UNIVERSITY OF COPENHAGEN
DEPARTMENT OF COMPUTER SCIENCE

Tensor Networks for Language Modeling

Yuqin Zhou

Supervisor: Prof. Jakob Grue Simonsen

Thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN IT AND COGNITION

September 2023

行路难，行路难！多歧路，今安在?
长风破浪会有时，直挂云帆济沧海。

李白 (Li Bai), 744

Acknowledgements

Writing this thesis has been a joyful journey in sustained 'suffering'. Thanks to my supervisor, Prof. Jakob Grue Simonsen, for his professionalism and warmest guidance. It has been an honor being his student. Thanks to my friend, Zhan Su, for listening to my unrealistic research ideas without complaint. His suggestions have enriched this thesis. Thanks to my family—Father, Mother, Grandfather, and Grandmother—for unwavering support and understanding. This thesis is dedicated to them.

Abstract

Modern neural networks have shown success in language modeling tasks. Tensor networks (TNs) have been used to unravel their theoretical properties, albeit without concrete empirical support: no TN-like language model has proven effective on widely accepted natural language processing datasets. This thesis fills this gap by presenting a class of tensor-train language models (TTLMs). TTLMs encode the joint probability distribution of sequences into a wave function and operate on conditional probability distributions during training and inference. Theoretically, we demonstrate that the architectures of Second-order Recurrent Neural Networks (RNNs), Recurrent Arithmetic Circuits, and Multiplicative Integration RNNs are essentially special cases of TTLMs. Experimental evaluations reveal that the theoretical properties of TNs—linearity, multiplicativity, and complex-valued representations—cause difficulties during training. We employ statistical methods, such as the Kolmogorov-Smirnov test, to identify that the issues are unstable gradient flows, exponential decay in hidden states, and complex number multiplications. To enhance the effectiveness of TTLMs, we propose a class of variants called the Linear Multiplicative Models (LMMs). LMMs are linear, multiplicative, and complex-valued, achieving competitive results on language modeling tasks compared to vanilla RNNs. Their success provides empirical support for prior research that establishes the equivalence between TNs and neural networks, and offers a novel perspective in the field dominated by nonlinear and additive language models.

Contents

1 Introduction

The interest around *language modeling* has surged in these years thanks to the large amount of data available and the swift increase of computing performance (Min *et al.*, 2021; Bommasani *et al.*, 2021; Han *et al.*, 2021). It is the fundamental task and long-standing research topic in natural language processing (NLP) (Rosenfeld, 2000; Bommasani *et al.*, 2021; Wei *et al.*, 2023). Language models (LMs) find applications in many computational linguistic problems such as speech recognition (Kuhn and De Mori, 1990; Jelinek, 1998), machine translation (Brown *et al.*, 1993; Schwenk *et al.*, 2006), and semantic parsing (Deschacht *et al.*, 2012; Andreas *et al.*, 2013). More recently, the development of large-scale LMs has permeated various domains such as healthcare (Wang *et al.*, 2023; Yang, Zhao, *et al.*, 2023), finance (Wu *et al.*, 2023; Yang, Liu, *et al.*, 2023), and law (Cui *et al.*, 2023). Language modeling consists of developing algorithmic and engineering strategies for automatic text generation. From a mathematical point of view, widely different models have been employed and further developed, from n-gram LMs (Shannon, 1951; Bahl *et al.*, 1983; Church, 1989), to neural networks (Bengio *et al.*, 2000; Mikolov *et al.*, 2010; Vaswani *et al.*, 2017; Brown *et al.*, 2020; Thoppilan *et al.*, 2022). Their major role is to learn the probability distributions over a sequence of linguistic units (Rosenfeld, 2000; Jurafsky, 2000; Wei *et al.*, 2023).

