \Sigma_{\text{COMB}}^{-1} V^T W^T U^T \Psi_{\text{COMB}}^{-1} U^T W^T V^T
\]

where we can clearly see the presence of FMC \(J(k)\) of matrix networks without state memory \(Q\) in the FMC for those who consist of \(Q\). In-fact, there are three more independent
\( J(k) \) which can be seen in Equation 39 in each of the terms excluding \( J' \). Using this fact, we can discuss the worst case upper bound on the capacity of Equation 29, when all the other positive terms are zero, which can thus easily be seen to be

\[
J'_\text{tot} = \sum_{k=0}^{\infty} J(k) \leq 4 J_{\text{tot}} \text{ in the worst case.}
\]  

(42)

Hence, the addition of more terms along with \( J(k) \) to \( J(k)' \) is evident from addition of the queue-like memory \( Q \). However, we also note the difficulty of proving more general bounds for the capacity of Equation 29 unlike Equation 5, which occurs mainly due to inclusion of complicated infinite sums over \( m \), where each sum is itself a distorted version of \( J(k) \), as visible from 39 which do not lend itself to same analysis techniques derived in this paper so far and in Ganguli et al. (2008), hence more work might be needed in this direction.

However, we can still generalize 39 by using Lemma 3 on all four terms inside Tr operator instead of just \( J'_1 \) which yields us the following interesting result,

\[
J(k)' = \text{Tr} \left( \Sigma^{-1}_{\text{MF}} \Sigma^{-1}_{\text{STATE}} (\partial_{s_k} M^{T}_{\text{MEM}}) \Psi^{-1}_{\text{MF}} \Psi^{-1}_{\text{STATE}} (\partial_{s_k} M) + \Sigma^{-1}_{\text{MF}} \Sigma^{-1}_{\text{STATE}} (\partial_{s_k} M^{T}) \Psi^{-1}_{\text{MF}} \Psi^{-1}_{\text{STATE}} (\partial_{s_k} M_{\text{STATE}}) \right)
\]  

(43)

where,

\[
\Sigma^{-1}_{\text{MF}} = I - \Sigma^{-1}_{\text{STATE}} (\Sigma^{-1}_{\text{MEM}} + \Sigma^{-1}_{\text{STATE}})^{-1}
\]

\[
\Psi^{-1}_{\text{MF}} = I - \Psi^{-1}_{\text{STATE}} (\Psi^{-1}_{\text{MEM}} + \Psi^{-1}_{\text{STATE}})^{-1}
\]  

(44)

Here, \( \Sigma^{-1}_{\text{MF}} \) can be understood as the fraction of precision provided by memory \( Q \) via \( \Sigma^{-1}_{\text{MEM}} \), similarly for \( \Psi^{-1}_{\text{MF}} \). Note that \( \Sigma^{-1}_{\text{STATE}} \) and \( \Psi^{-1}_{\text{STATE}} \) is the same set of precision matrices as in the case when there was no memory \( Q \) given by Theorem 1. Due to this, one can say that addition of queue-like state memory \( Q \) marginalizes the precision to effective precision matrices \( \Sigma^{-1}_{\text{MF}} \Sigma^{-1}_{\text{STATE}} \) and \( \Psi^{-1}_{\text{MF}} \Psi^{-1}_{\text{STATE}} \) based on the respective precision contributed due to inclusion of \( Q \), i.e. \( \Sigma^{-1}_{\text{MEM}} \) and \( \Psi^{-1}_{\text{MEM}} \).

From the point of view of the FMC \( J(k)' \) with respect to mean \( M \), the \( J(k)' \) is divided among the contribution from the mixture of the derivative of the complete mean \( M \) and the contribution to mean via memory \( M_{\text{MEM}} \) and from the derivative of \( M \) again with mean contributed via state \( M_{\text{STATE}} \). The presence of derivatives of different parts of mean \( M \) in each of the matrices inside the trace operator in Equation 43 unlike Corollary 1 in which the derivative of the same mean were taken both times. This implies a stark difference in the underlying structure of \( J(k)' \) compared to \( J(k) \) where same derivative is taken in the only term present inside trace operator.

We note that the space-time analogy introduced previously in Section 3.2.2 would not be feasible with Equation 43 and hence combined with the above discussion on marginalization of covariances, in this context, deriving a general bound is not trivial. Due to our
first assumption that the capacity would increase with more amount of slots $p$, we thus conclude that adding the queue-like state memory $Q$ increases the memory capacity of matrix representation networks.

In summary, we introduced the queue-like state memory $Q$ to the state dynamics as defined by Equation 29. We then derived the FMC for such a memory augmented dynamics, and showed it’s complex structure, showing that the worst case upper bound of the capacity with just one slot of state memory is 4 times higher than that of the matrix networks without memory. Finally, we gave more insights on to the structure of the FMC by deriving even general form of FMC (Equation 43) and posing some insights about it’s basic structure.

5. Matrix Representation in Neural Memory

In the previous section, we revealed how the addition of state memory to a matrix representation recurrent network will effect the memory capacity of the system. In particular, we derived the Fisher memory curve $J'(i)$ and thus the memory capacity $J'_{tot}$ and argued that it is larger than the $J_{tot}$ derived earlier for matrix representation networks without the memory structure $Q_n$.

We need to note that a higher memory capacity doesn’t necessarily imply that a neural network architecture can efficiently transfer stored knowledge in its hidden representations to achieve the desired output. The memory structure $Q_n$ introduced earlier in Section 4.2 only serves the purpose of increasing the memory capacity, not the efficiency of representations in encoding past inputs clearly (Bengio et al., 2013).

It thus can be argued that the dynamical system in Equation 29 is not suited for real-life tasks where one immediate problem often faced is the similarity of input sequences fed across considerably long time span. One would expect the memory structure $Q_n$ to handle such cases and in-fact use the previously seen representation of the same input in the current time step to generate the new representation for the current time step. Hence an obvious flaw with this structure is the lack of coherence of the hidden representations. Now for a similar input, we have two different representations, one when the first input arrived long ago in the past and the new updated representation which is a result of current similar input and that past state. However, this will only be the case if the past representation is not deque() out of the memory $Q_n$. That is, the time span between the similar signals is not more than the memory size $p$ of $Q_n$ and even if the time span between the inputs is less than the memory size. The sequence $\{\alpha_n\}$ should be such that all weight is given to the slot, where the past representation is currently stored which can only be the case if $\{\alpha_n\}$ is learned to look at the correct address in the memory $Q_n$. It can also be seen that this problem is only exaggerated if there are multiple similar inputs in the input sequence whose periodicity is greater than the memory size $p$ of $Q_n$ which can be the case in recall and copy tasks.

