Ph.D. Dissertation

Complex-valued Recurrent Neural Networks with Memory Units

기억 뉴런으로 구성된 복소 회귀 신경망

February 2021

Interdisciplinary Program in Neuroscience
College of Natural Science
Seoul National University

Bon-Woong Ku

Complex-valued Recurrent Neural Networks with Memory Units

기억 뉴런으로 구성된 복소 회귀 신경망

지도 교수 장 병 탁

이 논문을 이학박사 학위논문으로 제출함
2021 년 1 월

서울대학교 대학원
협동과정 뇌과학 전공
구 본 웅

구본웅의 이학박사 학위논문을 인준함
2021 년 1 월

위 원 장 _____ 장 대 익

부위원장 _____ 장 병 탁

위 원 _____ 최 세 영

위 원 _____ 이 인 아

위 원 _____ 김 병 희

Abstract

Recurrent neural networks (RNNs) are deep network models created to deal with sequence data. It is challenging to train RNNs to learn long-term dependencies within sequences.

Among challenges for it, the vanishing and exploding gradient problems have long been a major stumbling block to learning long-term dependencies. In the past few years, orthogonal and unitary RNNs (ouRNNs) with the unitary recurrent weight matrix were shown to resolve the problems since the eigenvalues of the matrix only have unit absolute value.

After that, a study proposed the non-normal RNN (nnRNN) where the recurrent matrix is parametrized using Schur decomposition. This parameterization endows the network's dynamics with higher expressivity. At the same time, the parameterization makes it simple to impose the unit absolute value constraint on the eigenvalues as with ouRNNs, thus addressing the vanishing and exploding gradient problems.

Here, we extend this model to the complex-valued nnRNN with memory units (mcnnRNN). The hidden state and parameters are set to have complex values. Memory units are built in the network using a novel method, which is different from that for the leaky integrator, a conventional memory unit. The potential of the architecture is investigated by testing it on several tasks involving long input sequences. The mcnnRNN achieves superior or comparable performance to the nnRNN as well as the long short-term

memory (LSTM) network. The results suggest our strategy is effective in improving the capability to learn long-term dependencies in sequential data.

In addition, we examine other characteristics of the mcnnRNN. First, we find the architecture is more robust to noise than the nnRNN. Second, each eigenvalue of the complex-valued recurrent weight matrix is not necessarily conjugate to another in contrast with real-valued one. We demonstrate this is a characteristics that helps the mcnnRN work better than the nnRNN.

Keywords : recurrent neural network (RNN); complex-valued neural network; memory unit; non-normal RNN (nnRNN); Schur decomposition; long-term dependency

Student Number : 2002-20617

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recurrent neural networks (RNNs) are deep network models that are well-suited to learning from sequence data. RNNs hold relevant information on its input streams in the hidden state until it is reset to the initial state. In other words, RNNs retain a short-term memory trace for its past inputs. Thanks to this property, RNNs are able to learn temporal structures (Elman, 1990) such as long-term dependencies in sequences.

In practice, however, it is not a straightforward task to train RNNs to learn long-term dependencies. Among challenges for it, the vanishing and exploding gradient problems (Hochreiter, 1991; Bengio et al., 1994) have long been a major stumbling block when working with the backpropagation-through-time (BPTT) algorithm. Several architectures were developed to tackle the problems. Gated units such as the long short-term memory (LSTM) unit (Hochreiter & Schmidhuber, 1997) and the gated recurrent unit (GRU)

(Cho et al., 2014) have been successful and gained popularity in coping with vanishing gradients.

Another line of studies on Elman-type RNNs (Elman, 1990) proposed orthogonal or unitary RNNs (ouRNNs) with the unitary recurrent matrix (Arjovski et al., 2016; Henaff et al., 2016; Wisdom et al, 2016; Hyland & Gunnar, 2017; Jing et al., 2017; Mhammedi et al., 2017; Helfrich et al., 2018; Lezcano-Casado & Martinez-Rubio, 2019). Vanishing and exploding gradients are known to arise when the hidden-to-hidden matrix's eigenvalues deviate from unit absolute value (Bengio et al., 1994). The unitary recurrent matrix, mathematically including real-valued orthogonal one, only has eigenvalues with unit absolute value, and then resolve the problems (Arjovski et al., 2016).

A study pointed out unitary recurrent matrices limit the expressivity of ouRNNs' hidden state dynamics (Kerg et al., 2019). To address this issue, the authors proposed the non-normal RNN (nnRNN) whose hidden-to-hidden matrix was parametrized using Schur decomposition (Section 2). This parametrization enables the recurrent weight matrix to represent non-normal matrices (Section 2.1), which are non-unitary. Non-normal recurrent matrices equip RNNs with higher expressivity of network dynamics than normal matrices (Section 2.1), which include all unitary matrices. Besides, it was argued that non-normal recurrent matrices may endow with larger short-term memory capacity (Ganguli et al., 2008) than normal ones. The parametrization using Schur decomposition also makes it easy to set the eigenvalues of the recurrent weight matrix to have unit absolute value like those of unitary matrices.

To take one step further, we considered two additional architectural features to incorporate into the nnRNN. First, the complex-valued hidden state and parameters might have advantage over real-valued ones in dealing with sequence data. In recent studies, complex-valued networks showed improved performance in music transcription (Trabelsi et al., 2018), speech spectrum prediction (Wisdom et al., 2016; Trabelsi et al., 2018) and speech enhancement (Choi et al., 2019). Second, memory units are known to help with learning long-term dependencies (Goodfellow et al., 2016). Leaky integrator units, for example, as conventional memory units, remember signals by integrating them with different time constants (Mozer, 1992; El Hihi and Bengio, 1996).

In this work, we investigate the capability of the mcnnRNN to learn long-term dependencies. The hidden state, recurrent weight matrix and input weights of the network are set to have complex values. To add memory units in the network, we introduce a new method customized to the nnRNN (Section 3.1). The network is evaluated on four tasks: the copy problem (Hochreiter & Schmidhuber, 1997), the adding task (Hochreiter & Schmidhuber, 1997), MNIST pixel sequence classification (Le et al., 2015) and character-level prediction on the Penn Treebank (PTB) corpus (Marcus et al., 1993) (Section 4.1). We find the mcnnRNN beats the nnRNN, which is real-valued and made up of memoryless units, as well as the LSTM model. In addition, we test how robust to noise the mcnnRNN is, and examine an aspect of the complex-valued recurrent matrix that contributes to the improved performance (Section 4.2).

Chapter 2

Related Works and Background

2.1. Orthogonoal and Unitary RNNs

The vanishing and exploding gradient problems have long made it intractable to train RNNs involving long input sequences. Arjovski et al., in their seminal work (Arjovski et al., 2016), showed empirically that unitary recurrent weight matrices are effective in overcoming the exploding and vanishing gradient problems. They also proved analytically that orthogonal recurrent weight matrices prevent exploding gradients on condition that the derivative of the activation function has unit absolute value at the origin.

After that, several ouRNNs were proposed (Henaff et al., 2016; Wisdom et al, 2016; Hyland & Gunnar, 2017; Jing et al., 2017; Mhammedi et al., 2017; Helfrich et al., 2018; Lezcano-Casado & Martinez-Rubio, 2019). The method Arjovski et al. (2016) used to parametrize the unitary hidden-to-hidden matrix

limited the expressiveness of the matrix and, therefore, the representational capacity for the network. Wisdom et al. (2016) expanded the expressiveness to include all unitary matrices by using a multiplicative update algorithm based on the Cayley transform. Jing et al. (2016) and Mhammedi et al. (2017) proposed parametrization methods for unitary and orthogonal matrices using Givens rotations and Householder reflections, respectively. Helfrich et al. (2018) introduced a simple update scheme for orthogonal matrices by parametrizing them with a skew-symmetric matrix through the Cayley transform.

