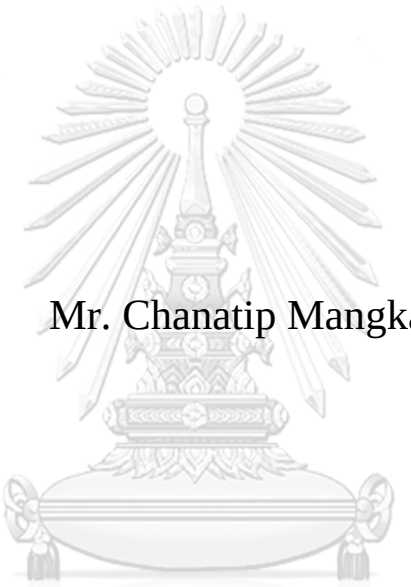
TOWARDS EXPLAINABLE SENTIMENT ANALYSIS FOR WRITTEN REVIEWS VIA QUANTUM TENSOR NETWORK STATES

Mr. Chanatip Mangkang



A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Physics
Department of Physics
FACULTY OF SCIENCE
Chulalongkorn University
Academic Year 2021

การวิเคราะห์ความรู้สึกของงานเขียนที่อธิบายได้ผ่านสถานะเครือข่ายควอนตัม

นายชนาธิป มั่งคั่ง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาฟิสิกส์ ภาควิชาฟิสิกส์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2564

Thesis Title	TOWARDS EXPLAINABLE SENTIMENT ANALYSIS FOR WRITTEN REVIEWS VIA QUANTUM TENSOR NETWORK STATES
By	Mr. Chanatip Mangkang
Field of Study	Physics
Thesis Advisor	Dr. THIPARAT CHOTIBUT

Accepted by the FACULTY OF SCIENCE, Chulalongkorn University in Partial Fulfillment of the Requirement for the Master of Science

..... Dean of the FACULTY OF SCIENCE
(Professor Dr. POLKIT SANGVANICH)

THESIS COMMITTEE

..... Chairman
(Associate Professor Dr. UDOMSILP PINSOOK)

..... Thesis Advisor
(Dr. THIPARAT CHOTIBUT)

..... Examiner
(Associate Professor Dr. SURACHATE LIMKUMNERD)

..... External Examiner
(Assistant Professor Dr. Sujint Suwanna)

ชนาธิป มั่งคั่ง : การวิเคราะห์ความรู้สึกของงานเขียนที่อธิบายได้ผ่านสถานะเครือข่ายควอนตัม. (TOWARDS EXPLAINABLE SENTIMENT ANALYSIS FOR WRITTEN REVIEWS VIA QUANTUM TENSOR NETWORK STATES) อ.ที่ปรึกษาหลัก : อ. ดร.ธิปรัชต์ โชติบุตร

โครงข่ายประสาทเทียมแบบวนซ้ำ (Recurrent Neural Network: RNN) เป็นการเรียนรู้ของเครื่องประเภทหนึ่งซึ่งมีความสามารถในการแก้ปัญหาจำพวกการประมวลผลภาษาธรรมชาติได้เป็นอย่างดี แต่การทำความเข้าใจพฤติกรรมของมันในทางทฤษฎีนั้นเป็นเรื่องที่เป็นไปได้ยากเนื่องจากการคำนวณภายในที่มีความซับซ้อน ในงานนี้เราทำการศึกษา RNN ประเภทหนึ่งที่มีชื่อว่า Recurrent Arithmetic Circuit หรือ RAC ซึ่งสามารถแปลงเป็น Matrix Product State (MPS) ที่ถูกใช้อย่างแพร่หลายในควอนตัมฟิสิกส์ได้ สิ่งนี้ช่วยให้เราสามารถคำนวณ Entanglement Entropy ของ MPS ซึ่งสามารถใช้อธิบายการส่งผ่านข้อมูลของโมเดลเพื่ออธิบายพฤติกรรมความถูกต้องที่เกิดขึ้นในการประมวลผลภาษาธรรมชาติ เราพบว่า Entanglement Entropy นั้นจะอิ่มตัวเมื่อความถูกต้องอิ่มตัวในกรณีของ Word Embedding มีค่าคงที่ ในกรณีที่ Word Embedding นั้นไม่ได้ถูกตั้งให้มีค่าคงที่ Entanglement Entropy นั้นมีค่าที่ลดลงเรื่อย ๆ เมื่อ Word Embedding มีความสามารถที่เพิ่มขึ้นโดยวัดจาก Cosine Similarity งานของเราช่วยให้การเรียนรู้ของเครื่องประเภท RNN สามารถถูกอธิบายได้มากยิ่งขึ้น

สาขาวิชา ฟิสิกส์ ลายมือชื่อนิสิต

.....

ปีการศึกษา 2564 ลายมือชื่อ อ.ที่ปรึกษาหลัก

.....

6270023923 : MAJOR PHYSICS
KEYWOR
D:

Chanatip Mangkang : TOWARDS EXPLAINABLE SENTIMENT ANALYSIS FOR WRITTEN REVIEWS VIA QUANTUM TENSOR NETWORK STATES. Advisor: Dr. THIPARAT CHOTIBUT

Recurrent Neural Networks (RNNs) have shown an incredible performance in supervised machine learning tasks such as Natural Language Processing (NLP). However, theoretical understanding of RNNs' performances in NLP are still limited due to intrinsically complex non-linear computations of RNNs. This thesis explores a class of RNNs called Recurrent Arithmetic Circuits (RACs), possessing a dual mathematical representation as a Matrix Product State (MPS) widely used in many-body quantum physics. The duality allows us to compute the entanglement entropy of an MPS, which can be used as a proxy for information propagation in the dual neural networks, to phenomenologically explain the RNNs-based model prediction accuracy's behaviors in NLP. We found that the entanglement entropy saturates when the accuracy saturates in the fixed word embedding case. The unfixed word embedding experiments also reveal that the entanglement entropy of the RACs is decaying as the word embedding becomes more meaningful, as reflected by the behaviors of cosine similarity between word embeddings. This thesis sheds light on more transparent and explainable behaviors of RNNs-based machine learning in NLP, using tools from many-body quantum physics.

Field of Study:	Physics	Student's Signature .....
Academic Year:	2021	Advisor's Signature .....

ACKNOWLEDGEMENTS

First of all, I would like to thank my thesis advisor, Professor Thiparat Chotibut, and Dr. Jirawat Tangpanitanon for giving me the opportunity to work on this project. There were times when I got stuck in coming up with research ideas and in making progress; however, they have always been providing helpful advice, ideas, and resource to overcome the obstacles. This project would not be successful without their continuous supports. I also acknowledge National e-Science Infrastructure Consortium, Chulalongkorn University cluster for providing us with computational resources.

Lastly, I also thank my friends and my family who always cheer me up and encourage me throughout the project.

Chanatip Mangkang

TABLE OF CONTENTS

Page

1. Introduction

Machine Learning (ML) has shown incredible performances in highly sophisticated computational tasks, ranging from defeating humans in the game of Go, Dota2, and in image recognition [1], for example. Despite the tremendous successes of ML, it is extremely difficult to understand and analyze ML behaviors [2]. From practical point of views, ML is typically treated as a black box computational model to perform a specific task. The more complex the task is, the more resources and the more complexity of the model are required. More complex models come at a cost of the requirement to tune a gigantic set of parameters to achieve the best model performance. It is difficult to determine the appropriate parameters for the model if we have a little understanding of the model's behaviors. A typical solution is to start with a gigantic model size, perhaps with many more parameters than one actually needs to fit the data.

In many-body quantum physics, many useful models have been extensively scrutinized, resulting in a comprehensive understanding of models' behaviors. One unique feature of many-body systems is that the Hilbert space of a many-body quantum state typically grows exponentially large with a system size. For the analysis of the model to be computationally tractable, several approximation techniques have been developed to efficiently and compactly represent a quantum state, compressing an exponential resource requirement to only a polynomial requirement. Examples of such compact representations of many-body quantum states include matrix product state (MPS) [3], projected entangled-pair states (PEPS) [4], and multiscale entanglement renormalization ansatz (MERA) [5].

To exploit the explainable aspects of quantum models and their useful compact representations, applying many-body quantum techniques to ML can give more insights into the behaviors of ML [6]. For example, Recurrent Arithmetic Circuits (RACs) are a type of Recurrent Neural Networks (RNNs) that can be mapped to an MPS in many-body quantum physics [7]. RNNs have a recurrent structure, making them special and suitable for tasks involving long-range memory such as time-series

or sequential predictions. RNNs have tremendous successes in Natural Language Processing (NLP) tasks, such as sentiment analysis and text generation. In this work, we will scrutinize RACs' behaviors in sentiment analysis, a computational task that classifies given sentences into categories, positive and negative sentiments in this work. We borrow the tools used to measure long-range correlations (entanglement entropy) in MPS to show that the information propagation in RACs saturates around the point where the model accuracy saturates. For the first time, our work reveals that a *minimal* model size which achieves highest prediction accuracy arises from the saturation of long-range information propagation in the model.

Note that the aforementioned result on the minimal model size disregards the contributions of correlations from other layers in the RACs network, particularly from the word embedding layer. Thus, we also analyze the role of the embedding layer on model prediction accuracy. As will be seen in this work, when the embedding layer is trainable, as the model size increases, the entanglement entropy increases before dropping down to saturate. As the entanglement entropy drops, we show that the word embedding becomes more meaningful semantically. This behavior implies that the larger the model size RACs possess, the more meaningful the embedding layer becomes, while the importance of information propagation in RACs decreases.

This thesis is organized as follows. First, we begin by providing a comprehensive background on RNNs and word embedding in sections 2.1-2.3. The mathematical equivalence between RACs and MPS as well as the discussion on the entanglement entropy in MPS are provided in sections 2.4-2.5. Then, in section 3, we show the RACs' performance and their information propagation as one increases the model size in a sentiment analysis task. The behaviors of the word embedding and of the entanglement entropy are also discussed. Finally, we conclude and discuss the future improvements of the model in section 4.

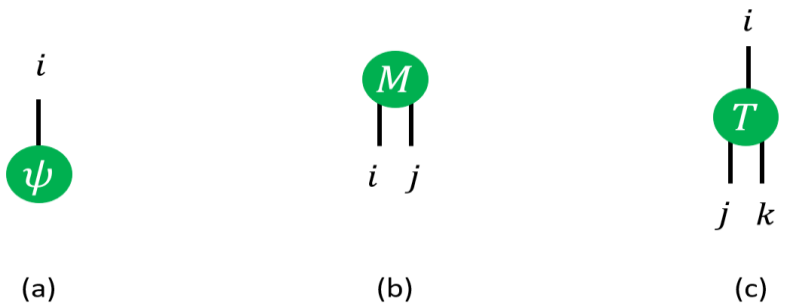
2. Background

Since language can be viewed as a sequential phenomenon, one common approach to model language (statistical language modeling) is to encode a meaningful sequence of words into a conditional probability to predict the n^{th} word given $n-1$ previous words $P(w_n|w_{1:n-1})$, where $w_{1:n-1}$ is the word sequence $(w_1, w_2, \dots, w_{n-1})$. However, the model built on this occurrence probability can be computationally impractical. To see why this problem could happen, assume that we have only 100 words in a dataset, and they can occur with the same frequency. In this case, the probability of a sequence of n words $(w_1, w_2, \dots, w_n)$ appearing in a data set is 100^{-n}, which is practically negligible. Even though a probabilistic model that predicts the next word given a shorter sequence of length k, say $P(w_n|w_{n-k:n-1})$, called a k-gram model, can be proposed, it is still hard to find a suitable k, while maintaining both exponential decay in probability and the accuracy of a model. For instance, assume that a model uses $k = n - 4$, which means we use four previous words for prediction. The model can give a wrong answer for a sentence like "This special dish that you prepared for me …", which may give "are," but the correct answer is "is," which requires a 6–8 gram model to achieve the correct answer. We could put or construct a model with more parameters in order to capture a long-range sequence while maintaining accuracy, but this requires more computational resources, which means more money to spend. We can achieve high accuracy with recurrent neural networks (RNNs) while only using a small amount of resources.