There can be multiple ways to solve the above problem. The memory can act as a placeholder for storing similar or correlated input’s representations in one slot only, thus all the information about similar inputs are available in one place and can be decoded accordingly, perhaps like an encoder-decoder structure (Cho et al., 2014). Thus, instead of storing current state in its entirety, one can instead use it to determine the closest representation already available in the memory and the updates it may need to fit the
representation of the similar group of inputs. We can use weights as an unique address of each slot for finding the location of closest representation in memory. Hence, the weights should somehow be able to look at each slot and determine which one is closest to the current representation.

We note that this problem of finding techniques for efficient storage and retrieval of memory elements is not new in the machine learning research community and there’s been multiple proposals for achieving efficient storage and retrieval for different domains (Santoro et al., 2016; Gulcehre et al., 2017; Le et al., 2019). One of the simplest of these proposals is the Neural Turing Machine (NTM) and the memory addressing mechanism it proposes. We thus propose to use the same addressing mechanisms for the memory which now stores matrix representations (Gao et al., 2016) as opposed to vector representations as originally proposed in NTM. This leads to our proposed Matrix Neural Turing Machine (MNTM) which takes in and stores matrix-sized sequences into a bounded memory. We now formally introduce MNTM and later argue that even though the modifications are in the overall memory structure as compared to the simple $Q_n$ introduced earlier for theoretic evaluations of effect of memory, the capacity doesn’t decrease in comparison while making matrix representations and storage feasible.

5.1 Matrix Neural Turing Machine

Storing the feature vector in a differentiable memory can be traced back to Das et al. (1992). However, storing structured representation has been a recent endeavor (Pham et al., 2018; Khasahmadi et al., 2020). We now introduce a neural memory which stores matrix representations in which addressing is done in the same way as in NTM.

Consider a sequence of input matrices $\{X_n\}$. At time $t$, the matrix $X_t$ is given as an input to the matrix RNN whose structure is described in Figure 1. The matrix RNN also receives the matrices read from memory at time $t-1$, i.e. $R_{t-1}$. Since we use only one Read and Write Head, thus we only receive one matrix read from past memory. The state $H_t$ thus generated by matrix RNN is now used to further generate more elements which determine what to add to and delete from the closest representation to the current input available in the memory so that it can be modified to also fit the current representation. The comparison is done through a key $K_t$ generated to compare the current input’s representation to the representations available in memory. Hence, read matrix $R_t$ is generated using both addressing by content and addressing by location methods for generating the weights $w_t$ for each memory slot.

The addressing by content (Equation 1 in Figure 1) allows to find the index of the memory closest to key $K_t$, where the similarity measure $K[\cdot,\cdot]$ is the cosine similarity as shown in Figure 1. The addressing by location (Equations 2-4, Figure 1) would allow for iterative shifts of the weights, which is an important feature in addition to the content addressing. This will allow to have an alternative mechanism to the content addressing mechanism in the case when there is no clear representation in the memory closest to the key. Hence, we can avoid adding noise to any of the memory slots by shifting the previous attention or the current closest slot index determined by content addressing mechanism by a pre-defined number of steps (usually one in either direction).
Figure 1: Complete structure of MatNTM. As visible, the whole architecture can be thought as matrix analogue of the usual NTMs with a matrix RNN controller network. Matrix FC $\Rightarrow$ Matrix Fully Connected layer.
With references to the theory explained previously in Section 4 and 5, we ought to only care about the retrieval and addition mechanism for memory $M_t$ as the weights $w$ (equivalent to $\{\alpha_n\}$) is retrieved via the (learned) state itself which doesn’t matter much as discussed earlier. However, the representations stored in memory are the one with striking differences with the theoretic $Q_n$, where we only stored the past states altogether. To see the difference more concretely, we can isolate the erase $Er_t$ and addition $A_t$ matrices for a particular time step $t$ as the following,

$$\begin{align*}
Er_t &= U^T_E H_t V_E \\
A_t &= U^T_A H_t V_A
\end{align*}$$

where $U_E, V_E$ and $U_A, V_A$ are the corresponding connection matrices for $Er_t$ and $A_t$ (Figure 1). Note we only exclude the bias term for comparability of the analysis with that of $Q_n$. To analyze the memory retrieval mechanism of MNTM more clearly and comparable to that of $Q_n$, we make an important assumption that the number of slot in memory $M_t$ is one for all $t$. With this assumption in hand, we can thus see that $R_t = M_t$, and thus we get the following recurrence relation (if we consider that initial memory is all zero),

$$R_t = M_{t-1} \odot (1 - Er_t) + A_t$$

$$= (M_{t-2} \odot (1 - Er_{t-1}) + A_{t-1}) \odot (1 - Er_t) + A_t$$

$$= \sum_{k=0}^{\infty} A_{t-k} \odot \left( \prod_{i=0}^{k-1} (1 - Er_{t-i}) \right)$$

$$= \sum_{k=0}^{\infty} U^T_A H_{t-k} V_A \odot \left( \prod_{i=0}^{k-1} (1 - U^T_E H_{t-i} V_E) \right)$$

(45)

where $\odot$ signifies the indexed Hadamard product. Hence, we see that with addition of just one slot of memory $M_t$, the matrix read from it $R_t$ would contain information about all the past states, which is a stark difference to the memory structure $Q_n$ also with only one slot of memory, where the only state read from $Q_n$ was $H_{t-2}$ (Eq. 30). Since we avail the $R_t$ in 45 to the state in next time step, hence essentially we are giving each state the access to all the past hidden states. This would imply the state solution at any time $t$ would contain infinitely many terms equivalent to the third term in Equation 32; one for each past state, thus making the asymptotic study of $M_t$ at least from a capacity standpoint essentially infeasible.

Note that when there are more than one slot in $M_t$, then the weighing $w$ can select which past states to group together in a particular slot or more than one slot (Writing) and which group to retrieve (Reading), which would thus give it a type of selection action over possibly each of the past states or the groups of past states stored in a slot which can be beneficial for the long-term recall tasks as shown by the experiments which follows.
Figure 2: A) FMC for vector (Equation 14) and matrix (Equation 10) representation networks with normal connections. B) FMC for matrix representation net for larger no. of neurons. C) Cumulative sum of $J(i)$ for vector and matrix representation nets. The sum asymptotically converges to 1 for vector nets while it remains $< 1$ for matrix nets for any number of neurons as shown in C) and also in D) for higher no. of neurons. E) For even higher no. of neurons for matrix nets, this figure shows their corresponding $J_{tot}$. It can be seen that variance in capacity is small as the no. of neurons is increased. All the above simulations assume $\text{Tr}(W^TW) = 1$ and $\varepsilon_1 = \varepsilon_2 = 1$ so that $J_{tot_{rel}} = J_{tot}$. 