2.2. Schur Decomposition and Normal and Non-normal Matrices

For both the nnRNN and mcnnRNN, parametrization of the recurrent weight matrix is based on Schur decomposition. To introduce Schur decomposition, we state a lemma which was adapted from Strang (2006).

Lemma 1 Any matrix $\mathbf{W} \in \mathbb{C}^{N\times N}$ can be expressed as $\mathbf{W} = \mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^{*}$ with unitary $\mathbf{P} \in \mathbb{C}^{N\times N}$ and triangular $\widehat{\mathbf{W}} \in \mathbb{C}^{N\times N}$ where $*$ denotes the conjugate transpose. The eigenvalues of $\mathbf{W}$ appear along the diagonal of $\widehat{\mathbf{W}}$.

$\mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^{*}$ is the Schur decomposition of $\mathbf{W}$. The recurrent matrix parameterized using Schur decompostion is able to represent both unitary and non-unitary matrices since Schur decomposition is applicable to any $N \times N$ matrix. This parametrization also makes it simple to set the matrix's eigenvalues to have unit absolute value as they appear on the diagonal of $\widehat{\mathbf{W}}$ making them directly accessible.

If a matrix has all its eigenvalues unitary with each other, it is called normal, or otherwise non-normal. As a necessary and sufficient condition, if $\widehat{\mathbf{W}}$ in Lemma 1 has only 0's on the off-diagonal part (i.e. diagonal), $\mathbf{W}$ is normal (Trefethen & Embree, 2005). In this case, $\mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^*$ is the eigen-decomposition of $\mathbf{W}$, a special case of Schur decomposition. Unitary matrices are normal since any unitary matrix has an eigen-decomposition (Strang, 2006). If $\widehat{\mathbf{W}}$ has any nonzero entry on its off-diagonal part, $\mathbf{W}$ is non-normal (Trefethen & Embree, 2005).

2.3. Non-normal RNN (nnRNN)

The description of Elman-type RNNs (Elman, 1990), where the nnRNN belongs, involves two equations:

$$\mathbf{h}_t = f(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}), \tag{2.1}$$

$$\mathbf{y}_t = \mathbf{V}\mathbf{h}_t + \mathbf{c} \tag{2.2}$$

with the hidden state $\mathbf{h}$ of dimension N, input $\mathbf{x}$ of dimension L, output $\mathbf{y}$ of dimension M and the activation function $f(\cdot)$. The parameters are as follows: $N \times N$ dimensional recurrent weight matrix $\mathbf{W}$, $N \times L$ dimensional input-to-hidden weight matrix $\mathbf{U}$, $M \times N$ dimensional hidden-to-output weight matrix $\mathbf{V}$, N dimensional bias vector in the hidden layer $\mathbf{b}$, and M dimentional bias vector for the output layer $\mathbf{c}$.

The nnRNN (Kerg et al. 2019) is comprised of Eq. (2.1)-(2.5). $\mathbf{W}$ in Eq. (2.1) is parameterized as

$$\mathbf{W} = \mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^{\mathbf{T}}, \tag{2.3}$$

$$\widehat{\mathbf{W}} = \widehat{\mathbf{W}}_{diag} + \widehat{\mathbf{W}}_{tr} \tag{2.4}$$

$$\widehat{\mathbf{W}}_{diag} = \begin{bmatrix} \mathbf{\Lambda}_1 & 0 & \cdots & 0 & 0 \\ 0 & \mathbf{\Lambda}_2 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \mathbf{\Lambda}_{N/2-1} & 0 \\ 0 & 0 & \cdots & 0 & \mathbf{\Lambda}_{N/2} \end{bmatrix}$$

$$\widehat{\mathbf{W}}_{tr} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ \tau_{2,1} & 0 & \cdots & 0 & 0 \\ \tau_{3,1} & \tau_{3,2} & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \tau_{N,1} & \tau_{N,2} & \cdots & \tau_{N,N-1} & 0 \end{bmatrix}$$

$$\mathbf{\Lambda}_j = \gamma_j \begin{bmatrix} \cos\theta_j & -\sin\theta_j \\ \sin\theta_j & \cos\theta_j \end{bmatrix}. \tag{2.5}$$

To avoid complex numbers in $\mathbf{W}$, Kerg et al. (2019) used real Schur decomposition instead of Schur decomposition in Lemma 1 for the parametrization. In the real Schur decomposition of $\mathbf{W}$ (Eq. (2.3)-(2.5)), matrix $\mathbf{P} \in \mathbb{R}^{N\times N}$ is orthogonal. $\widehat{\mathbf{W}}$ is represented as the sum of block diagonal matrix $\widehat{\mathbf{W}}_{diag}$ and strictly triangular matrix $\widehat{\mathbf{W}}_{tr}$ (Eq. (2.4)). $\widehat{\mathbf{W}}_{diag}$ contains 2×2 blocks $\mathbf{\Lambda}_j$'s along the diagonal. Each $\mathbf{\Lambda}_j$ consists of the real and imaginary parts, on the diagonal and off-diagonal entries, respectively, of a complex conjugate eigenvalue pair of $\mathbf{W}$ (Eq. (2.5)). The eigenvalues are reparametrized as $\gamma_j(\cos\theta_j \pm i\sin\theta_j)$ so that their absolute value is γ_j independent of θ_j. $\widehat{\mathbf{W}}_{tr}$ contains $\tau_{j,k}$'s on the lower triangular part and 0's everywhere else. If any $\tau_{j,k}$ has a nonzero value, the matrix is non-normal (Section 2.1). Non-normality is what makes the nnRNN more expressive than ouRNNs by providing the hidden state dynamics with stronger fluctuations.

In Kerg et al. (2019), each γ_j was encouraged to be close to 1 by a regularization term $\delta\|1-\boldsymbol{\gamma}\|_2^2$, with $\boldsymbol{\gamma} = [\gamma_1\ \gamma_2\ \ldots\ \gamma_{N/2}]^{\mathrm{T}}$ and δ as a Lagrange multiplier, added to the loss function. In our experiments for

comparison with the mcnnRNN, however, the absolute value γ_j of the eigenvalues are kept constantly 1 as in the mcnnRNN (Eq. (3.3)). This is essential to preventing vanishing and exploding gradients (Benjio et al., 1994; Arjovski et al., 2016).

For the activation function $f(\cdot)$, we use the identity function, exponential linear unit (ELU) and rectified linear unit (ReLU) while the modReLU (Arjovski et al. 2016) was adopted by Kerg et al. (2019). These functions are also selected as $g(\cdot)$ in Eq. (3.6) which is used for both the real and imaginary parts of the activation function in the mcnnRNN. As an important note, the derivative of $f(\cdot)$ at the origin $f'(0)$ as well as all the eigenvalues of $\mathbf{W}$ must have unit absolute value (i.e. $|f'(0)|=1$) to avoid vanishing and exploding gradients (Arjovski et al., 2016). This condition means that the state coefficient matrix of the linear approximation to Eq. (2.1) at the origin (Eq. (2.6)) must only have eigenvalues with unit absolute value.

$$\mathbf{h}_t \approx f'(0)(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \tag{2.6}$$

$$= \pm\mathbf{W}\mathbf{h}_{t-1} \pm \mathbf{U}\mathbf{x}_t \pm \mathbf{b}. \tag{2.7}$$

It is because, under the condition, the state coefficient matrix is $\pm\mathbf{W}$ (Eq. (2.6) & (2.7)) whose eigenvalues all have unit absolute value. The derivative at the origin of the identity function, ELU and ReLU have value 1 (i.e. $f'(0)=1$ and therefore $|f'(0)|=1$) as is often the case with common activation functions.