Despite their success across a wide swath of NLP tasks, neural network LMs are often used as black boxes (Belinkov and Glass, 2019; Creswell *et al.*, 2022; Yin and Neubig, 2022). Prior research uses *tensor networks* (TNs; Orús, 2014) to analyze the theoretical properties of neural network models. A better understanding of feed-forward, convolutional and recurrent architectures has been gained, including parameters compression (Novikov *et al.*, 2015), expressive power (Cohen *et al.*, 2016; Khrulkov *et al.*, 2018), and depth efficiency for long-term memory (Levine *et al.*, 2018). TNs have also garnered attention as a model architecture to capture linguistic information in sequences. Theoretical proposals by Pestun *et al.* (2017) and Pestun and Vlassopoulos (2017) seek to leverage

TNs to capture *long-range correlations* (Tanaka-Ishii, 2021) in human language that challenge the ability of neural network LMs (e.g., LSTMs; Hochreiter and Schmidhuber, 1997). Neither of them has an experimental evaluation of their models. Moving forward, several researchers have adopted the simplest TN, specifically tensor trains (Oseledets, 2011), as their proposed architecture with various motivations (Han *et al.*, 2018; Stokes and Terilla, 2019; Novikov *et al.*, 2021; Miller *et al.*, 2021). Instead of language modeling tasks, most of these models focus on solving *density estimation* problem (Silverman, 1986), which is learning the *empirical distribution* (Tang *et al.*, 2022) defined by the training set.

In this thesis we introduce TTLMs, a class of language models using tensor trains (Oseledets, 2011) as their architecture. Specifically, TTLMs encode the joint probability distribution of sequences into a wave function. The measurement of the wave function is the inner product of two same-sized high-dimensional tensors: the input data $\Phi(X)$ and global coefficients $\mathcal{A}$. TTLMs use Tensor Train decompositions (Oseledets, 2011) to approximate $\mathcal{A}$ in low-dimensional spaces to avoid curse-of-dimensionality (Bellman, 1966). As an original **contribution** of this thesis, we present two propositions, accompanied by their proofs, that demonstrate how TTLMs compute the joint probability of sequences by its conditional probability distributions during training and inference. Unlike prior tensor network LMs, the training paradigm of TTLMs aligns with neural network LMs, paving a critical step toward using TNs to tackle real-world language modeling tasks.

We conduct a comprehensive evaluation of TTLMs, examining their connections to previous models and theoretical properties. First, we clarify the relationship between TTLMs and a series of recurrent neural networks (RNNs): Second-order RNNs (Goudreau *et al.*, 1994), Recurrent Arithmetic Circuits (RACs; Levine *et al.*, 2018), and Multiplicative Integration RNNs (Wu *et al.*, 2016). These connections open a new eye to understanding RNNs and give some natural implementations for TTLMs. Second, using RACs as an example, we show that the theoretical properties of TNs (i.e., linearity, multiplicativity, and complex-valued representations) cause difficulties during training. Our statistical analysis reveals that the contributing factors are unstable gradient flows, exponential decay in hidden states, and the distribution of complex number multiplications.

To further enhance the effectiveness of TTLMs, we devise a class of variants called the *Linear Multiplicative Models* (LMMs). With H-dimensional hidden states, the total number of trainable parameters in the recurrent layer is reduced from $2H^2$ in vanilla RNNs to $4H$. More important, when combined with a linear normalization layer, such as Min-Max normalization (Sahu, 2015), LMMs show competitive effectiveness on language modeling tasks compared with vanilla RNNs. These findings underscore the feasibility of TNs in the realm of language modeling.

The original contributions of the thesis can be summarized as follows:

- We present a historical review of tensor network LMs, clarify their research objectives, and conclude their training paradigm. This work represents a pioneering effort to untangle and assess this research domain.

- We propose a class of tensor-train language models (TTLMs). The theoretical relationship between TTLMs and existing RNNs is clarified, and statistical methods evaluate the impact of theoretical properties of TNs—linearity, multiplicativity, and complex-valued representations—on model effectiveness.

- We propose a complex-valued class of TTLMs (i.e., LMMs) that can achieve competitive results on language modeling tasks compared with vanilla RNNs. LMMs inherit the fundamental properties of TNs and their success demonstrates the feasibility of TNs in language modeling tasks, providing empirical support for prior research that establishes the connections between TNs and neural networks.