25
Figure 3: The FMC of the system in Eq. 26 for varying amount of neurons. The area covered by each FMC, highlighted in pink, is clearly proportional to the memory capacity (Definition 1 and see Figure 4). The apparent rough nature of $J(k)'$ for some FMCs as in MAT-196 & MAT-225 might be due to approximation of Eq. 36 to a finite value of $m$ for practical realization. Note that the results are obtained for normal connections, as in general cases, frequent overflow can occur.
Figure 4: Above figure shows the cumulative sum of FMC at each timestep for the system in Eq. 29 with $p = 1$ for varying number of neurons, where the FMCs are plotted in Figure 3. Since the connection matrices are normal and convergent, thus the $J_{tot} < 1$ for a simple matrix network without queue memory $Q_n$ with the same recurrent connection matrices, hence if the worst case is achieved, then the upper bound (Corollary 2) implies $J'_{tot} < 4$ which clearly isn’t the case in all the above simulations, thus showing that the above cases are not worst case scenarios. Besides, we see that the capacity converges to a very large value in all cases.
6. Simulation

Corollary 2 & 3 describes the fundamental bounds on the memory capacity of certain type of matrix representation networks. However, it does not give much details about how traces of memory would achieve this bound. Hence, we now present simulations of the capacity of systems in Equation 5 and Equation 29 for normal connectivity and varying amount of neurons.

In particular, we see that the capacity $J_{tot}$ of the linear matrix representation network with normal convergent connectivity matrices is always less than 1 (Theorem 2) whereas the capacity for vector representation networks sums to 1 as shown in Figure 2 C). The sharper decrease in the FMC for linear matrix nets can also be seen Figure 2 A). We also see the apparent random nature of the memory capacity of such networks with varying number of neurons in Figure 2 E). Figure 3 shows the FMC and the capacity of the system 29 with normal connectivity and varying amount of neurons (note the overwhelming increase in the capacity).

One can compute $\Sigma$ and $\Psi$ using the discrete time Lyapunov equations. Equation 9 shows the infinite sum of the matrices needed to compute the noise covariances, note that they follow the discrete Lyapunov equations as given below

$$U^T \Psi U + I = \Psi$$
$$V^T \Sigma V + I = \Sigma$$

(46)

However, for the memory capacity of the system in Equation 29 ($J_{tot}'$) with $p = 1$, we use the Equation 36; where each of the term in $\Sigma_{MEM}$ and $\Psi_{MEM}$ can be found by recursive use of discrete time Lyapunov equation on $\Sigma_{STATE}$ and $\Psi_{STATE}$ (Equation 46), respectively.

7. Experiments

In this section, we show results of the MatNTM over two synthetic tasks while comparing it with a basic Matrix RNN. Since the addressing mechanism of the MatNTM is identical to NTM, it is natural to expect the learned memory addressing scheme of MatNTM to be identical to that of NTM for same type of tasks. However, since the main goal of the paper is to introduce higher order input input data to be processed directly, thus we are constrained to use matrix shaped inputs instead of vectors. But since the usual NTM only accepts vector shaped inputs, we construct matrix analogue of each such vector tasks. Hence, we can compare the performance of Matrix RNNs against the MatNTM to see the gain introduced by drastically high memory capacity (Theorem 4 & Figure 4) and improved addressing mechanism (in comparison to system in Equation 29) of the latter.

In order to keep the MatNTM closer to theoretic evaluations done previously, we perform minimal-to-no hyperparameter tuning. Only the size and number of hidden layers of Matrix RNN is changed in between tasks, which thus allows us to portray the difficulty faced by the MatNTM over various tasks much more explicitly. Note that the bilinear mapping between the states introduces quadratic form which can make the training unstable as can be seen from the learning curves in Figures 5 and 7. We use RMSprop (Tieleman and Hinton, 2012) optimizer for all tasks with a learning rate of $10^{-4}$ with tanh non-linearity for all recurrent matrix layers. We noted that using any non-saturating non-linearities like ReLU
and LeakyReLU causes frequent overflow in the operations of the addressing mechanisms of the MatNTM (Figure 1 Eq. 1-4). Further details for training parameters can be found in Table 7.

| Model                  | Batch Size | Input Shape | Hidden State | Memory Size | Learning Rate | No. of Parameters |
|------------------------|------------|-------------|--------------|-------------|---------------|------------------|
| MatNTM - Copy Task     | 16         | [5,5]       | $\frac{3}{15\times15}$ | [120,6,6]   | $1\times10^{-4}$ | 4121             |
| MatNTM - Associative Recall Task | 16         | [5,5]       | $\frac{4}{20\times20}$ | [120,6,6]   | $8\times10^{-5}$ | 7946             |
| Matrix RNN - Copy Task | 16         | [5,5]       | $\frac{3}{15\times15}$ | –           | $1\times10^{-4}$ | 2175             |
| Matrix RNN - Associative Recall Task | 16         | [5,5]       | $\frac{4}{20\times20}$ | –           | $8\times10^{-5}$ | 5675             |

Table 1: Hyperparameters for models used for Experiments. For Copy Task, we take $l$ to be random between 1 and 20. For Associative Recall Task, we also fix $n = 2$ and choose $k$ randomly between 2 and 10. Refer main text for more details.

We present open source code of the implementation with results here[^3].

### 7.1 Matrix Copy Task

A long standing problem in recurrent neural networks has been that of efficient recall of sequences observed long time in past (Pascanu et al., 2013). To benchmark the long term memory capability of MatNTM, we test it on a version of copy task as done in (Hochreiter and Schmidhuber, 1997; Graves et al., 2014) which has been extended for matrix sequences. Note that this framework includes the vector copy task as a special case.

Consider a sequence of $N \times N$ matrices denoted as $\{X_1, X_2, \ldots, X_l\}$ where, for our experiments, $(X_i)_{jk} = \text{Bernoulli} (\frac{1}{2})$ and $l$ can vary for each such sequence. Additionally, consider start-of-file ($X_{sof}$) and end-of-file ($X_{eof}$) delimiters which determine that the content which needs to be copied is present between them. These limiters are added to the sequence at the start and the end respectively to form the final sequence $\{X_n\} = \{X_{sof}, X_1, \ldots, X_l, X_{eof}\}$. Consider a model $L_{\theta}$ with parameters $\theta$ which takes as input a set of matrix (two dimensional) sequences, and gives out a fixed length matrix sequence of a predetermined shape,

$$\{Y_n\} = L_{\theta} (\{X_n\})$$

[^3]: [https://github.com/sydney-machine-learning/Matrix_NeuralTuringMachine](https://github.com/sydney-machine-learning/Matrix_NeuralTuringMachine)
The model is trained on copy task if the parameters \( \theta \) are tuned, so that output and input sequences are exactly same in content and in order, i.e.

\[
\{Y_n\} = \{X_n\}
\]

The training here is carried by minimizing the binary cross-entropy loss between input sequence \( \{X_1, \ldots, X_l\} \) and output sequences \( \{Y_1, \ldots, Y_l\} \).

Figure 5: Matrix Copy Task learning curves for MatNTM and Matrix RNN averaged over five runs.