Chapter 3

Methodology

3.1. Complex-valued nnRNN with Memory Units (mcnnRNN)

The mcnnRNN is described by the following equations:

$$\mathbf{h}_{\mathrm{t}} = \mathbf{M}\mathbf{h}_{\mathrm{t}-1} + f\left(\left(\mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^{*} - \mathbf{M}\right)\mathbf{h}_{\mathrm{t}-1} + \mathbf{U}\mathbf{x}_{\mathrm{t}}\right), \tag{3.1}$$

$$\widehat{\mathbf{W}} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 & 0 \\ \tau_{2,1} & \lambda_2 & \cdots & 0 & 0 \\ \tau_{3,1} & \tau_{3,2} & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \lambda_{N-1} & 0 \\ \tau_{N,1} & \tau_{N,2} & \cdots & \tau_{N,N-1} & \lambda_N \end{bmatrix}, \tag{3.2}$$

$$\lambda_j = \cos\theta_j + i\sin\theta_j, \tag{3.3}$$

$$\mathbf{M} = \begin{bmatrix} m_1 & 0 & 0 & \cdots & 0 \\ 0 & m_2 & 0 & \cdots & 0 \\ 0 & 0 & m_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & m_N \end{bmatrix}, \tag{3.4}$$

$$\mathbf{y}_t = \mathbf{V}\begin{bmatrix}\mathrm{Re}(\mathbf{h}_t)\\ \mathrm{Im}(\mathbf{h}_t)\end{bmatrix} + \mathbf{c} \tag{3.5}$$

with input $\mathbf{x} \in \mathbb{R}^M$, hidden state $\mathbf{h} \in \mathbb{C}^N$ and output $\mathbf{y} \in \mathbb{R}^L$.

The activation function $f(\cdot)$ is defined as

$$f(z) = g(\alpha) + ig(\beta) \tag{3.6}$$

with $z = \alpha + i\beta$ $(\alpha,\ \beta \in \mathbb{R})$ and $g(\cdot) \in \mathbb{R}$. Activation functions of this kind were introduced in a study on complex-valued neural networks (Trabelsi et al., 2018). Depending on the tasks, different $g(\cdot)'$s are selected with the best results being reported: : the identity function for the copy task (Section 4.1.1), the ReLU for the adding problem (Section 4.1.2) and PTB language modeling (Section 4.1.3) and the ELU for the pixel-by-pixel MNIST task (Section 4.1.4). These functions are also used as the activation function of the nnRNN (Section 2.3) in our experiments. All of their derivatives have value 1 at the origin (i.e. $g'(0) = 1$), letting the same hold for $f(\cdot)$ (i.e. $f'(0)=1$) (proof in Appendix A).

The coefficient matrix $\mathbf{M} \in \mathbb{C}^{N\times N}$ for $\mathbf{h}$ is associated with the units' time constant. The complex-valued nnRNN with memoryless unit (cnnRNN) is mathematically equivalent to the mcnnRNN with constant $\mathbf{M} = \mathbf{0}$:

$$\mathbf{h}_t = f\left(\mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^*\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t\right), \tag{3.7}$$

$$\mathbf{y}_t = \mathbf{V}\begin{bmatrix}\mathrm{Re}(\mathbf{h}_t)\\ \mathrm{Im}(\mathbf{h}_t)\end{bmatrix} + \mathbf{c}. \qquad \text{(same as Eq. (3.5))}$$

The cnnRNN is tested and compared with other models in our experiments for reference.

$\mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^*$ in Eq. (3.1) takes the form of Schur decomposition (Lemma 1). That is, $\mathbf{P} \in \mathbb{C}^{N\times \mathrm{N}}$ is unitary and $\widehat{\mathbf{W}} \in \mathbb{C}^{N\times N}$ is triangular. $\widehat{\mathbf{W}}$ has diagonal entries λ_j's which have unit absolute value (Eq. (3.3)). Unlike the nnRNN,

the hidden-to-hidden matrix here is set to $\mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^* - \mathbf{M}$ so that the state coefficient matrix of the linear approximation to Eq. (3.1) at the origin is $\mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^*$ (Eq. (3.8)). As a caveat, therefore, each λ_j is not an eigenvalue of the recurrent matrix, but of the state coefficient matrix. This leads the eigenvalues of the state coefficient matrix to have unit absolute value, which is a requirement for addressing the vanishing and exploding gradient problems as mentioned in Section 2.3.

$$\begin{aligned}\mathbf{h}_\mathrm{t} &\approx \mathbf{M}\mathbf{h}_{\mathrm{t}-1} + f'(0)\left(\left(\mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^* - \mathbf{M}\right)\mathbf{h}_{\mathrm{t}-1} + \mathbf{U}\mathbf{x}_\mathrm{t}\right) \\ &= \mathbf{M}\mathbf{h}_{\mathrm{t}-1} + \left(\mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^* - \mathbf{M}\right)\mathbf{h}_{\mathrm{t}-1} + \mathbf{U}\mathbf{x}_\mathrm{t}. \\ &= \mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^*\mathbf{h}_{\mathrm{t}-1} + \mathbf{U}\mathbf{x}_\mathrm{t}. \end{aligned} \tag{3.8}$$

$\mathbf{U} \in \mathbb{C}^{N\times M}$ (Eq. (3.1)) and $\mathbf{V} \in \mathbb{R}^{L\times 2N}$ (Eq. (3.5)) are the input-to-hidden and hidden-to-output matrices, respectively. $\mathbf{c} \in \mathbb{C}^L$ (Eq. (3.5)) is the bias parameters in the output layer. The biases for the hidden units are omitted for the mcnnRNN and cnnRNN in contrast to the nnRNN.

3.2. Optimizing the Parameters

If loss function F_L is a real-valued function, the gradient of F_L with respect to a complex-valued matrix $\mathbf{Z} = \mathbf{X} + i\mathbf{Y}$ is given by

$$\frac{\partial F_L}{\partial \mathbf{Z}} = \frac{\partial F_L}{\partial \mathbf{X}} + i\frac{\partial F_L}{\partial \mathbf{Y}} \tag{3.9}$$

(Anemüller et al., 2003). With respect to a complex-valued vector, the gradient is given in the same way (Haykin, 2002, p.798). In the mcnnRNN and cnnRNN, the gradients of F_L with respect to $\mathbf{P}$, $\widehat{\mathbf{W}}$, $\mathbf{U}$ (Eq. (3.1)) and $\mathbf{V}$ (Eq. (3.5)) are evaluated using Eq. (3.9).

Two optimizers are employed in training the mcnnRNN, cnnRNN and the nnRNN. For unitary matrix $\mathbf{P}$, we adopt a multiplicative update scheme (Eq. (3.10) & (3.11)). A study in optimization (Tagare, 2011) presented a gradient descent curve along the Stiefel manifold of all $N \times N$ unitary complex-valued matrices. Wisdom et al. (2016) applied this result to training their unitary RNN. A new solution $\mathbf{P}^{(k+1)}$ on the curve at iteration $k+1$ is the product of the Cayley transform of $\mathrm{A}^{(\mathrm{k})}$ and the previous solution $\mathbf{P}^{(k)}$.

$$\mathbf{P}^{(k+1)} = (\mathbf{I} + \frac{\eta}{2}\mathbf{A}^{(k)})^{-1}(\mathbf{I} - \frac{\eta}{2}\mathbf{A}^{(k)})\mathbf{P}^{(k)}, \tag{3.10}$$

$$\mathbf{A}^{(k)} = \mathbf{G}^{(k)*}\mathbf{P}^{(k)} - \mathbf{P}^{(k)*}\mathbf{G}^{(k)}. \tag{3.11}$$

η is the learning rate for this algorithm, and $\mathbf{G}^{(k)}$ is the gradient matrix of the loss function with respect to $\mathbf{P}^{(k)}$. For all the other parameters than $\mathbf{P}$, the Adam optimization algorithm (Kingma & Ba, 2014) is chosen. In our experiments, the algorithm's parameters have Tensorflow's default values.