RNNs are a type of machine learning specialized in performing sequential tasks. This work will focus on many-body physics-inspired RNNs called RACs, consisting of a linear operation which is a tensor contraction. We will show that RACs can be mapped to an MPS, a 1D chain of tensors. However, before diving into the details, let us begin with introducing the terminologies used throughout this work to visually represent a tensor calculation as a picture, called a tensor network diagram.

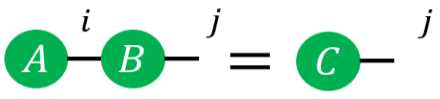
2.1 Tensor Network

When one talks about tensor contractions, indices manipulation is unavoidable. However, it is mentally more pleasing to represent tensor operations via pictorial thinking. This method is called the tensor diagram notation. Roger Penrose proposed the notation in early 1970's [8]. These methods have been exploited in many different domains of physics, computer science, and mathematics. To accommodate tensor manipulations as a LEGO-like concatenation, one visually represents a mathematical tensor by shapes such as a box or a circle, with zero or more output wires called legs, which point up and down corresponding to upper and lower indices, respectively.

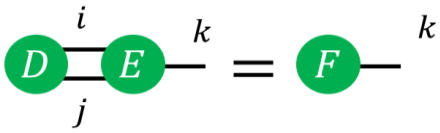


(a) (b) (c)

For example, diagram (a) above represents the tensor ψ^i with a single upper index (a vector), diagram (b) the tensor M_{ij} (a matrix), and diagram (c) the tensor T^i_{jk} (of rank 3). To do tensor contractions (summed over indices), one needs to connect two legs corresponding to those indices.



(d)



(e)

Diagram (d) represents the contraction of matrix B_i^j with vector A^i (a multiplying matrix with a vector), which results in another vector. The mathematical expression of diagram (d) is $B_i^j A^i = C^j$, where the leg labeled i is fully connected, and hence the corresponding index is summed over (Einstein's summation convention is assumed here). However, unlike in general relativity, we do not have to be concerned about whether the indices are upper or lower since. For the purpose of data science studied in this work, contravariant and covariant indices are the same as there are no requirement on the existence of a metric in our problem. So, we can raise and lower the indices freely (e.g., $A^i B_i^j = A^i B^{ji} = A_i B_{ji}$). In (e), the diagram is more complicated, contracting two indices between a matrix and an order-3 tensor. Diagram (e) mathematically expresses $D^{ij} E_{ij}^k = F^k$. Two or more tensor diagrams interact with each other will form a *tensor network.*

Now, we are armed with the knowledge of tensor networks that can be used to represent a cumbersome indices operation into a simpler pictorial representation. Next, we will introduce a general concept of RNNs and discuss the mapping between RACs and MPS.

2.2 Recurrent Neural Networks (RNNs)

As its name suggests, RNNs use recurrent computation to accumulate information from the past items in the sequence to predict the current item, as opposed to using the conditional probability prediction discussed earlier [9]. To keep things simple, we will stick to a sentiment analysis task that returns a binary output O for a given sequence of N words $(w_1, w_2, \dots, w_N)$. The output O in our case has two discrete values, which are 0 for negative sentiment and 1 for positive sentiment. At the core of calculation, RNN has the smallest unit called a cell, which calculates the hidden state at time step t. To calculate h at time t, the RNN cell receives two inputs: its previous hidden state $\boldsymbol{h}_{t-1}$ from the output of the previous cell and the embedded input $\boldsymbol{\phi}(w_t)$ at time step t. Thus, it is called recurrent. It is worth noting that, depending on the design, RNNs can have multiple hidden states. However, for the purposes of this paper, we will only consider one hidden state h because it can be

mapped to the tensor form, which will be discussed later. An example of a model that contains more than one state is LSTM [10].

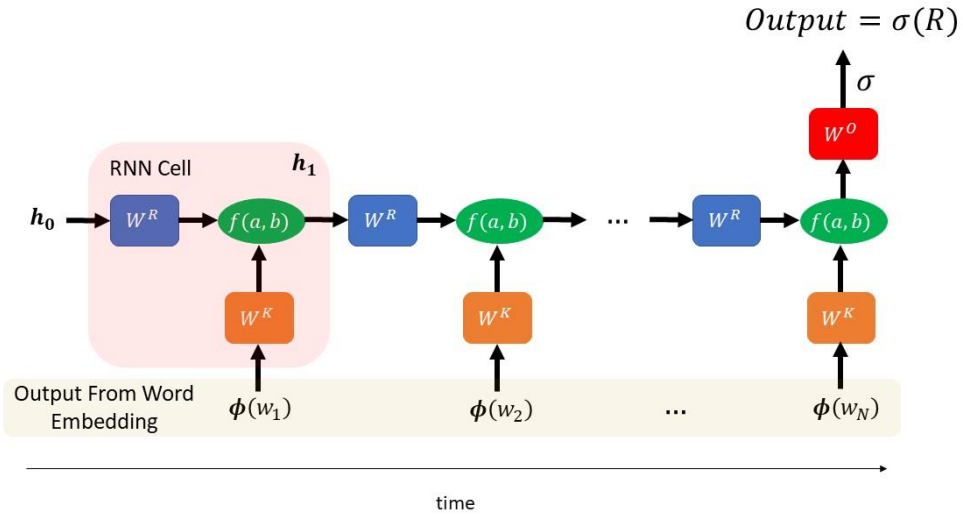


Figure 2.1: This figure illustrates the components of a simple RNN and how it works. A simple RNN is composed of a small subunit called a cell. The inputs to the cell are different at each time step, but the operations in the cell are identical. The hidden state h_0 is fed to the RNN cell at time step $t = 1$ together with the first word vector $\boldsymbol{\phi}(w_1)$. The output is called h_1 and will be fed to the next time step at t_2, and so on. This process keeps repeating until it reaches the final step with the hidden state output h_N, which will be passed to the output layer and sigmoid function, giving the final output, prediction.

Embedding is the process of transforming a w_t into a vector $\boldsymbol{\phi}(w_t) \in \mathbb{R}^{d_I}$ using an embedding layer (more on embedding layers in the following subsection). The operation of the RNN cell can be written as

$$\boldsymbol{h}_t = f(W^K \boldsymbol{\phi}(w_t), W^R \boldsymbol{h}_{t-1}, \boldsymbol{b}), \tag{2.1}$$

where $\mathbf{h}_t$ is a vector called hidden state at time t with dimension χ; W^K is called a kernel, which is a weight matrix for an input with dimension $d_I \times \chi$; W^R is a recurrent kernel which is a weight matrix for the hidden state with dimension $\chi \times \chi$; and $\mathbf{b}$ is a bias. The structure of the function f depends on what model we are using. It can be simple, as in vanilla RNN, where the function f is in the form of $\tanh(W^K \boldsymbol{\phi}(w_t) + W^R \boldsymbol{h}_t - 1 + \boldsymbol{b})$, or very complicated, as in LSTM, which contains

many operations and non-linearity terms. In machine learning, we call this function an activation function. The rectified linear unit or ReLU, defined by $f(x) = \text{Max}\{0, x\}$, along with its variant [11], is the most commonly used activation function in modern machine learning. The inspiration for these functions is to mimic the behavior of human neurons. It is believed that each neuron requires a certain threshold of incoming signals to be triggered. Hence, the activation function does the same job here. As for the bias term, it can be thought of as a background signal of a neuron.

When the RNN cell is specified, we are now ready for the real RNN calculation. By starting with initialized $\boldsymbol{h}_0$ (mostly set as null state) and the input $\boldsymbol{\phi}(w_1)$, the hidden state $\boldsymbol{h}_1$ can be calculated. Then, continuing the calculation recursively using the input $\boldsymbol{\phi}(w_2), \dots, \boldsymbol{\phi}(w_3)$ and the previous hidden state $(\boldsymbol{h}_1, \dots, \boldsymbol{h}_{N-1})$ with the same cell structure, the hidden states for time steps $t = 2$ to $t = N$, i.e., $\boldsymbol{h}_2, \dots, \boldsymbol{h}_N$, can be retrieved. The hidden state at the time t aggregates all the information from the past steps 1 to $(t - 1)$. Thus, the long-term correlation can be achieved without the need for the model to grow with the sequence length. The past input can affect the present prediction, while maintaining the independence between the length of a series and model parameters is what we needed.

In the sentiment task, we expect the output to be 0 or 1, $O \in \{0,1\}$. However, the hidden state is a vector, so we need a way to transform them to a scalar, which can be done by matrix multiplication. We define a dense matrix that will be multiplied with the last hidden state as W^O with dimension $\chi \times 1$, and a bias can also be added here. Nevertheless, the matrix W^O and bias are not enough to make the output within the [0,1] range, so we introduce another activation function here called the sigmoid activation function, which has the form $\sigma(x) \equiv \frac{1}{1+e^{-x}}$. Thus, the explicit form of the last layer is

$$O = \frac{1}{1 + \exp(-R)}, \tag{2.2}$$

where $R = W^O \boldsymbol{h}_N + b_o$ is the result before being put in the sigmoid function. We also denote a set of parameters that affect the output O as $\theta \equiv \{W^K, W^R, W^O, \boldsymbol{b}, b_o\}$. We will write O as O_θ to show that O_θ depends on the set of parameters in θ. Since the value of O_θ is inside the window $[0,1]$ not 0 or 1, the O_θ is interpreted as a probability of being 1 (or being 0 with probability $1 - O_\theta$).

The model parameters are initialized randomly. We expect them to be improved over time by a method called training. First, we let them do a task and measure how far the output predicted by the model and the expected results (the true answer) are. Then, the model's parameters are adjusted such that the distance between the model outputs and the expected results is closer than before. We use a function called loss function (L) to measure the aggregate of how far the predictions $O_\theta(\boldsymbol{w}_m)$ from the true sentiments $Y(\boldsymbol{w}_m)$, where $Y(\boldsymbol{w}_m) = 1$ for a positive review and 0 for a negative review. The $\boldsymbol{w}_m \equiv w_{1:N,m} = (w_1, w_2, \dots, w_N)_m$ denotes the m^{th} sequence in the dataset. If the dataset contains M sequences of words (M reviews in our case), then we have $m \in \{1,2, \dots, M\}$. Generally, for a binary classification problem whose output is probabilistic, we use the binary cross-entropy

$$L \equiv \sum_{m=1}^{M} \frac{1}{M} \big(Y(\mathbf{w}_m) \log[O_\theta(\mathbf{w}_m)] + \big(1 - Y(\mathbf{w}_m)\big) \log[1 - O_\theta(\mathbf{w}_m)]\big). \quad (2.3)$$

The whole training process is to search for θ^* that minimizes the loss function L or, in other words, the θ^* that makes the prediction as close to the true answer as possible (gradient descent, Adam, and their variants can be used to search for the θ^* [12]). Hence, we expect the model with the set of parameters θ^* will give the well the approximated probabilistic distribution that is

$$P(O|w_{1:N}) \approx \mathrm{RNN}_{\theta^*}(O|w_{1:N}), \quad (2.4)$$

where O is the output for the given sequence $w_{1:N}$, and RNN_{θ^*} is the RNN model with the given parameters θ^*.