Figure 5 shows that Matrix RNN fails to learn copy task whereas MatNTM learns this task since it fails to converge as the learning duration increases. The reason for this could be that the Matrix RNN had lack of precise memory of past input matrices and could not reconstruct the input sequence correctly. A simple addition of memory cells, as in the case of LSTM model may have been enough to learn this task for smaller length sequence, however, that would have required more parameters (weights and biases) to be trained. We show that addition of an external memory makes this task learnable by addition of very few more parameters as compared to LSTM models, as low as 5675. In contrast, the copy task (for vector sequences) in NTM with feedforward controller takes close to 17,000 parameters Graves et al. (2014).
As expected, the MatNTM learns the same algorithm for copy task similar to the NTM (Graves et al., 2014) while featuring matrix representation. Note that the input sequence contains an *eof* delimiter channel on the last row and the last element of the input sequence act as the actual *eof* delimiter. The corresponding representations stored in memory corresponding to each input sequence is also shown. By incorporating matrix representations, we have essentially shortened the timespan required for processing the whole input sequence as an usual NTM would have required 30 timesteps compared to just 6 in MatNTM, similarly for the output sequence. This is more evident by the right figure where we see the learned model actively iterating over the memory slots to write the corresponding representations, and during the output, reading from the same locations in the original order.

### 7.2 Matrix Associative Recall Task

The goal of this task is to test the learner’s capability to form links between the data it has previously seen. In particular, we form the following task, extending the one proposed in Graves et al. (2014).

Consider a sequence of matrices defined as an *item* as $\mathbf{X}_{\text{item}} = \{\mathbf{X}_{\text{sof}}, \mathbf{X}_{1}, \ldots, \mathbf{X}_{n}, \mathbf{X}_{\text{sof}}\}$ for $i = 1, \ldots, k$ where $n$ is kept fixed while training whereas the number of items $k$ can vary. Now, the complete sequence of inputs shown to the network can be written as $\{\mathbf{X}_{\text{item}1}, \ldots, \mathbf{X}_{\text{item}k}\}$. After processing this input, the network is shown a special end of input delimiter $\mathbf{X}_{\text{delim}}$ which signifies the end of input sequence and the beginning of a query which is $\mathbf{X}_{\text{query}} = \{\mathbf{X}_{1}, \ldots, \mathbf{X}_{n_c}\}$ where $c \sim \text{Unif}(1, 2, \ldots, k - 1)$. The target in this task is to thus output the next item in the input sequence processed earlier, i.e. for the model $L_{\theta}$, the output sequence $\{\mathbf{Y}_n\} = L_{\theta} (\{\mathbf{X}_n\})$ should be,

$$\{\mathbf{Y}_n\} = \{\mathbf{X}_{1+c+1}, \ldots, \mathbf{X}_{n+c+1}\}.$$

This task hence evaluates learner’s capability to form association between the target and the query based on the past sequences it has been provided with.

We can clearly see in Figure 7 that Matrix RNN cannot learn this task as well, whereas MatNTM seems to work better, however, still not learning completely (except on 2 runs out
of 5 where the model gave zero error). Figure 8 further depicts that the algorithm learned by MatNTM is identical to usual NTMs, albeit with matrix representations.

8. Discussion and Future Work

We studied the memory capacity of a new class of recurrent neural networks introduced by Gao et al. (2016) for generating matrix representations using a bilinear map. We briefly discussed various existing definitions of memory capacity and how most of the definitions proves to be not easily generalizable to matrix representations. We hence use a probabilistic model of the memory capacity using Fisher information as introduced by Ganguli et al. (2008). Using this, we investigated how the memory capacity for matrix representation networks are limited under various constraints, and in general, without any constraints. In the case of memory capacity without any constraints, we found that the upper bound on memory capacity to be $N^2$ for an $N \times N$ state matrix. This seems to generalize the similar upper bound for a vector representation RNN ($N$ for state vector of length $N$). Moreover, we demonstrated that inclusion of saturating non-linearities over the state transition may further bound the total memory capacity even tighter.

The theoretical study and subsequent simulations hence reveals the fundamental bounds on memory capacity as revealed by the proposed definition and the striking increase in memory capacity induced by external memory, which is in-line with basic intuition, but quantitatively has been a difficult task to ascertain. Moreover, we used a notion of memory
Figure 8: Top-most sequence denotes the input given to MatNTM and the output sequence received from the MatNTM. The corresponding next row shows the matrices stored in memory corresponding to that input (and combined with the information of all the cumulative inputs provided by controller RNN). Moreover, the bottom row shows the Read & Write weights distributed temporally which shows clearly the shift in read head to the memory location of the matrix next to the query. The solution learned for this task, as suggested by the bottom row, is identical to Graves et al. (2014).

One of the recent developments in the class of neural network architectures has been of that memory augmented neural networks. However, quantifying the increase in actual memory capacity induced by such an external memory hasn’t been discussed yet. With this motivation, we derived the memory trace (FMC) of a linear matrix recurrent network with queue-like state memory, which provides the current state with the information of not just the state at $t - 1$ timestep, but states at $t - p$ timesteps for a fixed $p \geq 1$. We note that this is very similar to the work of Soltani and Jiang (2016) dubbed as Higher Order RNNs, which also provide the current state with a fixed number of past states, albeit in vector representations.

Extending the memory networks to different types of representations has been proposed in the recent times (Pham et al., 2018; Khasahmadi et al., 2020). We extend this line of work by extending the NTM with matrix representations whose preliminary results on two synthetic tasks shows it’s advantage over the simple Matrix RNN. Due to our theory for memory of matrix networks in Sections (2-4) was developed only with a basic recurrent dynamics in mind, not for the much widely used LSTMs due to obvious difficulties for
Figure 9: Left figure shows the increase of cost with increase in the length \( l \) of the sequence of matrices in the copy task. Right figure shows the increase of cost with increase in the length of an item \( n \). Both are for trained models of MatNTM on respective tasks.

the first study of such kind, we refrained from using LSTM controller in MatNTM for the experiments. However, our internal tests did reveal that MatNTM with Matrix LSTM performs far better than that of the MatNTM with Matrix RNN controller presented in this work. We hope to extend this study for Matrix LSTM models in future.

This work introduced matrix representations as a natural next-step from the vector representations as envisioned by Gao et al. (2016) and subsequently expanded upon in Do et al. (2017). In-fact several such extensions have been made ever since, as touched upon in the introduction, which go one step further and generalizes the notion of neural networks to higher order tensors as the inputs. Clearly, such extensions are heavily non-trivial to work with from any, functional or probabilistic, point of view. Perhaps that is where an information geometric point of view might be helpful for future works in this line of study.