The initialization of trainable parameters in Eq. (3.1)-(3.5) is as follows. Unitary matrix $\mathbf{P}$ is initialized to the identity matrix. The initial value of θ_j's (Eq. (3.3)) are uniformly sampled from within (-90°, 90°), except in experiments on the copy problem (Section 4.1.1) for which from (-180°, 180°). $\tau_{j,k}$'s in $\widehat{\mathbf{W}}$ (Eq. (3.2)) are initially set to 0. Each diagonal element m_j of $\mathbf{M}$ (Eq. (3.4)) is initialized to $\mathrm{s}_{j,j}$ with $\mathbf{P}\widehat{\mathbf{W}}\mathbf{P}^* = [\mathrm{s}_{j,k}]$ so that all the weights of self-connections for hidden units are 0 initially. Importantly, initializing $\mathbf{M}$ in other way often leads to a numerical error while training the network. The real and imaginary part matrices of $\mathbf{U}$ are respectively initialized using Glorot uniform initialization except for the adding task (Section 4.1.2), for which only the real part matrix is initialized by Glorot

uniform initialization, and the imaginary part matrix to $\mathbf{0}$. $\mathbf{V}$ and $\mathbf{c}$ are intialized to $\mathbf{0}$

3.3. Model Regularization

We employ a regularization method for which noisy parameters of the network are needed, differently from Kerg et al. (2019) (Section 2.3). Murray and Edwards (1994) suggested multiplicative noise in the weights of the multilayer perceptrons was effective in improving generalization performance. Following this approach, we introduce multiplicative noise into the elements of $\mathbf{U}, \mathbf{V}$ and the off-diagonal part of $\widehat{\mathbf{W}}$ (i.e. $\tau_{j,k}{}'s$) (Eq. (3.1), (3.2) & (3.5)).

$$\omega = (1 + \xi)\omega_L \tag{3.12}$$

Let ω be any of those parameters. ω is separated into two parts: multiplicative noise $1 + \xi$ and trainable parameter ω_L (Eq. (3.12)). Random variable ξ follows a truncated Gaussian distribution with mean 0 and a standard deviation as a hyperparameter. The value of ξ changes every iteration during training, and remains 0 when the model is evaluated. We use this regularizer in PTB character-level language modeling (Section 4.1.4).

Chapter 4

Experiments and Results

4.1. Benchmark Tests

The performance of the mcnnRNN is evaluated on the copy task (Hochreiter & Schmidhuber, 1997), the adding problem (Hochreiter & Schmidhuber, 1997), pixel-by-pixel MNIST (Le et al. 2015) and PTB character-level language modeling (Marcus et al., 1993). These problems have been used as benchmarks in previous studies on RNNs (Martens & Sutskever, 2011; Graves et al., 2014; Arjovski et al., 2016; Henaff et al., 2016; Wisdom et al, 2016; Hyland & Gunnar, 2017; Jing et al., 2017; Mhammedi et al., 2017; Helfrich et al., 2018; Lezcano-Casado & Martinez-Rubio, 2019; Kerg et al., 2019). Through the copy and adding tasks, we evaluate the short-term memory capacity of models which is critical to learning long-term dependencies within sequence data. Using the pixel-by-pixel MNIST and

Table 4.1. The number of hidden units and the value of hyperparameters. N denotes the number of hidden units. For the mcnnRNN, cnnRNN and nnRNN, learning rate 2 is used to optimize $\mathbf{P}$ (Eq. (3.10) & (3.11)), and learning rate 1 to optimize the other parameters. In PTB language modeling, learning rates decay depending on changes in training loss.

Task	Model	N	Learning Rate 1	Learning Rate 2	Batch Size
Copy	mcnnRNN	64	$2x10^{-4}$	$1x10^{-8}$	100
	cnnRNN	64	$2x10^{-4}$	$1x10^{-8}$	100
	nnRNN	68	$2x10^{-4}$	$1x10^{-8}$	100
	LSTM	38	$1x10^{-3}$	-	50
Adding	mcnnRNN	64	$1x10^{-3}$	$2x10^{-12}$	100
	cnnRNN	64	$1x10^{-3}$	$1x10^{-10}$	100
	nnRNN	68	$5x10^{-4}$	$1x10^{-10}$	100
	LSTM	38	$1x10^{-4}$	-	50
MNIST	mcnnRNN	128	$5x10^{-4}$	$5x10^{-7}$	100
	cnnRNN	128	$5x10^{-4}$	$2x10^{-7}$	100
	nnRNN	130	$5x10^{-4}$	$5x10^{-7}$	100
Permuted MNIST	mcnnRNN	128	$5x10^{-4}$	$2x10^{-7}$	100
	cnnRNN	128	$5x10^{-4}$	$5x10^{-7}$	100
	nnRNN	130	$5x10^{-4}$	$2x10^{-7}$	100
PTB	mcnnRNN	890	$1x10^{-3}$	$1x10^{-12}$	100
	cnnRNN	890	$1x10^{-3}$	$1x10^{-12}$	100

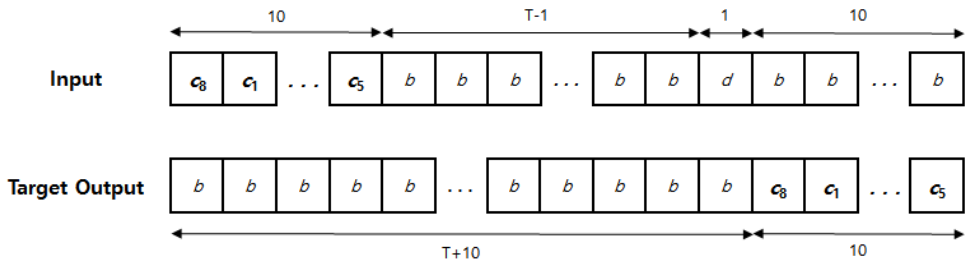

Figure 4.1. An illustration of the copy task. Symbols randomly sampled from $\{c_1, c_2, \ldots, c_8\}$ ($c_8, c_1, \ldots, c_5$) are given in the input sequence. After T+1 time steps, the model should start to recall the same symbols in the same order.

PTB character-level prediction problems, we test the ability of networks to capture long-term dependencies.

Though our main goal here is to assess the mcnnRNN in comparison to the nnRNN, other models such as the cnnRNN (Eq. (3.7) & (3.5)) and the LSTM network are also included for reference. The LSTM model is trained by Adam only. Gradient clipping is adopted for the LSTM, but not for the others. Hyperparameters are tuned using grid search (Table 4.1). All experiments are conducted using Tensorflow (Abadi et al., 2016).

4.1.1. Copy Task

This task tests the model's ability to recall the data seen many time steps before. The task is arranged following the same setup as in Arjovski et al. (2016) (Fig. 4.1). Input and output sequences involve 10 symbols: $\{c_1, c_2, \ldots, c_8, b, d\}$ which are encoded in one-hot fashion. The input is a T+20 long sequence of the symbols. The first 10 entries are sampled independently and uniformly at random from $\{c_1, c_2, \ldots, c_8\}$. These entries are to be recalled by