2.3 Word Embedding

The word embedding was mentioned in the previous subsection, but we did not discuss it in detail. This subsection discusses the importance of the embedding layer or the word embedding. To change words into the form that a statistical model or a numerical computation can perform a calculation, one needs a way to modify them to numbers. One of the easiest ways is to tokenize them. To do this, one first sorts all the words in the dictionary in the ascending order from the most used to the lowest one. After the words are sorted, one can then tokenize each word by assigning each word with its order. As an example, assume that a dictionary contains {a, the, you}. If the most used word is “the” followed by “you" and “a" respectively, the tokenization is the process that maps {the, you, a} to {1, 2, 3}. Hence, each word sequence in the dataset can now be represented by a sequence of integers $w_{1:N}$ where each sequence is forced to have a constant length N. To do this, we do padding at the beginning of the sequence by 0. We use pre-padding instead of post-padding to ensure that a data vanishing will not occur during the training process. For example, we set $N = 5$ with the sentence “I love you” which can be encoded as $w_{1:5} = (0,0,7,53,8)$ where “I”$= 7$, “love”$= 53$, “you”$= 8$, and 0’s come from the padding process. Nevertheless, these processes are not enough. Each word has its meaning, but a number does not, so word embedding is needed. The embedding layer embeds a number (a word) which is a scalar $\mathbb{R}$ to a higher dimensional vector $\mathbb{R}^{d_I}$, and the elements of the embedding layer are set such that they can be trained (trainable parameters) during the training process. The embedded words could learn and adapt themselves corresponding to the given task by doing this. In this way, the embedding parameters can extract the meaning of the words in the dataset.

In this work, we use Word2vec [13]. The reason behind it is that they are simple to implement and keep the model’s simpleness. Although the complex structure can give rise to a better model, we are focusing on the explainability of the model. Thus, there is no point in building a complex model that we cannot explain, giving us more burdens than benefits. Assume that the size of the dictionary is D, and we want to embed each word to a vector with d_I dimensions. We define the

embedding function $\boldsymbol{\phi}(w)$ by a (trainable) matrix ϕ of size $d_I \times D$, where its i^{th} column represents the word vector $\boldsymbol{\phi}(w_i)$ of the word w_i. Notice that, in the last subsection, the hidden state at a given time t receives $\boldsymbol{\phi}(w_t)$ as the input. This means that the model's output does not depend only on its weight but also on the embedding layer. Thus, if the (unfixed) embedding layer is used in our model, the set of trainable parameters θ should include the ϕ within its, $\theta \equiv \{W^K, W^R, W^O, \mathbf{b}, b_o, \phi\}$.

Because the word embedding is a matrix ϕ of size $d_I \times D$, it will be more intuitive to write an input as a vector w_i such that $\phi \mathbf{w}_i = \boldsymbol{\phi}(w_i)$. One of the simplest ways to do this is one-hot encoding. If we have a dictionary of size D and the i-th word w_i in the dictionary, we can encode them to a vector of dimension D whose all elements are zero except the i-th position which equal to one, $\boldsymbol{w}_i = (0, \dots, 0, 1, 0, \dots, 0)$. Combining with the definition of the word embedding, we have

$$\phi \boldsymbol{w}_i = \begin{pmatrix} | & & | & & | \\ \boldsymbol{\phi}(w_1) & \dots & \boldsymbol{\phi}(w_i) & \dots & \boldsymbol{\phi}(w_D) \\ | & & | & & | \end{pmatrix} \boldsymbol{w}_i = \boldsymbol{\phi}(w_i). \tag{2.5}$$

After the training process, we get the θ^* which includes the trained embedding layer ϕ^* in there, but how could we know that this new embedding layer is useful, or does it learn a proper representation of the words? Since each word is already embedded into the higher dimension, the simple yet intuitive metric is to look at the angle between two words in the higher dimension using an inner product. One could argue that why can't we measure the Euclidean distance instead of the angle between two words? The magnitude of the vector (and hence the distance) can depend on the occurrence of a word. This implies that the two words with similar meanings can have a long distance between them since one may occur more frequently compared to the other. However, the angle between them is more resilient to this variation, and hence measuring the angle (or cosine similarity) is semantically preferred [14]. Given two words w_a and w_b, the cosine similarity, which is the cosine of the angle between these words, can be computed as

$$\text{sim}\big(\boldsymbol{\phi}(w_a), \boldsymbol{\phi}(w_b)\big) = \frac{\boldsymbol{\phi}(w_a) \cdot \boldsymbol{\phi}(w_b)}{|\boldsymbol{\phi}(w_a)||\boldsymbol{\phi}(w_b)|}. \tag{2.6}$$

If two words give positive cosine, they tend to be used in a similar context. Otherwise, it shows the opposite meaning of two words, except zero cosines which shows that two words might not correlate.

Although we keep our embedding layer as simple as possible which can be seen that there is no non-linear function, the RNN layer is still implicated with several iterations of activation functions, hence yielding highly non-linearity. In the following subsection, we discuss the attempt to map the RNN to a MPS by using a specific activation function, which helps us transform the highly non-linearity into a simple tensor operation with no activation function required.

2.4 Matrix Product State and Recurrent Arithmetic Circuits

This section will focus on the mapping between RNN to MPS. For simplicity, let us restrict ourselves to the RNN with no bias. The bias can be added back to the model later quite easily, as we will show in the following section.

Instead of setting the activation function f as a non-linear function, let us define f as element-wise multiplication or Hadamard product

$$f^{RAC}(\boldsymbol{A}, \boldsymbol{B}) = \boldsymbol{A} \odot \boldsymbol{B}, \tag{2.7}$$

which can be written in the component form as $f_i^{\text{RAC}}(\boldsymbol{A}, \boldsymbol{B}) = A_i B_i$. The RNN that uses the structure in equation (2.1) together with the f^{RAC} activation function is known as Recurrent Arithmetic Circuits (RACs). The special thing about RACs is that it can be mapped to the Matrix Product State (MPS) or Tensor Train (TT) [15]. To show this, we introduce a rank-3 tensor δ_{ijk} where the value of tensor is 1 if $i = j = k$, and 0 otherwise:

$$\delta_{ijk} = \begin{cases} 1, & \text{if } i = j = k \\ 0, & \text{otherwise.} \end{cases} \tag{2.8}$$

Then, the element-wise multiplication of the two tensors $\boldsymbol{A}$ and $\boldsymbol{B}$ can be written in the component form as

$$(\boldsymbol{A} \odot \boldsymbol{B})_k = \sum_{i,j} A_i \, B_j \delta_{ijk} = A_k B_k. \tag{2.9}$$

Thus, using the equation (2.1), the hidden state of RACs at time t (with no bias) can be written in the component form as

$$\boldsymbol{h}_t^{(\alpha_t)} = \sum_{i,j,\alpha_{t-1}=1}^{\chi} \sum_{\gamma_t=1}^{d_I} \delta_{ij\alpha_t} W_{i\gamma_t}^K \phi_{\gamma_t}(w_t) W_{j\alpha_{t-1}}^R \boldsymbol{h}_{t-1}^{(\alpha_{t-1})}. \tag{2.10}$$

Let us go a little further by defining a new tensor $T_{\alpha_t\alpha_{t-1}}^{\gamma_t} = \sum_{i,j}^{\chi} \sum_{\gamma_t}^{d_I} W_{i\gamma_t}^K \, W_{j\alpha_{t-1}}^R$ to make thing cleaner. Thus, equation (2.10) becomes

$$\boldsymbol{h}_t^{(\alpha_t)} = \sum_{\gamma_t=1}^{d_I} \sum_{\alpha_{t-1}=1}^{\chi} \phi_{\gamma_t}(w_t) T_{\alpha_t\alpha_{t-1}}^{\gamma_t} \boldsymbol{h}_{t-1}^{(\alpha_{t-1})}. \tag{2.11}$$

By exploiting its recurrent structure, we can substitute $\boldsymbol{h}_{t-1}^{(\alpha_{t-1})}$ into itself, and, hence, its explicit form can be retrieved

$$\boldsymbol{h}_t^{(\alpha_t)} = \sum_{\gamma_t,\ldots,\gamma_1=1}^{d_I} \sum_{\alpha_{t-1},\ldots,\alpha_0=1}^{\chi} \left[\phi_{\gamma_t}(w_t) T_{\alpha_t\alpha_{t-1}}^{\gamma_t}\right] \left[\phi_{\gamma_{t-1}}(w_{t-1}) T_{\alpha_{t-1}\alpha_{t-2}}^{\gamma_{t-1}}\right] \ldots \tag{2.12}$$

$$\left[\phi_{\gamma_1}(w_1) T_{\alpha_1\alpha_0}^{\gamma_1}\right] \boldsymbol{h}_0^{(\alpha_0)}.$$

One can obviously see that the hidden output at time step t can be constructed using three components. The first component is a translational invariant MPS, which takes the form

$$\bar{\Psi}_{\alpha_t\alpha_0}^{\gamma_t\cdots\gamma_1} = \sum_{\alpha_{t-1},\ldots,\alpha_1=1}^{\chi} T_{\alpha_t\alpha_{t-1}}^{\gamma_t} T_{\alpha_{t-1}\alpha_{t-2}}^{\gamma_{t-1}} \ldots T_{\alpha_1\alpha_0}^{\gamma_1}. \tag{2.13}$$

The second one is

$$\Phi_{\gamma_t,\ldots\gamma_1} = \phi_{\gamma_t}(w_t)\phi_{\gamma_{t-1}}(w_{t-1}) \ldots \phi_{\gamma_1}(w_1), \tag{2.14}$$

which is the tensor of rank t constructed from the sequence of t word embedding vectors. The last component is $\boldsymbol{h}_0$, which is the hidden state at the initial time step ($t = 0$). Therefore, the hidden state at time step t can be written as

$$\boldsymbol{h}_t^{(\alpha_t)} = \sum_{\gamma_t,\ldots,\gamma_1=1}^{d_I} \sum_{\alpha_0=1}^{\chi} \bar{\Psi}_{\alpha_t\alpha_0}^{\gamma_t\cdots\gamma_1} \Phi_{\gamma_t\cdots\gamma_1} \boldsymbol{h}_0^{(\alpha_0)}. \tag{2.15}$$

The main purpose of this subsection is to show that the RACs can be represented by an MPS. An entanglement entropy that can be extracted from the MPS is one of the main characters in this work. It can measure the information propagated between two bipartite sub-systems, as we will discuss next.

2.5 Entanglement Entropy

The von-Neumann entropy or entanglement entropy of a MPS can be derived from Schmidt coefficient λ, which can be obtained from applying a singular value decomposition (SVD) to partition a system into two sub-systems. Let H be a Hilbert space of dimension $n \times m$, and there are two sub-Hilbert spaces H_L and H_R with dimension n and m respectively, where $H = H_L \otimes H_R$. Without loss of generality, assume that $n \geq m$, which can be easily extended to the general case. Define orthonormal bases $\{u_1, \ldots, u_n\} \subset H_L$ and $\{v_1, \ldots, v_m\} \subset H_R$. Then an arbitrary vector $w \in H$ can be written as

$$w = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{ij}\, u_i \otimes v_j. \tag{2.16}$$