Another way ahead is in developing robust uncertainty quantification framework via Bayesian inference for Bayesian Matrix NTMs. We note that the copy tasks by NTMs are computationally expensive and Bayesian inference via Markov Chain Monte Carlo Methods (MCMC) require thousands of samples or model realisations for sampling the posterior distribution of neural weights and biases. We can incorporate recent frameworks that used MCMC with gradient-methods and parallel computing Chandra et al. (2019) to overcome computational challenges. Moreover, surrogate-based MCMC methods for computationally expensive models can also be used Chandra et al. (2020) along with variational inference methods (Blundell et al., 2015).
Appendix A. Lemma 1: KL-Divergence for Matrix Gaussian Distribution

Consider two normally distributed random matrices $X_1 \sim \mathcal{MN}_{n \times p}(M_1, A_1, B_1)$ and $X_2 \sim \mathcal{MN}_{n \times p}(M_2, A_2, B_2)$. We have that

$$p(X_i) = (2\pi)^{-\frac{1}{2}np} |A_i|^{-\frac{1}{2}p} |B_i|^{-\frac{1}{2}n} \exp\left[-\frac{1}{2} \text{Tr} \left( B_i^{-1} (X - M_i)^T A_i^{-1} (X - M_i) \right) \right]$$  (47)

for both $i \in \{1, 2\}$.

The KL-divergence between two distributions $p_1$ and $p_2$ is given by (note log here is the natural logarithm),

$$D_{KL}(p_1 \parallel p_2) = \mathbb{E}_{p_1} \left[ \log \frac{p_1}{p_2} \right]$$

Hence, the KL-divergence between $p(X_1)$ and $p(X_2)$ can be calculated as,

$$D_{KL}(p(X_1) \parallel p(X_2)) = \mathbb{E}_{p(X_1)} \left[ \log p(X_1) - \log p(X_2) \right]$$

$$= \underbrace{\mathbb{E}_{p(X_1)} \left[ \log p(X_1) \right]}_{\text{PART 1}} - \underbrace{\mathbb{E}_{p(X_1)} \left[ \log p(X_2) \right]}_{\text{PART 2}}$$

We first calculate PART 2 as follows,

$$\mathbb{E}_{p(X_1)} [\log p(X_2)] = \mathbb{E}_{p(X_1)} \left[ -\log \left( (2\pi)^{\frac{1}{2}np} |A_2|^{\frac{1}{2}p} |B_2|^{\frac{1}{2}n} \exp\left[-\frac{1}{2} \text{Tr} \left( B_2^{-1} (X - M_2)^T A_2^{-1} (X - M_2) \right) \right] \right) \right]$$

$$= -C_2 - \frac{1}{2} \mathbb{E}_{p(X_1)} \left[ \text{Tr} \left( (X - M_2)^T A_2^{-1} (X - M_2) B_2^{-1} \right) \right]$$

$$= -C_2 - \frac{1}{2} \mathbb{E}_{p(X_1)} \left[ \sum_{i=1}^{p} \sum_{j=1}^{p} \sum_{k=1}^{n} \sum_{t=1}^{n} (x_{ti} - m_{2;ti}) (x_{kj} - m_{2;kj}) u_{2;tk} v_{2;ji} \right]$$

$$= -C_2 - \frac{1}{2} \mathbb{E}_{p(X_1)} \left[ \sum_{i=1}^{p} \sum_{j=1}^{p} \sum_{k=1}^{n} \sum_{t=1}^{n} u_{2;tk}^{-1} v_{2;ji}^{-1} (x_{ti} x_{kj} - m_{2;ti} x_{kj} - m_{2;kj} x_{ti} + m_{2;ti} m_{2;kj}) \right]$$
Now, using the fact that $E[x_{i_1,j_1}, x_{i_2,j_2}] = a_{i_1,i_2} b_{j_1,j_2} + m_{i_1,j_1} m_{i_2,j_2}$ (Gupta and Nagar, 1999) and $E[x_{i_1,i_1}] = m_{i_1,i_1}$ after taking expectation inside the sum, we get:

$$E_{p(X_1)} \left[ \log p \left( X_2 \right) \right] = -C_2 - \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \sum_{k=1}^{n} \sum_{t=1}^{n} u_{i;tk}^{-1} v_{j;ti}^{-1} a_{i;tk} b_{1;ij} + m_{i;tk} m_{1;ij} u_{2;tk}^{-1} v_{j;ti}^{-1} u_{2;tk} v_{j;ti}^{-1} m_{2;ti} m_{1;ij}$$

$$= -C_2 - \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \sum_{k=1}^{n} \sum_{t=1}^{n} u_{i;tk}^{-1} v_{j;ti}^{-1} a_{i;tk} b_{1;ij} + v_{j;ti}^{-1} m_{1;ij} u_{2;tk} m_{1;ij} - v_{j;ti}^{-1} m_{1;ij} u_{2;tk} m_{2;ti}$$

$$= -C_2 - \frac{1}{2} \left( \text{Tr} \left( B_2^{-1} M_1 A_2^{-1} M_1 - B_2^{-1} M_1 A_2^{-1} M_2 - B_2^{-1} M_2 A_2^{-1} M_1 \right. \right.$$

$$\left. \left. + B_2^{-1} M_2 A_2^{-1} M_2 \right) + \text{Tr} \left( A_2^{-1} A_1 \right) \text{Tr} \left( B_2^{-1} B_1 \right) \right)$$

Note that we interchanged the order of summation of Traces in the last line only for the purpose of better representation. After some more rearrangement of terms inside the first Trace, we get the following solution of PART 2,

$$E_{p(X_1)} \left[ \log p \left( X_2 \right) \right] = -C_2 - \frac{1}{2} \text{Tr} \left( A_2^{-1} A_1 \right) \text{Tr} \left( B_2^{-1} B_1 \right) - \frac{1}{2} \text{Tr} \left( B_2^{-1} (M_2 - M_1)^T A_2^{-1} (M_2 - M_1) \right)$$

Calculating PART 1 in a similar path yields us the following,

$$E_{p(X_1)} \left[ \log p \left( X_1 \right) \right] = E_{p(X_1)} \left[ -\log \left( 2\pi \right)^{\frac{1}{2}n p} |A_1|^{\frac{1}{2} p} |B_1|^{\frac{1}{2} n} \right] - \frac{1}{2} \text{Tr} \left( B_1^{-1} (X - M_1)^T A_1^{-1} (X - M_1) \right)$$

$$= -C_1 - \frac{1}{2} E_{p(X_1)} \left[ \sum_{i=1}^{p} \sum_{j=1}^{p} \sum_{k=1}^{n} \sum_{t=1}^{n} u_{i;tk}^{-1} v_{j;ti}^{-1} \left( a_{i;tk} b_{1;ij} + m_{i;tk} m_{1;ij} - m_{1;ij} x_{ti} + m_{1;ij} x_{tk} \right) \right]$$