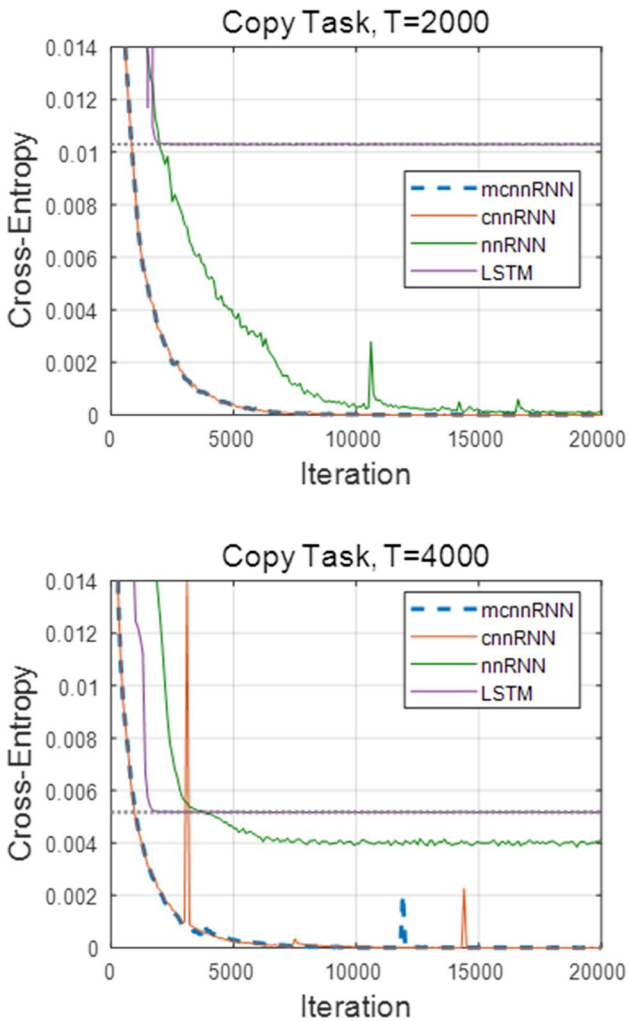

Figure 4.2. Training loss on the copy task for time lag T=2000 and 4000. The dotted line indicates baselines 0.010294 and 0.005172 which are approximately the expected losses of a random guess strategy that outputs symbol b for the first T+10 time steps and then symbols randomly sampled from $\{c_1, c_2, \ldots, c_8\}$ in the remaining.

the network. The next T-1 entries are *b* which can be thought of as the blank symbol. The next entry is *d*, the delimiter indicating that in the next time step the network should start to recall and output the initial 10 symbols in the input sequence. The remaining 10 symbols are *b*. The output is also a T+20 long sequence of the symbols. The target output consists of T+10 repeated entries of *b*, and then the first 10 symbols in the input sequence in the exactly same order. The loss function is the average cross-entropy of symbol predictions at each time step in the output sequence.

The baseline loss is determined by considering a random guess strategy (the dotted line in Fig. 4.2). This strategy outputs *b* for the first T+10 time steps, and then outputs symbols sampled independently and uniformly at random from $\{c_1, c_2, \ldots, c_8\}$ in the remaining. This gives an expected average cross-entropy of 10log8/(T+20) (Arjovski et al., 2016) which is chosen as the baseline loss value.

We consider input sequences of length T=2000 and 4000. The latter is, to the best of our knowledge, longer than in any previous studies. At each iteration, a new batch of sequences for training the model are generated randomly. The training process runs for 20000 iterations.

With $g(\cdot)$ (Eq. (3.6)) to be the identity function for this task (Section 3.1), $f(\cdot)$ is also the identity function. This leads $\mathbf{Mh}_{t-1}$ in the linear and nonlinear terms of Eq. (3.1) to cancel each other. In this case, the mcnnRNN becomes equivalent to the cnnRNN (Eq. (3.7)). Thus, they show nearly same loss curves.

The mcnnRNN and cnnRNN converge stably toward zero average cross-entropy in about 8000 iterations for both T=2000 and 4000 (Fig. 4.2). When T=2000, the nnRNN settles down to zero cross-entropy, but more slowly and

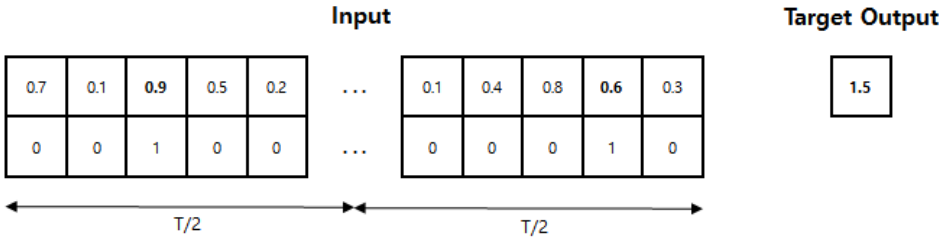

Figure 4.3. An illustration of the adding task. The model should produce the target output (1.5) which is the sum of two numbers (0.9 and 0.6) randomly sampled from within (0, 1) and marked by 1 in the first and second half (in time) of the input sequence, respectively.

with a rather noisy loss curve. For T=4000, it bypasses the baseline, but never converges to zero and remains near the baseline. The LSTM network gets stuck at the baseline in either case, performing the worst, which was also reported in previous studies (Arjovski et al., 2016; Henaff et al., 2016).

4.1.2. Adding Task

The network should remember two numbers marked in a long sequence and output the sum of them. The setup for this task followed that in Arjovski et al. (2016) (Fig. 4.3). Two sequences of length T comprise each input. The first one consists of numbers sampled independently and uniformly at random from [0, 1]. The second contains markers for the first one: two entries of 1 and the remaining entries of 0. The first 1 is within time steps [1, T/2] and the second within [T/2+1, T]. The target output is the sum of the two entries in the first sequence marked by the two 1's located in the second.

The model is trained to minimize the mean squared error (MSE) between the target and predicted outputs over each batch (Fig. 4.4). The length of input

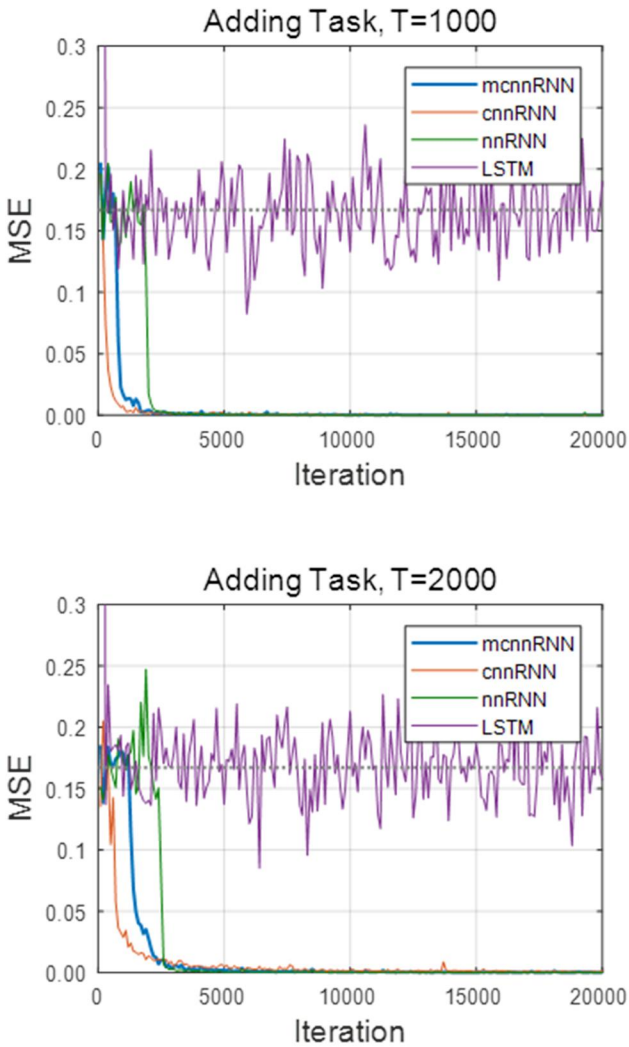

Figure 4.4. Training loss on the adding task for time lag T=1000 and 2000. The dotted line represents the baseline 0.167, which is approximately the expected loss of a strategy that always predicts 1 as the output regardless of the input.

sequences is chosen to be T=1000 and 2000, both of which are longer than in any other studies. At each iteration, new sequences as a training batch are generated randomly. The networks are trained for 20000 iterations.

The baseline MSE comes from a naïve strategy (the dotted line in Fig. 4.4). This strategy always predicts 1 as the output regardless of the input. This yields an expected MSE of 0.167 approximately. This value is the variance of the sum of two independent uniform distributions (Arjovski et al., 2016), and the baseline to beat for the task.