The rest of the thesis is organized as follows: **(1)** Ch. 2 outlines the evolution of LMs, setting the general background for discussing tensor network LMs. **(2)** Ch. 3 presents the notation, diagrams, and TNs to cater to readers with different background knowledge. **(3)** Ch. 4 reviews the development of tensor network LMs, including their research objectives, paradigm, limitations, and potential enhancements. We also review recent advancements in RNNs, since they have similar objectives and mathematical relationships with our models. **(4)** Ch. 5 presents our theoretical innovations: a class of tensor-trains

LMs (i.e., TTLMs) and its variant (i.e., LMMs). **(5)** Ch. 6 introduces our experimental evaluation of our proposed models, with a specific focus on their theoretical properties, including linearity, multiplicativity, and complex numbers. **(6**) Discussion and future research directions are pointed out in Ch. 7, before concluding in Ch. 8.

2

Background: Evolution of Language Modeling

This chapter outlines the evolution of language modeling to set a good foundation for discussing tensor networks language models (LMs) in Ch. 4 and our proposed models in Ch. 5. The organization of this chapter is as follows: We introduce the basic concepts and terminology of language modeling in Sec. 2.1. Moving on, we discuss traditional LMs, as exemplified by n-gram LMs in Sec. 2.2. Subsequently, we introduce neural network LMs, as exemplified by Recurrent Neural Networks (RNNs) (Mikolov *et al.*, 2010) and state-of-the-art LMs in Sec. 2.3. Last, we present the evaluation metric for LMs in Sec. 2.4.

By the end of this chapter, readers will grasp a basic understanding of language modeling. For those readers who seek a comprehensive and updated overview of the field, we recommend the work of Wei *et al.* (2023).

2.1 Basic Concepts and Terminology

We begin by introducing basic concepts that will be used throughout the thesis (see Table 3.1 for the notation). We denote a sequence of $L \in \mathbb{N}_+$ discrete random variables as $\mathbf{X} = [X^{(1)}, X^{(2)}, \cdots, X^{(L)}]$, where each $X^{(i)} \in V$ for all $i \in [L]$. The notation V refers to the *vocabulary* (i.e., the non-empty set of *types* used to construct the training set). The term *types* refers to distinct words; in contrast, the term *tokens* refers to all occurrences of words, including repeats. The *vocabulary size* refers to the number of types, denoted as $|V| \in \mathbb{N}_+$. Each type is typically treated as a random variable in LMs, even if some of them share the same *stem* (e.g., *books* and *book*) (Jurafsky, 2000).

The probability distribution of $\mathbf{X}$ is denoted as $P : V^L \rightarrow \mathbb{R}_{[0,1]}$. Let $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$ be a sample according to $\mathbf{X}$. We use $P(\mathbf{x})$ to represent the probability mass function that takes the value $\mathbf{x}$, which is short for $P\left(X^{(1)} = x^{(1)}, X^{(2)} = x^{(2)}, \cdots, X^{(L)} = x^{(L)}\right)$. Language modeling studies the probability distributions over a linguistic sequence, such as words, sentences, and whole documents (Rosenfeld, 2000; Jurafsky, 2000; Wei *et al.*, 2023). To establish LMs, the initial and fundamental step is converting the sequence $\mathbf{x}$ into numeric values that computers can process, allowing for the identification and comprehension of linguistic patterns. This process is commonly known as *text representation* and can be broadly classified into two groups: *discrete representation* and *continuous representation* (Cartuyvels *et al.*, 2021).

To ensure clarity, we define the two representations here. **(1)** The term *discrete representation* refers to a variable that takes on a finite or countably infinite number of values to represent some concept (Cartuyvels *et al.*, 2021). In the context of language modeling, this term means that each type in a dataset is represented as a discrete random variable. For example, the value of a variable can be the frequency of the corresponding type or a binary number, depending on whether the type appears or not. **(2)** The term *continuous representation* denotes a real-valued variable, taking on values in the Euclidean space $\mathbb{R}^D$ where $D \in \mathbb{N}_+$ is a specific dimension dependent on the model being used (Cartuyvels *et al.*, 2021). In language modeling, this term means that each type is represented by a real-valued vector, commonly known as its *word embedding*.