Using the same two theorems used in calculations for PART 2, we get the following solution of PART 1,

$$E_{p(X_1)} \left[ \log p \left( X_1 \right) \right] = -C_1 - \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \sum_{k=1}^{n} \sum_{t=1}^{n} u_{i;tk}^{-1} v_{j;ti}^{-1} \left( a_{i;tk} b_{1;ij} + m_{i;tk} m_{1;ij} - m_{1;ij} m_{1;ti} \right)$$

$$= -C_1 - \frac{1}{2} \text{Tr} \left( A_1^{-1} A_1 \right) \text{Tr} \left( B_1^{-1} B_1 \right)$$

$$= -C_1 - \frac{1}{2} np$$
Thus, combining the solutions of both PART 1 and PART 2 yields us,

\[
D_{\text{KL}}(p(X_1) \parallel p(X_2)) = -C_1 - \frac{1}{2} np + C_2 + \frac{1}{2} \text{Tr}(A_2^{-1}A_1) \text{Tr}(B_2^{-1}B_1) + \frac{1}{2} \text{Tr}
\left(B_2^{-1}(M_2 - M_1)^T A_2^{-1}(M_2 - M_1)\right)
\]

Substituting the values of \(C_1\) and \(C_2\) back in their respective places, we finally get the required form,

\[
D_{\text{KL}}(p(X_1) \parallel p(X_2)) = \log |A_2^{\frac{1}{2}} B_2^{\frac{1}{2}}|^{\frac{1}{2}} n - \frac{1}{2} np + \frac{1}{2} \text{Tr}(A_2^{-1}A_1) \text{Tr}(B_2^{-1}B_1) + \frac{1}{2} \text{Tr}
\left(B_2^{-1}(M_2 - M_1)^T A_2^{-1}(M_2 - M_1)\right)
\]

**Appendix B. Lemma 2 : Input Fisher Information**

The Fisher Information for a random variable \(x\) is given as,

\[
\mathcal{I}(\theta) = -\mathbb{E} \left[ \frac{\partial^2}{\partial \theta^2} \log f_x(x; \theta) \right] \quad (48)
\]

where \(f_x(x; \theta)\) is the density function of \(x\) under the regularity condition that \(\log f_x(x; \theta)\) is twice differentiable.

Consider now the timestep \(t\), where the input is given to the state via the following,

\[
X(t) = Ws_0 + Z
\]

Note that \(s_0 = s(t-0)\) and \(Z \sim \mathcal{MN}_{n \times p}(0, \text{diag}(\varepsilon_1), \text{diag}(\varepsilon_2))\). Hence, \(X\) will also be a random matrix with mean \(Ws_0\) and same covariance matrices,

\[
X(t) \sim \mathcal{MN}_{n \times p}(Ws_0, \text{diag}(\varepsilon_1), \text{diag}(\varepsilon_2))
\]
We can now calculate Fisher information of $X(t)$ parameterized by $s_0$ simply by using 48 to get,

$$I(s_0) = -\mathbb{E} \left[ \frac{\partial^2}{\partial s_0^2} \left( - \log \left( (2\pi)^{\frac{1}{2}} np |\text{diag}(\varepsilon_1)|^{\frac{1}{2}} |\text{diag}(\varepsilon_2)|^{\frac{1}{2}} \right) - \frac{1}{2} \text{Tr} \left( (X - Ws_0)^T \text{diag}(\varepsilon_1)^{-1} (X - Ws_0) \right) \right) \right]$$

$$= -\mathbb{E} \left[ \frac{\partial^2}{\partial s_0^2} \left( - \frac{1}{2\varepsilon_1 \varepsilon_2} \text{Tr} \left( I(X - Ws_0)^T I(X - Ws_0) \right) \right) \right]$$

$$= \frac{1}{2\varepsilon_1 \varepsilon_2} \mathbb{E} \left[ \frac{\partial^2}{\partial s_0^2} \text{Tr} \left( (X - Ws_0)^T (X - Ws_0) \right) \right]$$

$$= \frac{1}{2\varepsilon_1 \varepsilon_2} \mathbb{E} \left[ \text{Tr} \left( (X - Ws_0)^T (X - Ws_0) \right) \right] \quad \text{(Note that } s_0 \text{ is a scalar)}$$

$$= \frac{1}{\varepsilon_1 \varepsilon_2} \text{Tr} \left( W^T W \right)$$

Appendix C. Theorem 3 : Memory Capacity of General Matrix Representation Network

We have that,

$$J_{tot} = \sum_{i=0}^{\infty} \text{Tr} \left( WJ_{V_{(i,i)}}^{st} J_{U_{(i,i)}}^{stT} \right)$$

where $\varepsilon_2 \sum_{i=0}^{\infty} \text{Tr} \left( J_{V_{(i,i)}}^{st} \right) = \varepsilon_1 \sum_{i=0}^{\infty} \text{Tr} \left( J_{U_{(i,i)}}^{st} \right) = N$. Now to obtain the inequality in Eq. 25, we proceed as follows,

$$J_{tot} = \sum_{i=0}^{\infty} \text{Tr} \left( WJ_{V_{(i,i)}}^{st} \left( J_{U_{(i,i)}}^{st} W \right)^T \right)$$

Since the norm induced by Frobenius inner product $\langle A, B \rangle_F = \text{Tr} \left( A^T B \right)$ will obey Cauchy-Schwarz inequality, i.e. $|\langle A, B \rangle_F| \leq \|A\|_F \|B\|_F$, we hence get,

$$J_{tot} \leq \sum_{i=0}^{\infty} \sqrt{\text{Tr} \left( WJ_{V_{(i,i)}}^{st} J_{V_{(i,i)}}^{stT} W^T \right) \text{Tr} \left( W^T J_{U_{(i,i)}}^{st} J_{U_{(i,i)}}^{stT} W \right)}$$

$$= \sum_{i=0}^{\infty} \sqrt{\text{Tr} \left( W^T W J_{V_{(i,i)}}^{st} J_{V_{(i,i)}}^{stT} \right) \text{Tr} \left( W^T W J_{U_{(i,i)}}^{st} J_{U_{(i,i)}}^{stT} \right)}$$
Now since $W^T W$, $J_{V_{(i,i)}}^{st} J_{V_{(i,i)}}^{st^T}$ and $J_{U_{(i,i)}}^{st} J_{U_{(i,i)}}^{st^T}$ are positive semi-definite and $\text{Tr} \left(J_{V_{(i,i)}}^{st}\right) = \text{Tr} \left(V_i V_i^T \Sigma^{-1}\right) \geq 0$ as $V_i V_i^T$ and $\Sigma^{-1}$ are positive semi-definite, hence we can get,

$$J_{tot} \leq \sum_{i=0}^\infty \text{Tr} \left(W^T W\right) \sqrt{\text{Tr} \left(J_{V_{(i,i)}}^{st} J_{V_{(i,i)}}^{st^T}\right) \text{Tr} \left(J_{U_{(i,i)}}^{st} J_{U_{(i,i)}}^{st^T}\right)}$$

$$\leq \sum_{i=0}^\infty \text{Tr} \left(W^T W\right) \sqrt{\text{Tr}^2 \left(J_{V_{(i,i)}}^{st}\right) \text{Tr}^2 \left(J_{U_{(i,i)}}^{st}\right)}$$