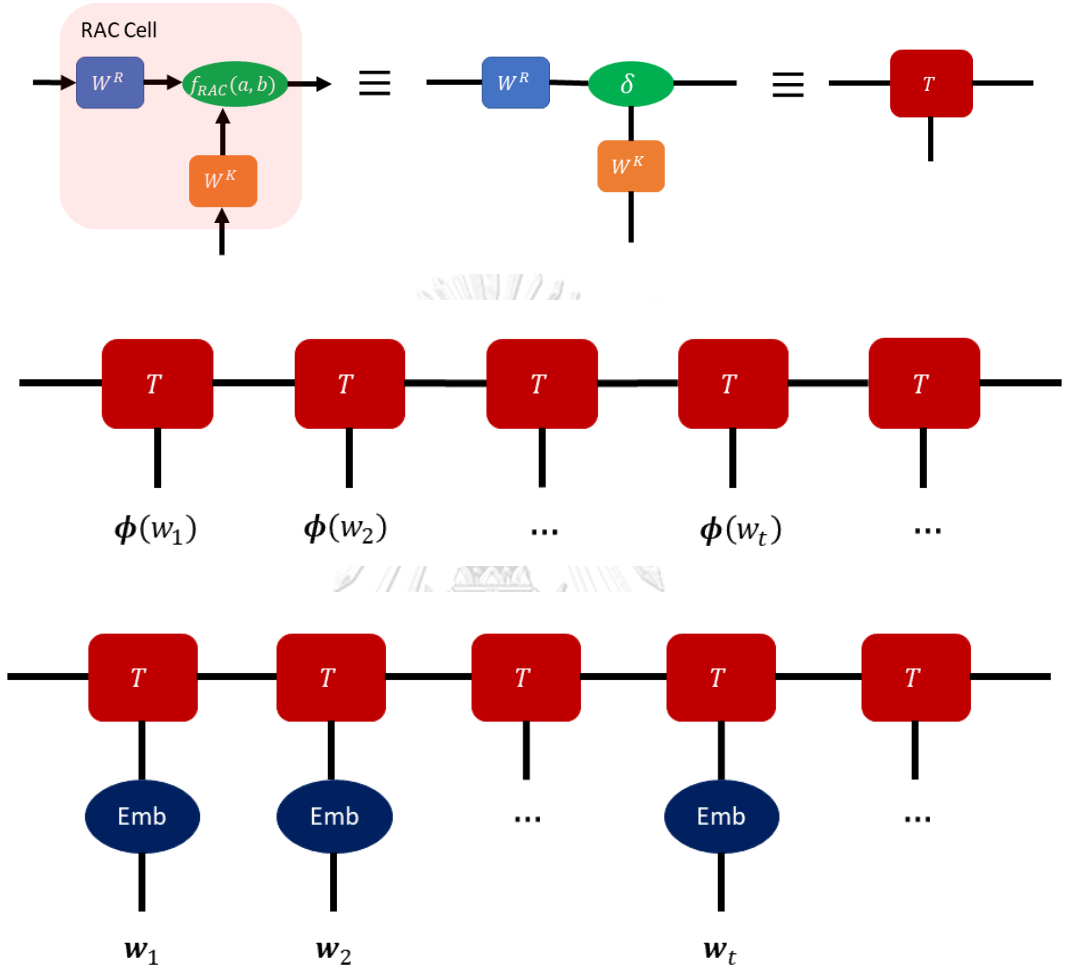


Figure 2.2: (a) depicts a graphical representation of how the RAC cell transforms to a new rank-3 tensor T, as described in (2.12). The RAC can then be expressed as the translational invariant MPS (b) by mapping the cell to T. The word embedding, according to section 2.3, is a matrix of dimension $d_I \times D$ contract with a D-dimension one-hot vectorw_i. As a result, the explicit form of (b) is (c), which is the RAC contract to the embedding matrix denoted as Emb in diagram (c). This suggests that the contraction of T and Emb results in a new rank-3 tensor, implying that the word embedding may potentially contribute to the entanglement entropy of the system.

From singular value decomposition (SVD), there exist an $n \times n$ orthonormal matrix L, an $m \times m$ orthonormal matrix R, and an $m \times m$ positive semidefinite diagonal matrix Σ such that a matrix M can be written/decomposed as

$$M = L \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} R^T, \tag{2.17}$$

where 0 is the zero matrix with dimension $(n-m) \times m$. Next, split L into two sub matrices $L = [L_1 \quad L_2]$, where L_1 is $n \times m$. By doing so, the equation (2.16) becomes

$$M = L_1 \Sigma R^T. \tag{2.18}$$

Let $\{l_1, \dots, l_n\}$ and $\{r_1, \dots, r_m\}$ be another set of orthogonal bases of H_L and H_R respectively and $\{\sigma_1, \dots, \sigma_m\}$ be diagonal elements of the matrix Σ. Here, l_i and r_j are the i_{th} and the j_{th} column vectors of the matrices L and R. The above equation becomes

$$M = \sum_{k=1}^{m} \sigma_k \, l_k r_k^T. \tag{2.19}$$

The σ_k in the above equation is called Schmidt coefficient satisfying $\sum_{k=1}^{m} \sigma_k^2 = 1$. Since α_{ij} in the equation (2.16) is a matrix, it is possible to apply the SVD to the α_{ij}. By choosing the appropriate set of bases, transforming the $\{u_1, \dots, u_n\} \to \{l_1, \dots, l_n\}$ and $\{v_1, \dots, v_m\} \to \{r_1, \dots, r_m\}$, we can rewrite w as

$$w = \sum_{k=1}^{m} \lambda_k \, l_k \otimes r_k, \tag{2.20}$$

where λ_k is the Schmidt coefficients of w which satisfies $\sum_{k=1}^{m} \lambda_k^2 = 1$. This can be applied to the MPS with closed boundary as well. However, the boundary of our MPS in (2.13) is still open because the α_0 and α_t are the free indices. To close them, we need to contract its boundaries, α_0 and α_t, with vectors. We will discuss about the appropriate choice of vectors in the next section. For now, let us assume that the MPS already has a closed boundary and given by $\Psi^{\gamma_t \cdots \gamma_2 \gamma_1}$. Since the MPS $\Psi^{\gamma_t \cdots \gamma_2 \gamma_1}$ is a

rank-t tensor, we know that it can be written in the quantum state form (simply put the bases state in) as

$$|\Psi\rangle_{\mathrm{MPS}} = \sum_{\gamma_1,\gamma_2,\ldots,\gamma_t} \Psi^{\gamma_t \ldots \gamma_2 \gamma_1} |\gamma_t\rangle \otimes \ldots |\gamma_2\rangle \otimes |\gamma_1\rangle. \tag{2.21}$$

Then, the MPS is bipartited into two sub-systems $\left|\phi_i^L\right\rangle \in H_L$ and $\left|\phi_i^R\right\rangle \in H_R$ where they are made up from the $|\gamma_1\rangle, \ldots, \left|\gamma_{t/2}\right\rangle$ and $\left|\gamma_{t/2+1}\right\rangle, \ldots, |\gamma_t\rangle$ respectively. Since the MPS is constructed using the same tensor (RAC cell) along the chain, making it a translational invariant MPS, it does not matter where one cut them. However, cutting near the edge of the tensor should be avoided as the boundary effect may occur since we cannot truly build an infinite chain of tensors (bounded by computational resources). According to SVD that we have done in (2.16)-(2.20), we have

$$|\Psi\rangle_{MPS} = \sum_{i=1}^{l} \lambda_i \left|\phi_i^L\right\rangle \otimes \left|\phi_i^R\right\rangle . \tag{2.22}$$

The entanglement entropy [16] between two subsystems of the $|\Psi\rangle_{MPS}$ can be calculated by

$$S = -\sum_{i=1}^{l} \lambda_i^2 \log_2 \lambda_i^2. \tag{2.23}$$

The entanglement entropy measures how much two subsystems are entangled. There are several ways to interpret entanglement entropy. One of them is to use pure states and mixed states [17]. A pure quantum state is used to describe the state in which we have complete knowledge about the quantum system, which has $S = 0$. On the other hand, a mixed state, which $S \neq 0$, can be thought of as the state in which we do not have the complete knowledge of the system since we trace over some components of a pure state (like we sum over some probability in statistic class). Thus, the non-zero

entropy could be interpreted as the ignorance of knowledge or the correlation between the current state and the ignored state [18].

To make the above statement clearer, let us translate it to the physics notation and maths. The rough definition of a pure state is the state that can be written in a vector form (e.g., $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$), but a mixed state is opposite, which cannot be written in the vector form like the pure state. Thus, a density matrix is introduced to enable us to represent the mixed state naturally. The density matrix ρ of the pure state $|\psi\rangle$ is defined by

$$\rho = |\psi\rangle\langle\psi|. \tag{2.24}$$

For the mixed state, it is defined by the summation of all possible pure state,

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|, \tag{2.25}$$

where p_i is a classical probability that the system will end up in the pure state $|\psi_i\rangle$. One can also differentiate the pure and mixed states using the density matrix ρ. Using equations (2.24) and (2.25), we can show that $\rho^2 = \rho$ for the pure state, but this is not true in the case of the mixed state. From Shannon entropy, the entropy of the ρ is

$$S_{\text{Shannon}} = -\sum_i p_i \log_2 p_i. \tag{2.26}$$

Since we know that $\text{Tr}(\rho\log_2\rho) = \sum_i p_i \log p_i$ (by using the fact that trace is invariant under bases transformation), we then define the von-Neumann entropy or quantum-version of entropy as

$$S = -\text{Tr}(\rho\log\rho). \tag{2.27}$$

If we calculate the entropy of the $|\Psi\rangle_{MPS}$, it will obviously give zero due to the fact that it is a pure state. However, if we trace out its component, let's pick the right part $|\phi_i^R\rangle$, then the corresponding density matrix becomes a mixed state

$$\rho = \sum_{i,j=1}^{n} \lambda_i \lambda_j |\phi_i^L\rangle \otimes |\phi_i^R\rangle\langle\phi_j^R| \otimes \langle\phi_j^L|$$

$$\rho_L = \mathrm{Tr}_R(\rho) = \sum_{i,j,k=1}^{n} (\mathbb{1} \otimes \langle\phi_k^R|)\lambda_i\lambda_j|\phi_i^L\rangle \otimes |\phi_i^R\rangle\langle\phi_j^L| \otimes \langle\phi_j^R|(\mathbb{1} \otimes |\phi_k^R\rangle) \tag{2.28}$$

$$\rho_L = \sum_{i,j,k=1}^{n} \delta_{ik}\,\delta_{jk}\lambda_i\lambda_j|\phi_i^L\rangle\langle\phi_j^L| = \sum_{k=1}^{n} \lambda_k^2|\phi_k^L\rangle\langle\phi_k^L|$$

The above equation shows that the density matrix cannot be decomposed into the outer product of two states, except $n = 1$. Comparing ρ_L to the equation (2.25), we have $p_i = \lambda_i^2$, which, in this case, gives $S \neq 0$. This implies that the non-zero entropy comes from the lack of knowledge of the $|\phi_k^R\rangle$. If we know what $|\phi_k^R\rangle$ is, then we can reconstruct back the complete pure state. If not, we are left with the mixed state with an ensemble of a new set of pure states $|\phi_k^L\rangle$, and we might conclude that the subsystems $|\phi_k^L\rangle$ and $|\phi_k^R\rangle$ entangle to each other in such a way that if we lost either one of these states, we lost the information to construct a whole system. Thus, subsystem *L* and *R* share information with each other.

We expect that the stronger the correlation of the two subsystems is, the larger the entanglement entropy becomes (due to the more ignorance of the information). This implies that the partitioned RACs with large entanglement entropy might be a good model for approximating data with strong correlations. Another great explanation can be found in [15]. The mentioned paper describes the correlation in terms of separation rank measured from the Schmidt coefficients, which can be used to calculate the entanglement entropy we just defined.

Nevertheless, in the entanglement entropy calculation, we did not consider the effect of the embedding layer. Since the word embedding is an ordinary matrix that contracts with the RACs, it can be viewed as a part of the RAC. Thus, the embedding

layer can also contribute to the entanglement entropy of the whole system. One more thing to point out is that we also ignored the bias term when the RAC model was constructed. Adding the bias to the model is a tricky part of this work. In the next section, we will cover how to add the bias to the model and the numerical effect of the word embedding.

3. Experiments & Results

In this section, we will go through how to set up the model and continue from where we have left in the previous section. So far, we have discussed the theoretical perspective of the model, which gives the intuition of the physical meaning behind it, but there are some topics that we have not yet covered. First, the bias that we have ignored plays a crucial role in this work. Because our model only comprises the tensor contraction, this leads to a gradient vanishing. This problem arises due to the contraction is composed of many multiplication terms. If we start with a small number, the results from the operation will converge to zero quickly. To solve this, we need to add a summation term, a bias, to stabilize the result of the contraction not getting too small.