Both the mcnnRNN and nnRNN drop sharply far below the baseline MSE, and then settle down to zero, being comparable in training loss. The mcnnRNN, however, bypasses the baseline faster in about 1000 and 1500 iterations for T=1000 and 2000, respectively, whereas it takes the nnRNN about 2000 and 2500 iterations. The cnnRNN also achieves zero MSE, but with its loss curve rather noisy for T=2000. In contrast, the LSTM network never converge to a solution with zero error. The model remains fluctuating around the baseline, which is consistent with results from previous studies (Arjovski et al., 2016; Henaff et al., 2016).

4.1.3. Pixel-by-pixel MNIST

The network is fed a sequence of pixels from an MNIST image (LeCun et al., 1998), and are required to output the corresponding class label. Each MNIST example as a 28x28 image results in a pixel-by-pixel input sequence of length 784. We consider two versions of the task, the unpermuted and permuted pixel-by-pixel MNIST classification problems: the former with the input pixels in order, and the latter with permuted input pixels where the same randomly generated permutation matrix is applied to all the MNIST images.

Table 4.2. Test accuracy on the pixel-by-pixel MNIST classification tasks. N represents the number of hidden units.

Model	N	# parameters	MNIST Accuracy (%)	Permuted MNIST Accuracy (%)
mcnnRNN	128	≈ 27k	**98.6**	**96.1**
cnnRNN	128	≈ 27k	98.1	94.7
nnRNN	130	≈ 27k	97.8	93.9
LSTM (Wisdom et al., 2016)	128	≈ 68k	97.8	91.3
LSTM (Wisdom et al., 2016)	256	≈270k	98.2	91.7

This shuffling creates many longer-term dependencies across pixels of each input sequence than in the original pixel ordering where lots of local dependencies exist, and therefore making the problem harder. The average categorical cross-entropy is chosen as the loss function. We run the optimization algorithms for 70 epochs. We test the network on the validation dataset every epoch to select the parameters yielding the lowest mean cross-entropy. 10000 of the 60000 training examples are used as the validation set. The model with the selected parameters is evaluated on the test data set of 10000 samples (Table 4.2).

The mcnnRNN achieves the best test accuracy. The cnnRNN also performs over the nnRNN. The LSTM model falls short of the mcnnRNN in accuracy. Notably, in the unpermuted case, the mcnnRNN outperforms the

LSTM network with 256 hidden units, even with an order of magnitude fewer parameters and half the number of hidden units.

4.1.4. PTB Character-level Language Modeling

The model is trained on the PTB corpus (Marcus et al., 1993) to predict the next character at each step, given a sequence of characters. Input sequences have a length of 150 and 300 in two experiments, respectively. Each character is embedded as a one-hot vector. We train the network for 400 epochs on the training dataset (5017K characters), and select the model producing the lowest bits-per-character (BPC) on the validation dataset (393K characters). The selected model is evaluated on the test dataset (442K characters) (Table 4.3). During training, learning rates decay by a factor of 0.97 if average training loss over an epoch is larger than over the previous epoch. The PTB data is known to be prone to overfitting (Chang et al., 2017). Thus, we employ

Table 4.3. Test loss (BPC) on the PTB character-level prediction task. N denotes the number of hidden units.

Model	N	# parameters	Input length = 150 BPC	Input length = 300 BPC
mcnnRNN	890	≈1.32M	**1.31**	**1.32**
cnnRNN	890	≈1.32M	1.32	1.33
nnRNN (Kerg et al., 2019)	1024	≈1.32M	1.47	1.49
LSTM (Kerg et al., 2019)	1024	≈4.20M	1.37	-

the regularization scheme defined in Section 3.3. The standard deviation of ξ for it (Eq. (3.12)) is 0.3.

The mcnnRNN shows significantly lower test BPC values than the nnRNN (Table 4.3). The mcnnRNN also beats the LSTM model whereas the nnRNN in Kerg et al. (2019) fell short of the LSTM network in performance. The BPC values of the cnnRNNN are slightly lower than for the mcnnRNN.

4.2. Exploratory Experiments

4.2.1. Robustness to Noise

We test the mcnnRNN's robustness against noise as we often have to deal with noisy data in real-world problems. To do this, we add noise to MNIST images, and evaluate the accuracy of the model on the unpermuted and permuted MNIST tasks (Fig. 4.5). The noise follows a uniform distribution with lower bound 0 and an upper bound varied from 0 to 1.2 by 0.3. The larger the upper bound is, the stronger the noise is.

The mcnnRNN is more robust to the noisy data compared to the nnRNN (Fig. 4.5). As noise gets stronger, the accuracy of the mcnnRNN declines more slowly than that of the nnRNN. Especially, for unpermuted MNIST, the gap between the two models is quite significant. The cnnRNN also shows accuracy that decreases more slowly than for the nnRNN as noise increases, but faster than for the mcnnRNN.

4.2.2. Contribution from the Complex-valued Recurrent Weight Matrix

We investigate an aspect of the complex-valued hidden-to-hidden matrix that may help complex-valued RNNs work better than the nnRNN. For the real-

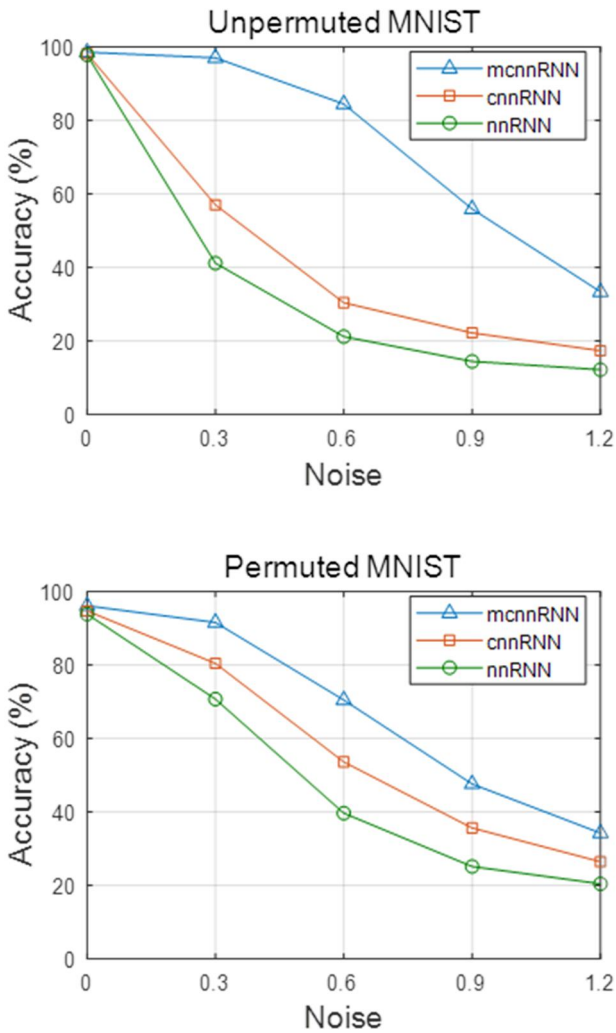

Figure 4.5. Change in accuracy on the pixel-by-pixel MNIST tasks with increasing noise. The horizontal axis denotes the upper bound of the uniform distribution from which noise is sampled.

valued recurrent matrix, every eigenvalue has its conjugate as an eigenvalue as well. Therefore, half the eigenvalues of the nnRNN are automatically determined by the other half. The complex-valued recurrent matrix, on the other hand, does not have this constraint on its eigenvalues. Thus, every eigenvalue of it can be placed anywhere in the complex plane. This leads to a higher diversity in the dynamics of complex-valued RNNs which may, in part, help the network work better.