$$= \text{Tr} \left(W^T W\right) \sum_{i=0}^\infty \left| \text{Tr} \left(J_{V_{(i,i)}}^{st}\right) \right| \left| \text{Tr} \left(J_{U_{(i,i)}}^{st}\right) \right|$$

$$\leq \text{Tr} \left(W^T W\right) \left( \sum_{i=0}^\infty \left| \text{Tr} \left(J_{V_{(i,i)}}^{st}\right) \right| \right) \left( \sum_{i=0}^\infty \left| \text{Tr} \left(J_{U_{(i,i)}}^{st}\right) \right| \right)$$

$$= \text{Tr} \left(W^T W\right) \frac{N^2}{\varepsilon_1 \varepsilon_2}$$

**Appendix D. Lemma 3 : Woodburry Matrix Identity, (Woodbury, 1950)**

**Lemma 3.** (Woodbury Matrix Identity) Consider conformable matrices $A$, $U$, $C$ and $V$, then it is true that,

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

**Appendix E. Extended Calculation : Deriving simplest form of Memory Capacity from Eq. 17**

To derive the form of $J_{tot}$ as in Eq. 18 from $J(i)$ (Eq. 17), we would first need to notice that the sum over Trace is just Trace of the sum, enabling us to write,

$$J_{tot} = \frac{1}{\varepsilon_1 \varepsilon_2} \text{Tr} \left( \sum_{i=0}^\infty \left( I - \Lambda_V^2 \right) \Lambda_V^i B^H \Lambda_U^2 \left( I - \Lambda_U^2 \right) B \Lambda_V^i \right)$$

(49)

Since the $J_{tot}$ above includes $B$ and it’s transpose in between the diagonal eigenvalue matrices, thus there will be off diagonal interaction terms in between the eigenvalues, which would complicate the sum. However, one can note that we can write a matrix equation of the form $D_x B^H D_y B D_z$ where $D_x$, $D_y$ and $D_z$ are diagonal matrices formed by column vectors $x$, $y$ and $z$ respectively as follows

$$D_x B^H D_y B D_z = \sum_{j=1}^N y_j \left( x \odot b_j^H \right) \left( z^H \odot b_j \right)$$

$$= \sum_{j=1}^N y_j \left( x z^H \right) \odot \left( b_j^H b_j \right)$$

39
where $b_j$ is the $j^{th}$ row of $B$. Hence, we can represent 49 by letting $x^i = \text{vec} \left( (I - \Lambda_{\mathbf{v}}^2) \Lambda_{\mathbf{v}}^i \right)$, $y^i = \text{vec} \left( (I - \Lambda_{\mathbf{v}}^2) \Lambda_{\mathbf{v}}^i \right)$ and $z^i = \text{vec} \left( \Lambda_{\mathbf{v}}^i \right)$

$$J_{tot} = \frac{1}{\varepsilon_1 \varepsilon_2} \text{Tr} \left( \sum_{i=0}^{N} \sum_{j=1}^{N} y_j \left( x^i (z^i)^H \right) \odot (b_j^H b_j) \right)$$

$$= \frac{1}{\varepsilon_1 \varepsilon_2} \text{Tr} \left( \sum_{j=1}^{N} \sum_{i=0}^{N} y_j \left( x^i (z^i)^H \right) \odot (b_j^H b_j) \right)$$

$$= \frac{1}{\varepsilon_1 \varepsilon_2} \text{Tr} \left( \sum_{j=1}^{N} \left( \sum_{i=0}^{\infty} y_j \left( x^i (z^i)^H \right) \right) \odot (b_j^H b_j) \right)$$

where $\Lambda_j$ is the matrix of interaction in between the eigenvalues of $U$ and $V$ obtained after taking the infinite geometric sum present in each of it’s element, which can be represented as the following matrix,

$$\Lambda_j = \begin{bmatrix}
(1-|\lambda_{V_1}|^2)(1-|\lambda_{U_j}|^2) & \cdots & (1-|\lambda_{V_1}|^2)(1-|\lambda_{U_j}|^2) \\
(1-|\lambda_{V_2}|^2)(1-|\lambda_{U_j}|^2) & \cdots & (1-|\lambda_{V_2}|^2)(1-|\lambda_{U_j}|^2) \\
\vdots & \ddots & \vdots \\
(1-|\lambda_{V_N}|^2)(1-|\lambda_{U_j}|^2) & \cdots & (1-|\lambda_{V_N}|^2)(1-|\lambda_{U_j}|^2) \\
(1-|\lambda_{V_1}|^2)(1-|\lambda_{U_j}|^2) & \cdots & (1-|\lambda_{V_1}|^2)(1-|\lambda_{U_j}|^2) \\
(1-|\lambda_{V_2}|^2)(1-|\lambda_{U_j}|^2) & \cdots & (1-|\lambda_{V_2}|^2)(1-|\lambda_{U_j}|^2) \\
\end{bmatrix}$$

where $\lambda_{V_k}$ and $\lambda_{U_l}$ for $k,l \in \{1, \ldots, N\}$ are the eigenvalues of $V$ and $U$ respectively.

The matrix $\Lambda_j$ essentially represents the weight of each entry in the $N \times N$ matrix $b_j^H b_j$, where $\sum_{j=1}^{N} b_j^H b_j = B^H B = E_{\mathbf{v}}^H W^T W E_{\mathbf{v}}$. Note that $W E_{\mathbf{v}}$ is the projection of each row of $W$ onto the eigenspace of $V$. This is similar to the case [cite Memory Trace paper] where the authors obtained the projection of feedforward connection vector onto the eigenspace of recurrent connectivity. However, in this case, these projections doesn’t add up to 1 unless we constrain $W$ to be unitary in which case $B^H B = I$. However, such a constraint will not be beneficial as in essence, we are restricting the input signal to only be visible to the diagonal elements of the state matrix after transformation by $U$ and $V$. This have an attenuation effect on the signal entering the state due to the matrix product, which can affect the information negatively in long term. In the main text, we showed this effect of such constraint by proving a bound on $J_{tot}$ which shows the inability of such constraint to store past information.

Now using $\Lambda_j$, we can find a form of $J_{tot}$ which is a bit simpler,

$$J_{tot} = \frac{1}{\varepsilon_1 \varepsilon_2} \sum_{j=1}^{N} \text{Tr} \left( \Lambda_j \odot b_j^H b_j \right)$$
or, equivalently, we can represent it as the sum of all sums of weighted diagonal elements of $b^H b$,

$$J_{tot} = \frac{1}{\epsilon_1 \epsilon_2} \sum_{j=1}^{N} \sum_{k=1}^{N} \frac{(1 - |\lambda_{V_k}|^2)(1 - |\lambda_{U_j}|^2)}{1 - |\lambda_{V_k}|^2 |\lambda_{U_j}|^2} |b_{jk}|^2$$ \hspace{1cm} (53)

where $b_{jk}$ is the corresponding element of matrix $B$.