The following subsection will show how the bias is added to our MPS. Next, we will show the result of the accuracy together with the entanglement entropy of the RACs. As mentioned in the previous section, the word embedding might contribute to the entanglement entropy. To show that this is the case, we divide our training process into two following strategies: fixed pre-trained word embedding and (non-fixed) regular training. The idea of the first strategy is to fix the effect of the entanglement entropy that arises from the word embedding. In this case, we show that the entanglement entropy and accuracy of the model behave in the same way. In the beginning, they slightly increase, but after reaching some specific point, both entanglement entropy and accuracy are saturated. This interesting behavior helps us determine the smallest dimension (or parameter) that can be sufficiently used to approximate our task. Second, a word embedding is trained together with the RAC. We do this study on the effect of the embedding layer that affects the entanglement entropy. Surprisingly, although the accuracy behaves the same way as the first strategy, the entanglement entropy is different. The entanglement entropy rapidly increases at the beginning before it decreases and plateaus at some saturate value. We also report the cosine similarity of the word embedding, which exhibits the opposite

behavior to the entanglement entropy—implying that the embedding layer involves propagation of the information.

3.1 RACs with Bias

Before we add the additive biases to the model, first let's rewrite the RACs in equation (2.1) in the easier form as

$$\mathbf{h}_t = f(W^K \boldsymbol{\phi}(w_t), W^R \mathbf{h}_{t-1}, \mathbf{b}) \to f_{\text{RAC}}\left(W^K \boldsymbol{\phi}(w_t) + \mathbf{b}^k, W^R \mathbf{h}_{t-1} + \mathbf{b}^R\right). \tag{3.1}$$

To achieve this, we transform the kernel matrix W^K and recurrent kernel matrix W^R to the new matrix $\widetilde{W}^R$ and $\widetilde{W}^K$ as follow

$$\widetilde{W}^K = \underbrace{\begin{pmatrix} W^K & \boldsymbol{b}^k \\ 0 \dots 0 & 1 \end{pmatrix}}_{d_I + 1} \Big\} \chi + 1, \quad \widetilde{W}^R = \underbrace{\begin{pmatrix} W^R & \boldsymbol{b}^R \\ 0 \dots 0 & 1 \end{pmatrix}}_{\chi + 1} \Big\} \chi + 1 \tag{3.2}$$

Because $\boldsymbol{\phi}(w_t)$ and $\boldsymbol{h}_t$ are contracting to the W^R and W^K, we need to enforce them to have the same dimension by also transforming them by adding an additional dimension as

$$\widetilde{\boldsymbol{\phi}}(w_t) = \begin{pmatrix} \boldsymbol{\phi}(w_t) \\ 1 \end{pmatrix}, \qquad \widetilde{\boldsymbol{h}}_t = \begin{pmatrix} \boldsymbol{h}_t \\ 1 \end{pmatrix}. \tag{3.3}$$

By using these new forms, one can immediately see that

$$\widetilde{W}^K \widetilde{\boldsymbol{\phi}}(w_t) = \begin{pmatrix} W^K & \boldsymbol{b}^k \\ 0 \dots 0 & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{\phi}(w_t) \\ 1 \end{pmatrix} = \begin{pmatrix} W^K \boldsymbol{\phi}(w_t) + \boldsymbol{b}^k \\ 1 \end{pmatrix} \tag{3.4}$$

and

$$\widetilde{W}^R \widetilde{\boldsymbol{h}}_t = \begin{pmatrix} W^R & \boldsymbol{b}^R \\ 0 \dots 0 & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{h}_t \\ 1 \end{pmatrix} = \begin{pmatrix} W^R \boldsymbol{h}_t + \boldsymbol{b}^R \\ 1 \end{pmatrix}. \tag{3.5}$$

Thus, the new $\widetilde{\boldsymbol{h}}_t$ can be written in the form of activation function as

$$\widetilde{\boldsymbol{h}}_t = \begin{pmatrix} f_{\mathrm{RAC}}(W^K \boldsymbol{\phi}(w_t) + \boldsymbol{b}^k, W^R \boldsymbol{h}_{t-1} + \boldsymbol{b}^R) \\ 1 \end{pmatrix}. \tag{3.6}$$

or

$$\widetilde{\boldsymbol{h}}_t = \sum_{i,j,\alpha_{t-1}=1}^{\chi+1} \sum_{\gamma_t=1}^{d_I+1} \delta_{ij\alpha_t} \widetilde{W}^K_{i\gamma_t} \widetilde{\boldsymbol{\phi}}_{\gamma_t}(w_t) \widetilde{W}^R_{j\alpha_{t-1}} \widetilde{\boldsymbol{h}}^{(\alpha_{t-1})}_{t-1}. \tag{3.7}$$

Thus, one need to change the corresponding MPS tensor $T^{\gamma_t}_{\alpha_t\alpha_{t-1}} \rightarrow \tilde{T}^{\gamma_t}_{\alpha_t\alpha_{t-1}}$,

$$\tilde{T}^{\gamma_t}_{\alpha_t\alpha_{t-1}} = \sum_{i,j,\alpha_{t-1}=1}^{\chi} \sum_{\gamma_t=1}^{d_I} \widetilde{W}^K_{i\gamma_t} . \widetilde{W}^R_{j\alpha_{t-1}}. \tag{3.8}$$

Now, we can follow the same flow as (2.11)-(2.15), and the chain of MPS can be retrieved. Nevertheless, the MPS chain still has open boundaries at the beginning and the end. To close these, we define two additional vectors, $\widetilde{\boldsymbol{h}}_0$ and $\widetilde{\boldsymbol{h}}_f$, which are the $\widetilde{\boldsymbol{h}}_t$ at time $t = 0$ and the final time step. Normally, the initial state of the RNN is set to null vector or zero vector, so the $\widetilde{\boldsymbol{h}}_t$ at $t = 0$ becomes

$$\widetilde{\boldsymbol{h}}_0 = \begin{pmatrix} \boldsymbol{0} \\ 1 \end{pmatrix}. \tag{3.9}$$

Since we need to make sure that all the output results in the last time step must be taken into account, the appropriate choice is to contract them with the vector comprised of one in each component. Thus, we define the $\widetilde{\boldsymbol{h}}_f$ as

$$\tilde{\mathbf{h}}_f = \begin{pmatrix} \boldsymbol{1} \\ 1 \end{pmatrix}. \tag{3.10}$$

However, this choice of $\widetilde{\boldsymbol{h}}_0$ and $\widetilde{\boldsymbol{h}}_f$ should not affect the entanglement entropy much because if the chain is long enough, the boundary effect can be ignored. Lastly, one can perform the entanglement entropy calculation in the middle of the (closed boundary) MPS chain following the described flow in section 2.5.

In this subsection, we have modified our existing tensors to work with additional biases, whether by adding an additional dimension with one as a component or extending both row and column to include the bias vector within them. This simple modification allows us to use the same mathematics to calculate the entropy and maintain the same mathematical structure of the tensor forms. With this help, we can achieve the training process just like the pre-transformed form. In the following subsection, we will discuss the training model and the parameters and strategy needed in this work.

3.2 Data Prepossessing

The IMDb dataset is used in this experiment, which is a standard dataset for binary sentiment classification containing movie reviews [19]. This task has two labels, “1" for a positive review and “0" for a negative review. The dataset is divided into 40,000 reviews for training and 10,000 reviews for testing. The positive and negative reviews ratio in the train and test are $20{,}027:19{,}973$ and $4{,}913:5{,}027$, respectively. The maximum length of each review is set to $N = 50$ words with dictionary size $D = 3{,}000$ and $D = 10{,}000$. Because the datasets are different in size (determined by the number of reviews), the dictionary sizes might have an effect on the accuracy and behavior since they determine both numbers of parameters and words being used in the models.

We use Keras to implement our RACs model [20]. We use Adam optimizer to optimize the loss function for the training process. The number of epochs is set to 200 with batch size equal to 128. Early stopping is also deployed for stability reasons. The model is set to terminate if the change of the loss function at the end of each epoch is smaller than 0.001 and has no improvement after 4 epochs. The training process is done on a set of bond dimensions χ given by $\chi \in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,$

16,17,18,19,20,25,30,35,40,45,50,55,60,65,70,75,80]. Then, we repeat every training process 50 times with the randomly initialized model parameters for each round to obtain averaged accuracy and entanglement entropy for each χ, where the entropy is obtained by performing a bipartization at the middle of the MPS chain.

We also experiment on Rotten Tomatoes movie review (MR) with some tweaked parameters as follows [21]. In the MR dataset, the batch size, epochs, and maximum length are set to 32, 100, and 20, respectively. The parameters are changed because the size of the MR dataset, which 8,400 for the train and 2,662 for the test, is smaller than the IMDb. The rest of the flow is the same as the IMDb dataset.

3.3 Results on pre-trained word embedding

We want to focus solely on the behavior of RACs, so our first strategy is to fix the word embedding during the RACs training. However, to do that, we need to have a way to train our embedding layer first; otherwise, the outputs of the non-trained embedding layer are random values. We achieve this by pre-train the word embedding with the flatten layer (available in Keras). The structure of the model is almost the same, except the flatten layer is used instead of the RACs. The reason behind using flattening is that the layer does not contain trainable parameters. It only picks outputs from the previous layer, embedding layer in this case, concatenates them into a long sequence of numbers, then feeds this sequence to the next layer, which is a dense layer (W^O and a sigmoid function). The word embedding is pre-trained with the same setting as the RACs as described in section 3.2, except the early stopping plays a role in preventing overfitting instead of the stability of the model.

The pre-trained process of the word embedding ϕ is repeated over 50 times with randomly initialized parameters. Since we will train the RAC 50 times with random initialization parameters, this gives us a set of 50 distinct ϕ which can be used to train the RAC. Next, we set the embedding layer to be ϕ in the RACs model, then train the whole model again while the parameters of the pre-trained ϕ are fixed. The experimental results of the IMDb dataset with $d_I = 4$ and $D = 10{,}000$ are shown in figure 3.1, which shows the plots of accuracy and entanglement entropy of the

bipartited RACs at site $N/2 = 25$ for different bond dimensions χ. From $\chi = 1$ to $\chi = 20$, the entanglement entropy increases rapidly before it saturates at $\chi^* \approx 20$ and fluctuates mildly after that, where the maximum of the averaged entropy is $\bar{S}_{\max} \approx 2.53$. The training and test accuracy also shows a similar trend, but it increases much slower than the entropy when χ approaches χ^*. The training and test accuracy starts from 87.1% and 84.7% at $\chi = 1$ then climbs slowly and reaches 91.5% and 86.3% at $\chi \approx 20$. Figure 3.2 shows the plots of Schmidt's coefficients λ_i in the case $d_I = 4$ for each bond dimension χ as a function of indices, where the coefficients are labeled in the descending order, when $\chi < \chi^*$, the spectrum lines of the λ_i are separated. However, after the bond dimension χ hits the value of the χ^*, the spectrum of the λ_i of the χ that is larger than χ^* collapses to the same trend.