To demonstrate this idea, we constrain the complex-valued recurrent matrix of the mcnnRNN or cnnRNN to have the conjugate of every eigenvalue as an eigenvalue as well like the real-valued hidden-to-hidden matrix of the nnRNN. The constrained models are trained and tested on the copy and MNIST tasks. The value of hyperparameters and the batch size for the models are the same as in Table 1.

The constrained models fall short of the unconstrained models in Section 4.1.1 and 4.1.3 in convergence rate or performance. For the copy problem, the constrained mcnnRNN shows a training curve that converges more slowly than without the constraint on the eigenvalues (Fig. 4.6). For the unpermuted and permuted pixel-by-pixel MNIST tasks, the accuracy of the constrained mcnnRNN and cnnRNN are lower than without the constraint (Table 4.4).

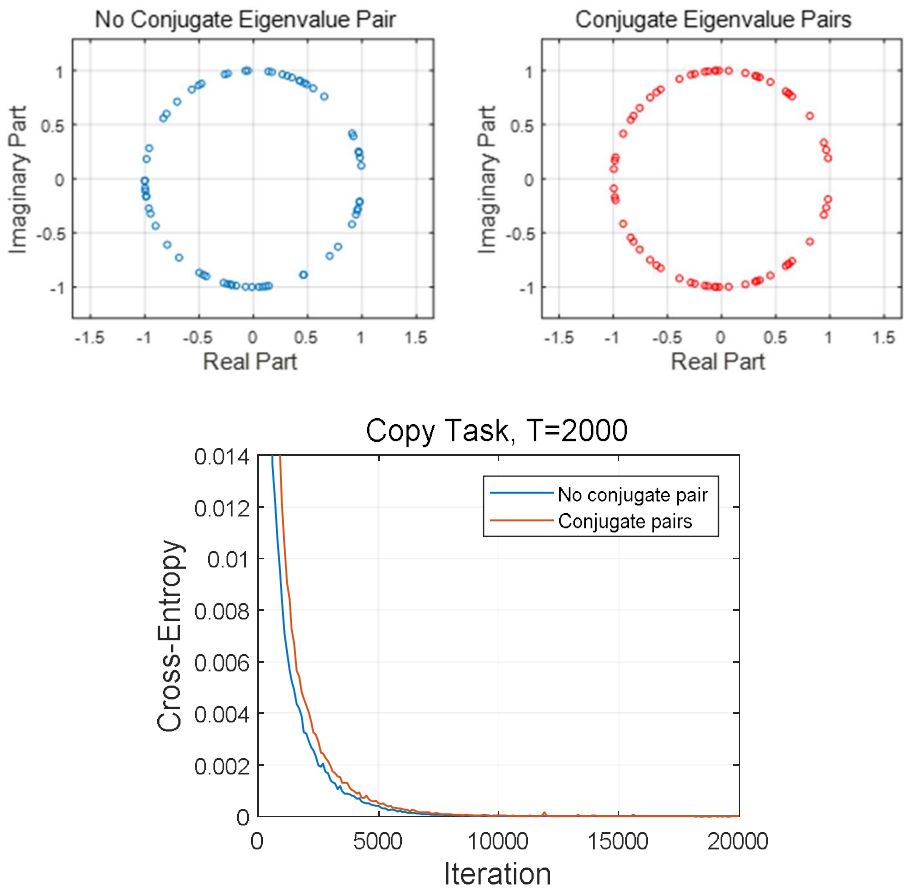

Figure 4.6. Comparison between two mcnnRNNs with and without conjugate eigenvalue pairs in the recurrent weight matrix. The plot in the lower panel shows the training curves of two mcnnRNNs trained on the copy task with T=2000. The plots in the upper panel show eigenvalues in the complex plane for the two models after training. The first mcnnRNN (blue) is the same as in Section 4.1.1. The recurrent matrix of the second one (red) is constrained to have the conjugate of every eigenvalue be an eigenvalue as well (upper right). The first model does not have this constraint on the eigenvalues (upper left).

Table 4.4. Comparison between accuracy on the MNIST tasks for models with and without conjugate eigenvalue pairs in the recurrent weight matrix. Accuracy values in the third column come from Section 4.1.3. The values of accuracy in the fourth column are obtained from models whose recurrent matrices are constrained to have the conjugate of every eigenvalue be an eigenvalue as well.

Task	Model	No conjugate Eigenvalue Pair Accuracy (%)	Conjugate Eigenvalue Pairs Accuracy (%)
MNIST	mcnnRNN	98.6	*98.5*
	cnnRNN	98.1	*98.0*
Permuted MNIST	mcnnRNN	96.1	*95.9*
	cnnRNN	94.7	*94.5*

Chapter 5

Conclusion and Discussion

We introduced a new method to build memory units in the nnRNN, also changing the hidden state and the input and recurrent weight matrices into complex ones. The method was customized to the nnRNN to keep the unit absolute value constraint on the eigenvalues of the state coefficient matrix for the linearized mcnnRNN. In terms of performance and robustness to noise, the complex-valued setting alone could augment the nnRNN. The memory units along with it added further improvement. The mcnnRNN also won out over the LSTM model in all of the benchmarks. The results suggest the mcnnRNN is effective in learning long-term dependencies.

The weight of self-connections for the mcnnRNN were initialized to 0 (Section 3.2). Otherwise, it often caused a numerical error while training the network. This is reminiscent of some studies on the autapse in neuroscience. An experimental work (Jiang et al., 2012) found that the autapse of fast-

spiking neurons in neocortical tissues of intractable epilepsy patients tend to show a different release property than in non-epileptic tissues. A computational study (Fan et al., 2018) suggested autapses can promote synchronization significantly which is believed to cause epileptic seizures. We speculate similar principles may, at least in part, underlie self-connections in both artificial and biological RNNs. It would be interesting to study more closely how the self-connections of the mcnnRNN influence on its activity and performance referring to researches in neuroscience.

Complex-valued neural networks have been studied in the past (Hirose, 2013), but their success and adoption were limited (Arjovski et al., 2016). Elman-type RNNs with memory units also have not been widely used in the neural network community ever since they were proposed more than two decades ago (Mozer, 1992; El Hihi and Bengio, 1996). We hope our findings will be a step forward to change this. Applying complex-valued neural networks and Elman-type RNNs with memory units in real-world problems may help to draw attention to them, but is yet to be more investigated. In future work, we will study further to test the utility of the mcnnRNN in real-world applications.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2016) Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467.*

Anemüller, J., Sejnowski, T. J., & Makeig, S. (2003). Complex independent component analysis of frequency-domain electroencephalographic data. *Neural Networks*, *16*(9), 1311-1323.

Arjovsky, M., Shah, A., & Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *International Conference on Machine Learning* (pp. 1120-1128).

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157-166.

Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., Cui, X., Witbrock, M., Hasegawa-Johnson, M. & Huang, T. S. (2017). Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems* (pp. 77-87).

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Conference on Empirical Methods in Natural Language Processing* (pp. 1724-1734).

Choi, H. S., Kim, J. H., Huh, J., Kim, A., Ha, J. W., & Lee, K. (2018). Phase-Aware Speech Enhancement with Deep Complex U-Net. In *International Conference on Learning Representations*.

El Hihi, S., & Bengio, Y. (1996). Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems* (pp. 493-499).

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179-211.

Fan, H., Wang, Y., Wang, H., Lai, Y. C., & Wang, X. (2018). Autapses promote synchronization in neuronal networks. Scientific reports, 8(1), 1-13.

Ganguli, S., Huh, D., & Sompolinsky, H. (2008). Memory traces in dynamical systems. *Proceedings of the National Academy of Sciences*, *105*(48), 18970-18975.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Cambridge, MA: The MIT Press.

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Haykin, S. (2002). *Adaptive Filter Theory* (4th ed.). Upper Saddle River, NJ: Prentice Hall.

Helfrich, K., Willmott, D., & Ye, Q. (2018). Orthogonal Recurrent Neural Networks with Scaled Cayley Transform. In *International Conference on Machine Learning* (pp. 1969-1978).