The form in 53 now can be explored analytically, especially due to the fact that the function of eigenvalues in 53, $\frac{(1 - |\lambda_{V_k}|^2)(1 - |\lambda_{U_j}|^2)}{1 - |\lambda_{V_k}|^2 |\lambda_{U_j}|^2}$ can easily seen to be $\leq 1$, if $|\lambda_{V_k}|, |\lambda_{U_j}| \leq 1$, which informally implies that the weights for the elements $|b_{jk}|^2$ are $\leq 1$ which can be used to create bounds for $J_{tot}$ as done in the main text.

References

Mingyuan Bai, Boyan Zhang, and Junbin Gao. Tensorial neural networks and its application in longitudinal network data analysis. In Derong Liu, Shengli Xie, Yuanqing Li, Dongbin Zhao, and El-Sayed M. El-Alfy, editors, Neural Information Processing, pages 562–571, Cham, 2017. Springer International Publishing.

Pierre Baldi and Roman Vershynin. On neuronal capacity. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems 31, pages 7729–7738. Curran Associates, Inc., 2018.

Pierre Baldi and Roman Vershynin. The capacity of feedforward neural networks, 2019.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261, 2018.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence, 35(8):1798–1828, 2013.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37, pages 1613–1622, 2015.

Carlos Busso, Murtaza Bulut, Chi-Chun Lee, Abe Kazemzadeh, Emily Mower, Samuel Kim, Jeannette N. Chang, Sungbok Lee, and Shrikanth S. Narayanan. Iemocap: interactive emotional dyadic motion capture database. Language Resources and Evaluation, 42(4):335–359, 2008. ISSN 1574020X, 15728412.

Rohitash Chandra, Konark Jain, Ratneel V. Deo, and Sally Cripps. Langevin-gradient parallel tempering for bayesian neural learning. Neurocomputing, 359:315 – 326, 2019.
Rohitash Chandra, Konark Jain, Arpit Kapoor, and Ashray Aman. Surrogate-assisted parallel tempering for bayesian neural learning. *Engineering Applications of Artificial Intelligence*, 94:103700, 2020.

Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. Antisymmetricrnn: A dynamical system view on recurrent neural networks, 2019.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches, 2014.

Sreerupa Das, C Lee Giles, and Guo-Zheng Sun. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, page 14, 1992.

Kien Do, Truyen Tran, and Svetha Venkatesh. Learning deep matrix representations, 2017.

Surya Ganguli, Dongsung Huh, and Haim Sompolinsky. Memory traces in dynamical systems. *Proceedings of the National Academy of Sciences*, 105(48):18970–18975, 2008. ISSN 0027-8424. doi: 10.1073/pnas.0804451105.

Junbin Gao, Yi Guo, and Zhiyong Wang. Matrix neural networks, 2016.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*, 2017.

A.K. Gupta and D.K. Nagar. *Matrix Variate Distributions*. Monographs and Surveys in Pure and Applied Mathematics. Taylor & Francis, 1999. ISBN 9781584880462. URL https://books.google.co.in/books?id=PQ0YN7P1loC

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Zhiwu Huang and Luc Van Gool. A riemannian network for spd matrix learning, 2016.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems 31*, pages 8571–8580. Curran Associates, Inc., 2018.

Amir Hosein Khasahmadi, Kaveh Hassani, Parsa Moradi, Leo Lee, and Quaid Morris. Memory-based graph networks. *arXiv preprint arXiv:2002.09518*, 2020.

Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models, 2019.

42
Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

Solomon Kullback and Richard A Leibler. On information and sufficiency. The annals of mathematical statistics, 22(1):79–86, 1951.

Hung Le, Truyen Tran, and Svetha Venkatesh. Neural stored-program memory. arXiv preprint arXiv:1906.08862, 2019.

James Martens. New insights and perspectives on the natural gradient method, 2020.

Tom M Mitchell. The need for biases in learning generalizations. Department of Computer Science, Laboratory for Computer Science Research . . . , 1980.

Hans-Georg Muller, Ulrich Stadtmuller, et al. Estimation of heteroscedasticity in regression analysis. The Annals of Statistics, 15(2):610–625, 1987.

Tu Nguyen, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Tensor-variate restricted boltzmann machines, 2015.

C. W. Omlin, K. K. Thornber, and C. L. Giles. Fuzzy finite state automata can be deterministically encoded into recurrent neural networks. IEEE Trans. Fuzzy Syst., 6:76–89, 1998.

Christian W. Omlin and C. Lee Giles. Training second-order recurrent neural networks using hints. In Proceedings of the Ninth International Conference on Machine Learning, pages 363–368. Morgan Kaufmann, 1992.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. volume 28 of Proceedings of Machine Learning Research, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

Trang Pham, Truyen Tran, and Svetha Venkatesh. Graph memory networks for molecular activity prediction, 2018.

F. Ringeval, A. Sonderegger, J. Sauer, and D. Lalanne. Introducing the recola multimodal corpus of remote collaborative and affective interactions. In 2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG), pages 1–8, 2013.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In International conference on machine learning, pages 1842–1850, 2016.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. Ordered neurons: Integrating tree structures into recurrent neural networks, 2018.

Rohollah Soltani and Hui Jiang. Higher order recurrent neural networks, 2016.
Jiahao Su, Jingling Li, Bobby Bhattacharjee, and Furong Huang. Tensorial neural networks: Generalization of neural networks and application to model compression, 2018.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pages 3104–3112, 2014.

Jalil Taghia, Maria Bánkestad, Fredrik Lindsten, and Thomas B Schön. Constructing the matrix multilayer perceptron and its application to the vae. arXiv preprint arXiv:1902.01182, 2019.

Damien Teney, Lingqiao Liu, and Anton van den Hengel. Graph-structured representations for visual question answering, 2016.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Peter Tino. Fisher memory of linear wigner echo state networks. In ESANN, 2017.

D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj. Temporal attention-augmented bilinear network for financial time-series data analysis. IEEE Transactions on Neural Networks and Learning Systems, 30(5):1407–1418, 2019. doi: 10.1109/TNNLS.2018.2869225.

Roman Vershynin. Memory capacity of neural networks with threshold and relu activations, 2020.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. arXiv preprint arXiv:1410.3916, 2014.

Olivia L. White, Daniel D. Lee, and Haim Sompolinsky. Short-term memory in orthogonal neural networks. Physical Review Letters, 92(14), Apr 2004. ISSN 1079-7114. doi: 10.1103/physrevlett.92.148102.

Max A Woodbury. Inverting modified matrices. Memorandum report, 42(106):336, 1950.

Y. Zhang, R. Chandra, and J. Gao. Cyclone track prediction with matrix neural networks. In 2018 International Joint Conference on Neural Networks (IJCNN), pages 1–8, 2018. doi: 10.1109/IJCNN.2018.8489077.