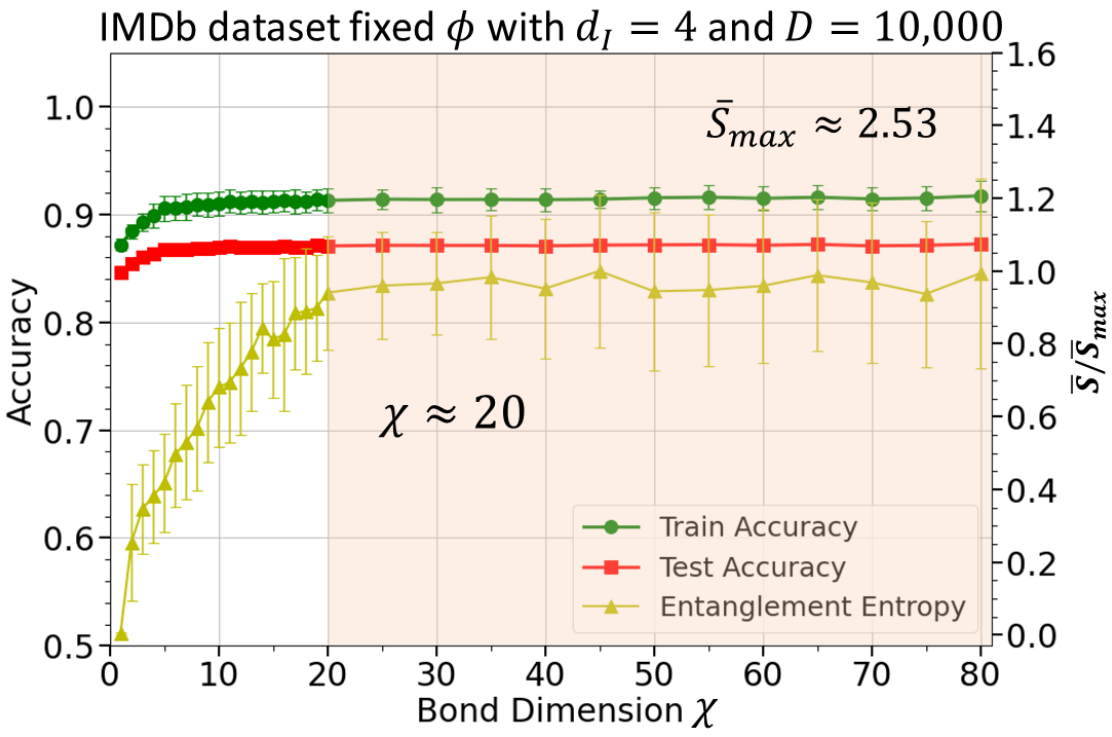


Figure 3.1: This plot shows the training accuracy, test accuracy, and the entanglement entropy of the IMDb dataset with $d_I = 4$ and $D = 10{,}000$, where the saturation occurs at around $\chi \approx 20$, and the averaged maximum entropy $\bar{S}_{\max} \approx 2.53$.

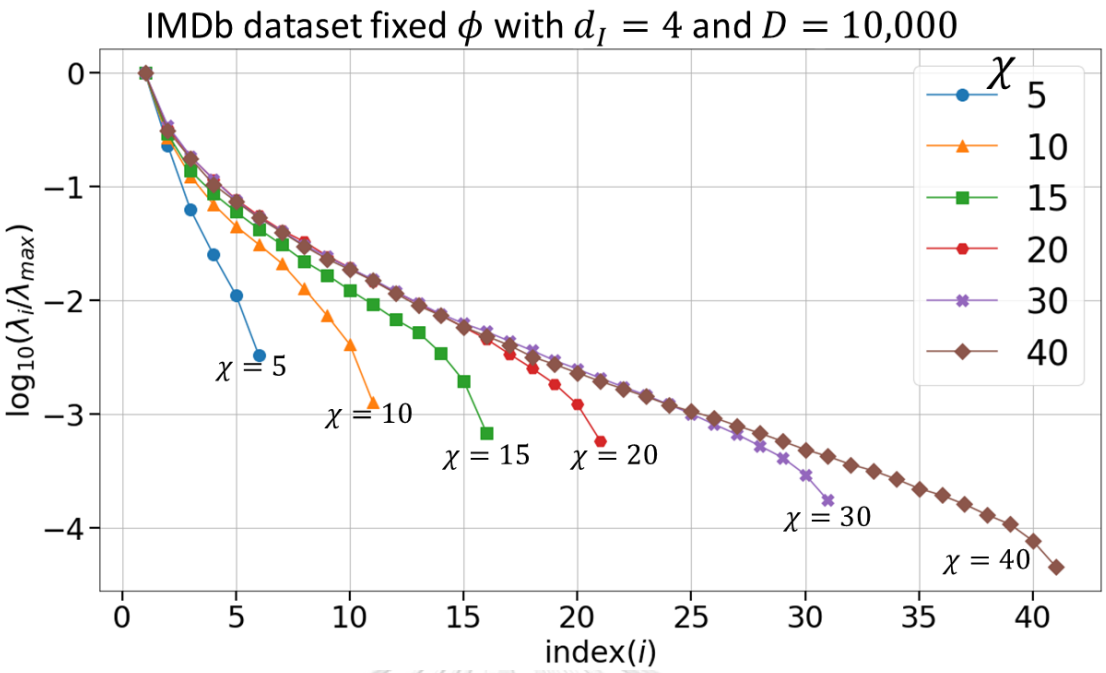


Figure 3.2: This graph illustrates the Schmidt coefficients λ_i used to calculate entanglement entropy S. The λ_i's are plotted as a function of indices labeled in a descending order corresponding to their value. The spectrum of the Schmidt coefficients collapses to the same trend when $\chi > 20$ except for the deviation at the tails, which are the results of the boundary effect. When the MPS is infinitely long, the deviation should not be seen in the ideal case.

Recalling the entanglement entropy equation (2.23), the maximum entropy which the model can achieve is when $\lambda_i^2 = 1/(\chi + 1)$ or

$$S = \log_2(\chi + 1), \tag{3.11}$$

where $\chi + 1$ comes from the fact that we need to add one extra dimension for the availability of the biases. As shown in the figure 3.1 (b, this implies that only the leading indices of the Schmidt coefficients matter, which means the two subsystems weakly interact. Although $\log_2(21) \approx 4.39$ and $\log_2(81) \approx 6.34$ are the maximum value which entanglement entropy can reach, it instead saturates at $\bar{S}_{max} \approx 2.53$ rather than keep increasing with the bond dimension. This saturation, together with the weak interaction, suggests that RAC is not the main source of information propagation. Thus, we might investigate further the word embedding, which will be discussed next.

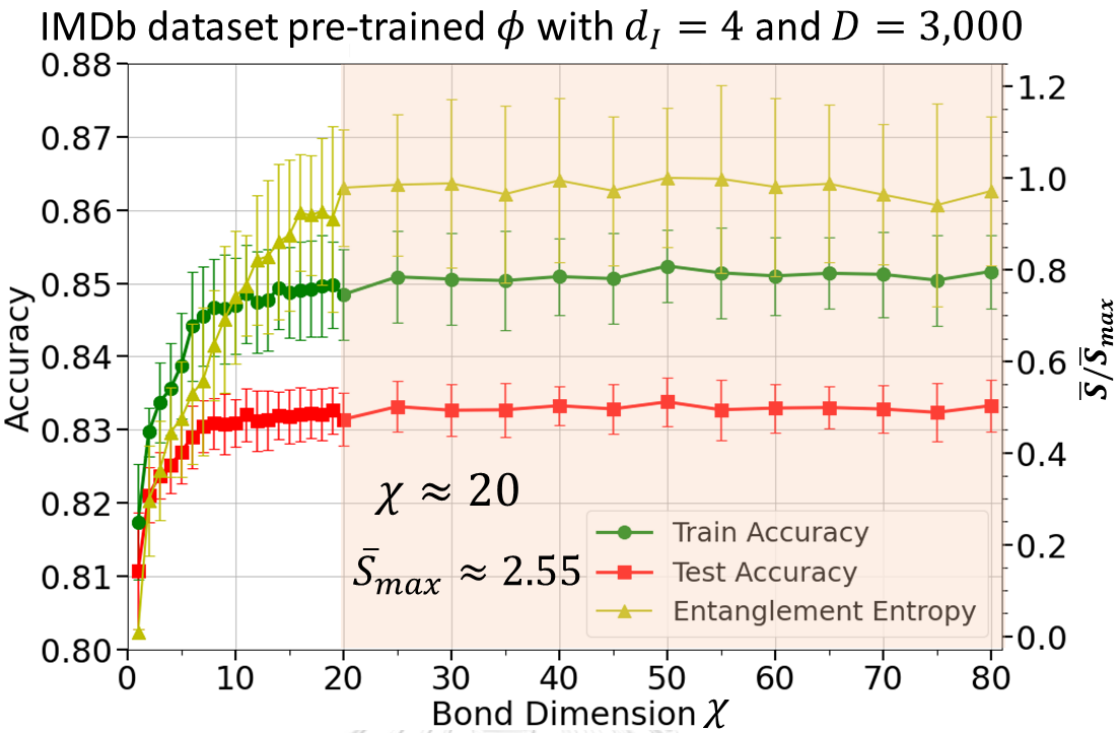


Figure 3.3: The result on IMDb with $d_I = 4$ and $D = 3{,}000$. The entropy saturates at $\chi \approx 20$ where $\bar{S}_{\max} \approx 2.55$.

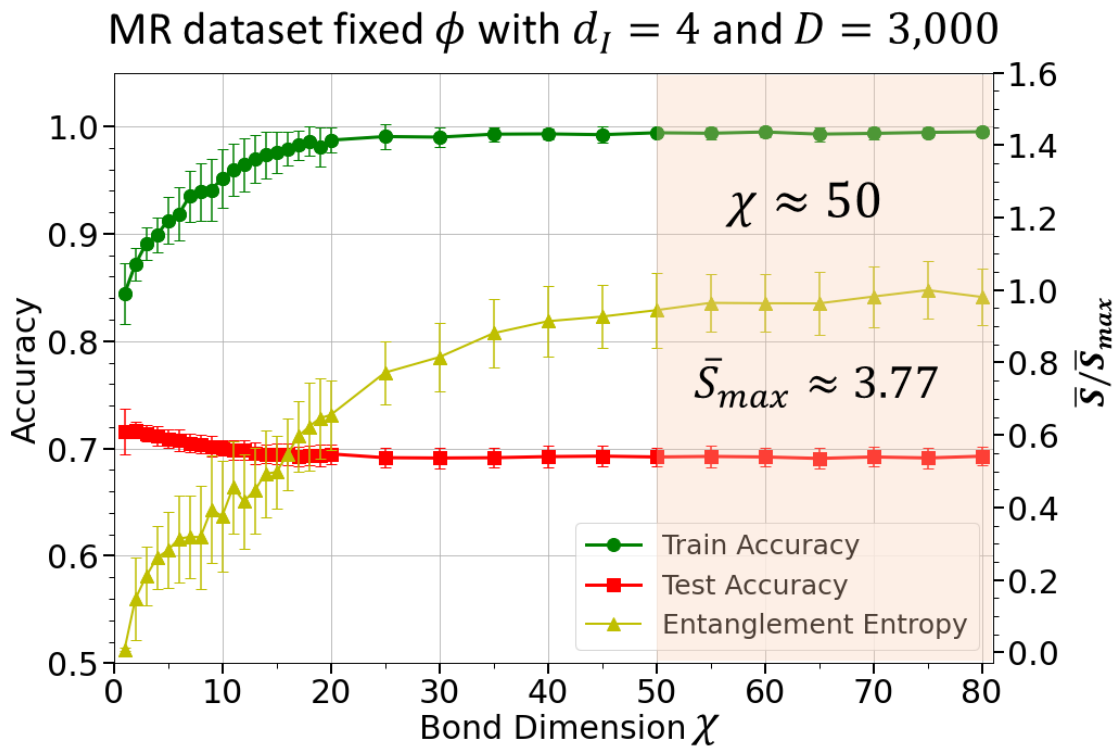


Figure 3.4: The plots of accuracy and the entropy of the model on the MR dataset with $d_I = 4$ and $D = 3{,}000$.