Henaff, M., Szlam, A., & LeCun, Y. (2016). Recurrent orthogonal networks and long-memory tasks. In *International Conference on International Conference on Machine Learning* (pp. 2034-2042).

Hirose, A. (2013). *Complex-valued Neural Networks: Advances and Applications*. Hoboken, NJ: John Wiley & Sons.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. *Diploma, Technische Universität München, 91*(1).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735-1780.

Hyland, S. L., & Rätsch, G. (2017). Learning unitary operators with help from u(n). In *AAAI Conference on Artificial Intelligence* (pp. 2050-2058).

Jaeger, H., Lukoševičius, M., Popovici, D., & Siewert, U. (2007). Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, *20*(3), 335-352.

Jiang, M., Zhu, J., Liu, Y., Yang, M., Tian, C., Jiang, S., Wang, Y., Guo, H., Wang, K., & Shu, Y. (2012). Enhancement of asynchronous release from

fast-spiking interneuron in human and rat epileptic neocortex. PLoS Biology, 10(5), e1001324.

Jing, L., Shen, Y., Dubcek, T., Peurifoy, J., Skirlo, S., LeCun, Y., ... & Soljačić, M. (2017). Tunable efficient unitary neural networks (eunn) and their application to rnns. In *International Conference on Machine Learning* (pp. 1733-1741).

Kerg, G., Goyette, K., Touzel, M. P., Gidel, G., Vorontsov, E., Bengio, Y., & Lajoie, G. (2019). Non-normal Recurrent Neural Network (nnRNN): learning long time dependencies while improving expressivity with transient dynamics. In *Advances in Neural Information Processing Systems* (pp. 13591-13601).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Le, Q. V., Jaitly, N., & Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

Lezcano-Casado, M., & Martıínez-Rubio, D. (2019). Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group. In *International Conference on Machine Learning* (pp. 3794-3803).

Martens, J., & Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *International Conference on Machine Learning* (pp. 1033-1040).

Mhammedi, Z., Hellicar, A., Rahman, A., & Bailey, J. (2017). Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *International Conference on Machine Learning* (pp. 2401-2409).

Marcus, P. M., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics, 19*(2):313–330.

Mozer, M. C. (1992). Induction of multiscale temporal structure. In *Advances in Neural Information Processing Systems* (pp. 275-282).

Murray, A. F. & Edwards, P. J. (1994). Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Transactions on neural networks*, *5*(5), 792-802.

Salehinejad, H., Sankar, S., Barfett, J., Colak, E., & Valaee, S. (2017). Recent advances in recurrent neural networks. arXiv preprint arXiv:1801.01078.

Strang, G. (2006) *Linear Algebra and Its Applications*. Belmont, CA: Thomson, Brooks/Cole.

Tagare, H. D. (2011). Notes on optimization on stiefel manifolds. In *Technical report,* Yale University.

Trabelsi, C., Bilaniuk, O., Zhang, Y., Serdyuk, D., Subramanian, S., Santos, J. F., Mehri, S., Rostamzadeh, N., Bengio, Y., & Pal, C. J. (2018). Deep Complex Networks. In *International Conference on Learning Representations*.

Trefethen, L. N., & Embree, M. (2005). *Spectra and Pseudospectra: the Behavior of Nonnormal Matrices and Operators*. Princeton University Press.

Wisdom, S., Powers, T., Hershey, J., Le Roux, J., & Atlas, L. (2016). Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems* (pp. 4880-4888).

Appendix A

Proof that if $g'(0) = 1$, then $f'(0) = 1$ in Eq. (3.6)

Given $z = \alpha + i\beta$, α and β can be expressed in terms of z and its conjugate $\overline{z}$.

$$\alpha = \frac{z + \overline{z}}{2},$$

$$\beta = \frac{z - \overline{z}}{2i}.$$

Then, it is trivial to derive their partial derivatives with respect to z.

$$\frac{\partial \alpha}{\partial z} = \frac{1}{2'} \tag{A.1}$$

$$\frac{\partial \beta}{\partial z} = \frac{1}{2i}. \tag{A.2}$$

Based on the chain rule and Eq. (A.1) and (A.2), the derivative of $f(z)$ is expressed as follows.

$$\frac{df(z)}{dz} = \frac{\partial g(\alpha)}{\partial \alpha}\frac{\partial \alpha}{\partial z} + i\frac{\partial g(\beta)}{\partial \beta}\frac{\partial \beta}{\partial z}$$
$$= \frac{\partial g(\alpha)}{\partial \alpha}\frac{1}{2} + i\frac{\partial g(\beta)}{\partial \beta}\frac{1}{2i}$$
$$= \frac{1}{2}\left(\frac{\partial g(\alpha)}{\partial \alpha} + \frac{\partial g(\beta)}{\partial \beta}\right). \quad \text{(A.3)}$$

Using Eq. (A.3), $f'(0)$ is calculated given that $g'(0) = 1$.

$$f'(0) \coloneqq \frac{df(z)}{dz}\bigg|_{z=0} = \frac{1}{2}\left(\frac{\partial g(\alpha)}{\partial \alpha} + \frac{\partial g(\beta)}{\partial \beta}\right)\bigg|_{z=0}$$
$$= \frac{1}{2}\big(g'(0) + g'(0)\big)$$
$$= \frac{1}{2}(1+1) = 1. \quad \square$$

초록

회귀 신경망 (RNN) 은 서열 데이터를 다루도록 고안된 심층망 모형이다. 회귀 신경망에 서열 데이터의 장기 의존성을 학습시키는 것은 어려운 문제이다.

특히 경도의 소실 및 폭발 문제는 오랫동안 장기 의존성 학습의 주요 난관이었다. 유니터리 회귀 가중치 행렬은 절대값이 1 인 고유값들만을 가지기 때문에, 직교 회귀 신경망과 유니터리 회귀 신경망이 그 문제들을 해결할 수 있음을 보인 연구들이 지난 몇 년 동안 발표되었다.

이어서, 회귀 가중치 행렬을 슈어 분해를 이용하여 모수화한 비정규 회귀 신경망 (nnRNN) 이 제안되었다. 이 모수화는 회귀 신경망에서 더욱 다양한 동역학적 궤적이 나타나도록 해준다. 또한, 직교 회귀 신경망과 유니터리 회귀 신경망처럼 회귀 행렬에서 고유값들의 절대값을 1 로 설정하는 것을 쉽게끔 해주어, 경도의 소실 및 폭발 문제를 해결할 수 있다.

본 연구에서는 nnRNN 을 기억 뉴런으로 구성된 복소 비정규 회귀 신경망 (mcnnRNN) 으로 확장한다. 은닉 상태와 모수들을 복소수로 설정하고, 기억 뉴런을 기존의 기억 뉴런인 누수 적분기와는 다른 새로운 방법으로 추가한다. 이 모형을 시험하기 위해, 긴 입력 서열을 사용하는 몇 가지 학습 과제를 mcnnRNN 이 수행하도록 한다. mcnnRNN 은 nnRNN 과 LSTM 모형보다 더 나은 성능을 보인다. 실험 결과들은 본 연구에서 제안한 방법이 장기 의존성 학습 능력을 향상하는 데 효과적임을 시사한다.

mcnnRNN 의 다른 특성들도 살펴본다. 첫째, 이 모형이 nnRNN 보다 잡음에 강하다는 것을 보인다. 둘째, 복소 회귀 행렬은 실수 회귀 행렬과는 달리 각 고유값이 반드시 다른 고유값의 켤레 복소수는 아닌데, 이것은 mcnnRNN 이 nnRNN 보다 더 나은 결과를 나타내는 데 도움이 됨을 보인다.

주요어 : 회귀 신경망; 복소 신경망; 기억 뉴런; 비정규 회귀 신경망; 슈어 분해, 장기 의존성
학번 : 2002-20617