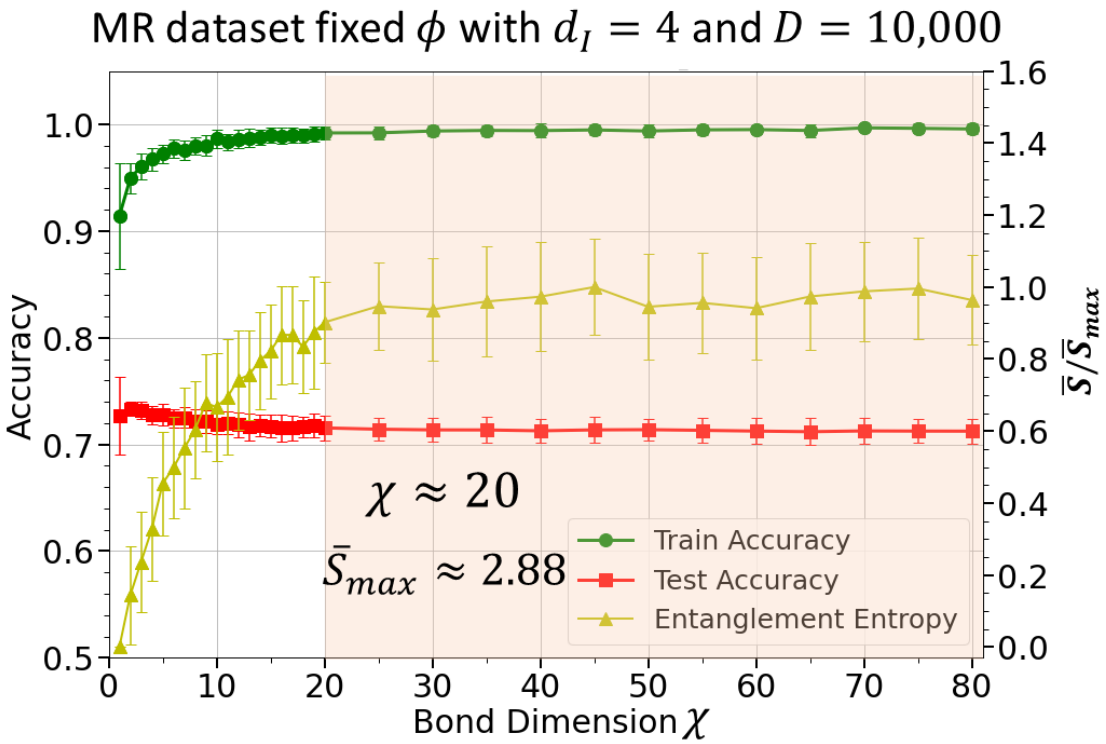


Figure 3.5: The plots of accuracy and the entropy of the model on the MR dataset with $d_I = 4$ and $D = 10{,}000$

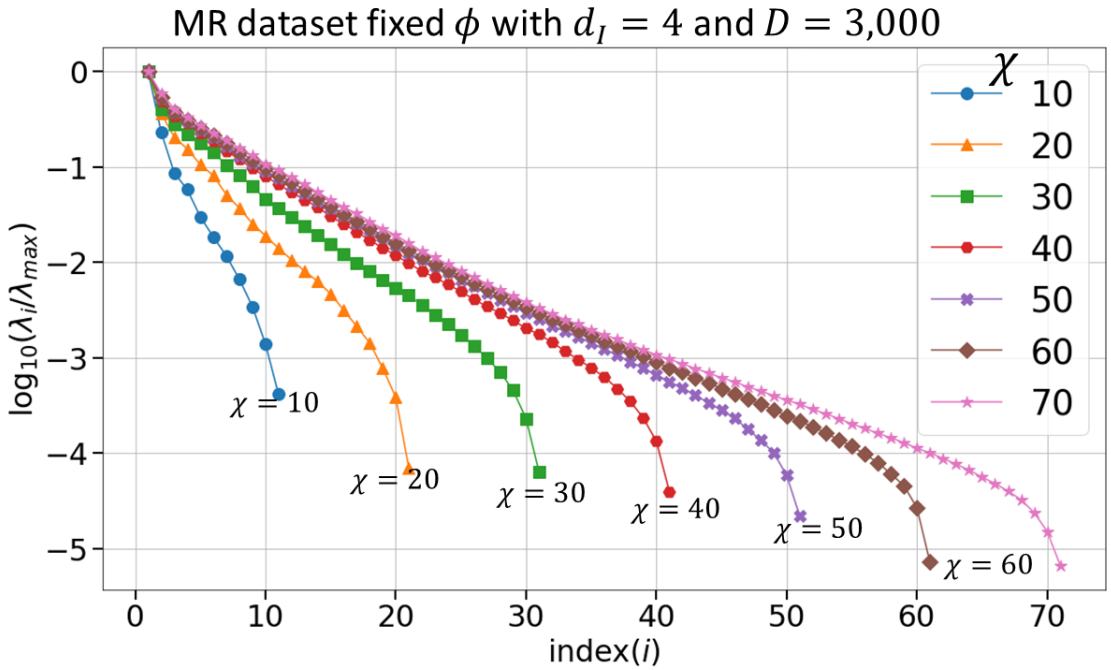


Figure 3.6: The plot of Schmidt coefficients λ_i on the MR dataset with $d_I = 4$ and $D = 3{,}000$.

From figure 3.3, the train and test accuracy of the IMDb dataset with a dictionary size of $D = 3{,}000$ saturate at 85.1% and 83.2%, which are lower than the $D = 10{,}000$ case. This is intuitive since the more parameters the model has, the higher the model's accuracy can achieve. Nevertheless, the accuracy and entropy of both $D = 3{,}000$ and $D = 10{,}000$ still plateau when they reach the specific bond dimension. Figures 3.4, 3.5 and 3.6 show the results of experiments conducted on the MR dataset, which show that the saturation phenomena appears in the other dataset.

In the fixed word embedding, the experiment's result tells us that the model can attain high accuracy, even though the subsystems of the MPS are weakly interacting, measuring from the entanglement entropy. In the following section, we train the RAC layer and the word embedding simultaneously. We show that information propagation of the RAC is not the main source of accuracy and expressiveness; rather, word embedding is important.

3.4 Results on unfixed word embedding

Unlike the previous experiment, the word embedding is not pre-trained, but we use a randomly initialized and unfixed word embedding. To study the effect the word embedding has on the model, we train the embedding layer with the RACs simultaneously. The results show that the entanglement entropy behaves differently from the fixed word embedding. In both datasets, the entropy shows increase and decay patterns. First, they sharply boost until hitting the highest value at bond dimension $\chi = 5$ before drops to about $0.8\bar{S}_{\max}$ and saturates at $\chi \approx 25$ and $\chi \approx 20$ for IMDb and MR dataset, respectively. In terms of accuracy, the model can achieve around 99% in both IMDb and MR datasets, which is higher than the previous case. Notice that the maximum of the average entropy which are $\bar{S}_{\max} \approx 1.17$ and $\bar{S}_{\max} \approx 1.20$ are lower than the fixed embedding case. The increase in accuracy while the entanglement entropy decreases are contrary to our belief that information propagation is the main source of accuracy and expressiveness. Because of the presence of the trainable word embedding, which is the only difference between this experiment and the previous one, this suggests that the embedding layer plays a role

in the contribution of the information or the entanglement entropy, which cannot be observed in the traditional RNNs.

To examine the word embedding in more detail, the cosine similarity between two opposite meaning words, which are (worst, best) and (boring, interesting), are shown in figure 3.9 and 3.10 for the IMDb and MR dataset, respectively. The words are chosen because they are the topmost frequently appear, and their meaning is likely to have a high effect on the sentiment prediction. One can see that the cosine similarity drops in both datasets while the entropy dramatically increases and hits its highest value at $\chi = 5$. When the entropy drops, the speed at which the cosine similarity decreases is also slowed down. When the dropping stops, the entropy and the cosine similarity saturate afterward. This behavior might imply that the model expressiveness mostly comes from the RACs when $\chi \leq 5$. However, when the word embedding becomes better, $\chi \geq 5$, the role is switched to the embedding layer, which can be seen from the decaying of RAC's entanglement entropy as the word embedding attains more meaning measured from the cosine similarity.

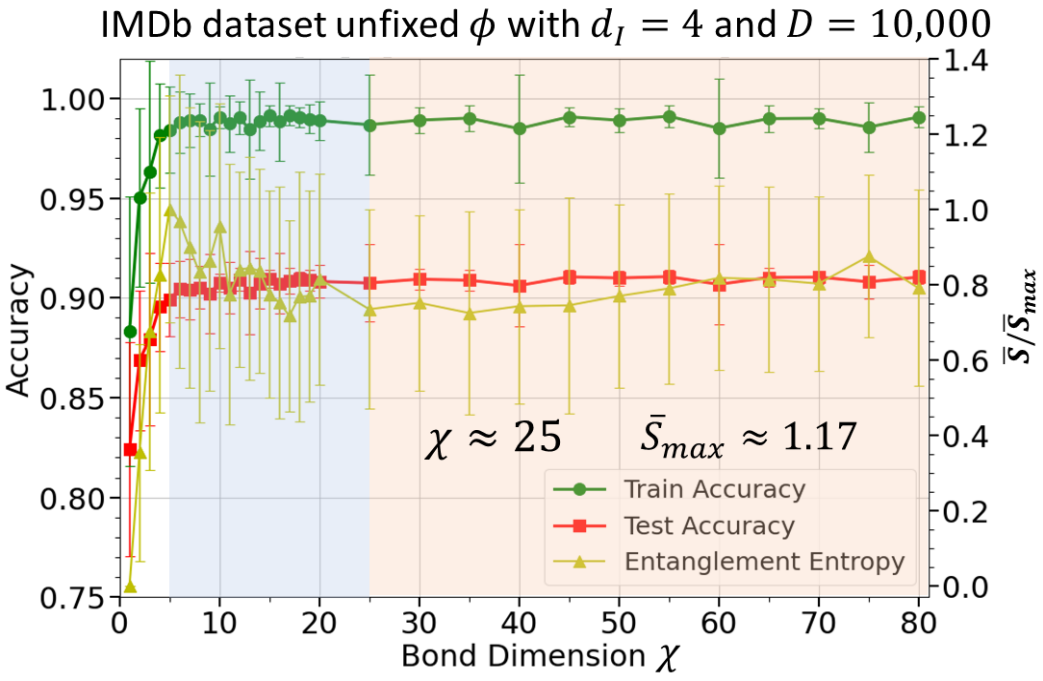


Figure 3.7: The plots of accuracy and the entropy of the unfixed word embedding on the IMDb dataset. Interestingly, unlike the fixed case, not only it can achieve higher accuracy and has lower maximum entropy, but it also shows different behavior in entropy which is the increase and decay to some saturated value.

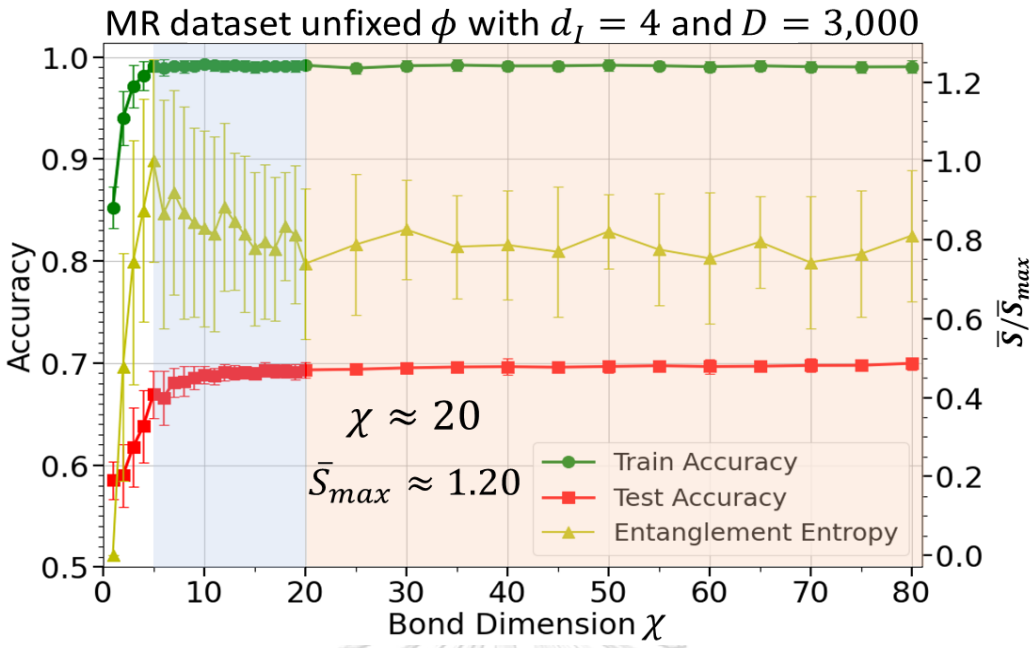


Figure 3.8: The result of the model on the MR dataset with unfixed word embedding illustrates that the same behavior appears in the MR dataset.

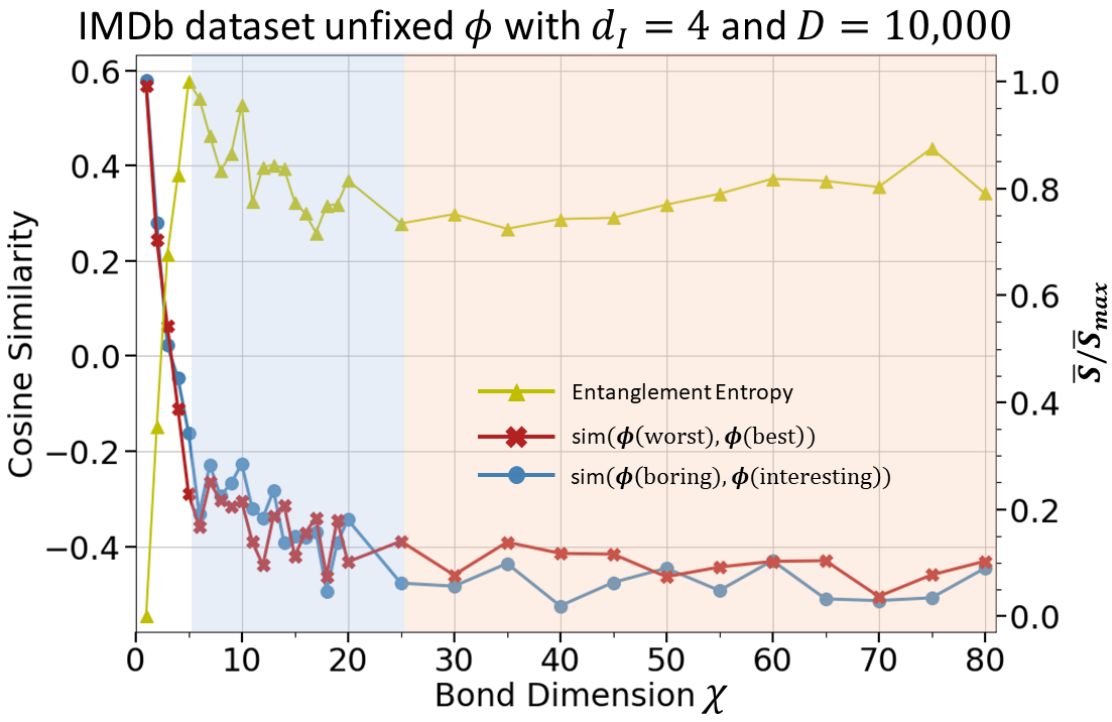


Figure 3.9: This figure shows the plots of entanglement entropy and cosine similarity of the opposite meanings of word pairs (worst, best) and (boring, interesting) on the IMDb dataset. The cosine similarity rapidly decreases while the entropy increases, which is intuitive since they should have an anti-correlation.

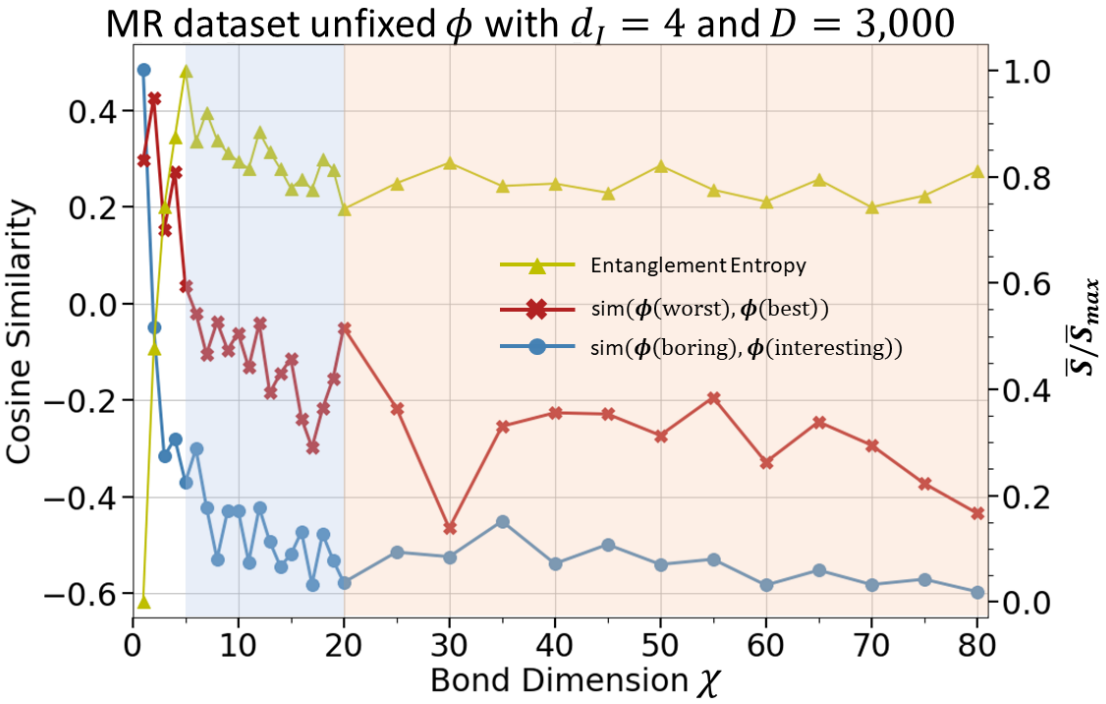


Figure 3.10: The plots show the cosine similarity and entropy of the model on the MR dataset.

Discussion and Outlook

In this work, we experiment on the RACs with additive biases and the effect of the word embedding by dividing our RACs training process into two strategies. The first strategy is to train RACs with the pre-trained and fixed word embedding; the second is to simultaneously train RACs and word embedding. In the former case, our results have shown that the entanglement entropy saturates when the model accuracy saturates. This saturation behavior points out the minimal bond dimensions (and hence resource) required to achieve the highest model accuracy, which the traditional RNN cannot determine. We notice that although the entanglement entropy is bounded by $\log_2(\chi + 1)$, $\bar{S}_{\max}$ is still less than the bounded value at any given χ indicating that the word embedding might play a crucial role in attaining higher accuracy and also contribute to the embedding layer. This leads us to investigate further in the latter case. By training the embedding layer and RACs concurrently, we find that the expressiveness and information propagation of the model stem from the RACs at the beginning before the role is switched to the word embedding and the entanglement entropy of the RACs starts to drop. We also notice that the latter case can achieve higher accuracy and have lower maximum entanglement entropy than the former. For the NLP sentiment analysis task on a movie review, we might not necessarily focus solely on designing RNNs to increase information propagation. Instead, the embedding layer should also be focused on achieving high accuracy.

4.1 Outlook

Although the results suggest that the word embedding affects the entanglement entropy, we cannot measure them directly. Instead, we have to find a way around it by using the cosine similarity. The problem is that the dimension of the bond dimension between input and the word embedding is too large (3,000 and 10,000 in our case), which cannot be performed by a normal computer. For example, if the MPS chain is chosen to contain 10 tensors when the MPS is constructed, the contained parameters go up to $3,000^{10}$. This problem can be solved by designing the word embedding to

contain arbitrary parameters without affecting the input and output dimensions. However, these works require too much effort, and it is not the aim of this project. It would be great to investigate further developing such a word embedding. Some of the ideas are that we can have two-word embedding instead of one so that the bond dimension connecting one to the other can be tuned without affecting the input and output size of the embedding layer.

Another important thing to consider is the stability of the model. Since the model comprises many multiplications, some deviation or perturbation (from stochastic gradient descent) can lead to exploding gradient. Hence, improvement is needed to use RACs with a larger model. We may need to adjust the activation function by using a more stable operator such as addition. Lastly, it would be great to expand the experiment to the multi-layer RACs because the machine learning model in the real-world use cases is mostly deep. Studying deep RACs might give us insights to understand more about the interaction and the correlation of layers in deep learning.

We also experiment on a character prediction task in this work. Given a sequence of n words, the model can predict the $(n+1)^{\text{th}}$ word, which can be used to generate a whole sentence or a long text composing of several sentences. However, the structure of the model is a matrix product operator (MPO) rather than an MPS. The classical mutual information is the only reasonable quantity that can be measured in this case. Although one can directly compute mutual information from the MPO, it is a quantum version, which cannot be interpreted as the same as classical mutual information. Also, for the model to work well, the size of the bond dimension needs to be big. Thus, the requirement leads to the same problems mentioned in the sentiment analysis task: stability and computationally intensive.

REFERENCES

1. Silver, D., et al., *Mastering the game of Go with deep neural networks and tree search.* Nature, 2016. **529**: p. 484-489.
2. Krishnan, M., *Against Interpretability: a Critical Examination of the Interpretability Problem in Machine Learning.* Philosophy & Technology, 2020. **33**.
3. Verstraete, F. and J. Cirac, *Matrix product states represent ground states faithfully.* Physical Review B, 2005. **73**.
4. Verstraete, F. and J. Cirac, *Renormalization algorithms for Quantum-Many Body Systems in two and higher dimensions.* 2004.
5. Vidal, G., *Entanglement Renormalization.* Physical Review Letters, 2007. **99**(22): p. 220405.
6. Stoudenmire, E. and D. Schwab, *Supervised Learning with Quantum-Inspired Tensor Networks.* 2016.
7. Levine, Y., et al., *Quantum Entanglement in Deep Learning Architectures.* Physical Review Letters, 2019. **122**(6): p. 065301.
8. Biamonte, J. and V. Bergholm, *Tensor Networks in a Nutshell.* 2017.
9. Jordan, M.I., *Chapter 25 - Serial Order: A Parallel Distributed Processing Approach*, in *Advances in Psychology*, J.W. Donahoe and V. Packard Dorsel, Editors. 1997, North-Holland. p. 471-495.
10. Hochreiter, S. and J. Schmidhuber, *Long Short-term Memory.* Neural computation, 1997. **9**: p. 1735-80.
11. Goodfellow, I., Y. Bengio, and A. Courville, *Deep Learning*. 2016: MIT Press.
12. Mehta, P., et al., *A high-bias, low-variance introduction to Machine Learning for physicists.* Physics Reports, 2018. **810**.
13. Mikolov, T., et al., *Distributed Representations of Words and Phrases and their Compositionality.* Advances in Neural Information Processing Systems, 2013. **26**.
14. Luo, C., et al. *Cosine Normalization: Using Cosine Similarity Instead of Dot Product in Neural Networks*. in *Artificial Neural Networks and Machine Learning – ICANN 2018*. 2018. Cham: Springer International Publishing.
15. Levine, Y., et al., *Deep Learning and Quantum Entanglement: Fundamental Connections with Implications to Network Design.* CoRR, 2017. **abs/1704.01552**.
16. Ekert, A. and P.L. Knight, *Entangled quantum systems and the Schmidt decomposition.* American Journal of Physics, 1995. **63**: p. 415-423.
17. Barata, J., et al., *Pure and mixed states*. 2019.
18. Mittelstaedt, P., *Ignorance Interpretation of Quantum Mechanics*, in *Compendium of Quantum Physics*, D. Greenberger, K. Hentschel, and F. Weinert, Editors. 2009, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 305-306.
19. Maas, A., et al., *Learning Word Vectors for Sentiment Analysis*. 2011. 142-150.
20. Chollet, F. and Others, *Keras.* 2015.
21. Pang, B. and L. Lee, *Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales.* ACL, 2005.

VITA

NAME	ชนาธิป มั่งคั่ง
DATE OF BIRTH	2 สิงหาคม 2539
PLACE OF BIRTH	นนทบุรี
INSTITUTIONS ATTENDED	ปริญญาตรี
HOME ADDRESS	61 หมู่ที่ 3 ต.หนองแสง อ.ปากพลี จ.นครนายก 26130