The difference between the two representations reflects a scientific paradigm shift in language modeling. During this shift, LMs have undergone significant changes, including their structures, training techniques, evaluation metrics, and applications (Wei *et al.*, 2023). This shift can be easily discerned in the timeline of LMs. The most well-known traditional LMs, n-gram LMs, represent text as discrete variables; n-gram models were first applied for English word sequences by Shannon (1948) and have been widely used in the 1990s (Bahl *et al.*, 1983; Jelinek, 1990; Church, 1989). Soon the first feed-forward neural network LMs was proposed by Bengio *et al.* (2000). Since then, the representation of text in LMs has undergone a significant shift; the widely cited work of Mikolov, Chen, *et al.* (2013) signifies the era of representing text as real-valued vectors. From 2018 to the present day,

pre-trained neural network LMs have quickly dominated the field, such as GPT-3 (Brown *et al.*, 2020) and LaMDA (Thoppilan *et al.*, 2022).

In the subsequent two sections, we will introduce traditional LMs with discrete representations and neural network LMs with continuous representations.

2.2 Discrete Representations

We introduce the most commonly used traditional LMs using discrete representations, namely n-gram LMs, in Sec. 2.2.1. Then, we outline the limitations of traditional LMs in Sec. 2.2.2.

2.2.1 n-gram Language Models

The n-gram LMs follow the *Markov assumption* and use the *chain rule of probabilities* (Jurafsky, 2000). The Markov assumption states that the probability of a word depends only on a fixed number of preceding words, rather than the entire history of the text (Markov, 2006). The chain rule of probability (Bahl *et al.*, 1983) decomposes the joint probability of words $P(\mathbf{x})$ as:

$$\begin{aligned} P(\mathbf{x}) &= P(x^{(1)})P(x^{(2)}|x^{(1)})\cdots P(x^{(L)}|x^{(1)},x^{(2)},\cdots,x^{(L-1)}), \\ &= \prod_{t=1}^{L} P(x^{(t)}|x^{(1:t-1)}), \end{aligned} \tag{2.1}$$

where $x^{(1:t-1)}$ denotes the sequence $[x^{(1)},x^{(2)},\cdots,x^{(t-1)}]$; and $P(x^{(t)}|x^{(1:t-1)}) \in \mathbb{R}_{[0,1]}$ is shorthand for $P(X^{(t)}=x^{(t)}|X^{(1:t-1)}=x^{(1:t-1)})$, which denotes the conditional probability distribution of $X^{(t)}$ given $X^{(1:t-1)}=x^{(1:t-1)}$. We can compute the probability of a sequence by multiplying its conditional probabilities in Eq. 2.1.

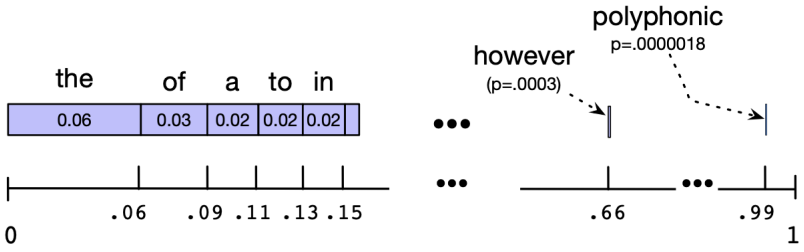

Figure 2.1 A visualization of the sampling distribution for sampling sentences by repeatedly sampling unigrams ($n = 1$) (Jurafsky, 2000). The blue bar represents the frequency of each word in the training set. The number line shows the cumulative probabilities. If we choose a random number between 0 and 1, it will fall in an interval corresponding to some word. The expectation for a random number to fall in the larger intervals of frequent words (e.g., *the, of, a*) is higher than in the smaller interval of rare words (e.g., *polyphonic*).

To compute the conditional probabilities, n-gram LMs consider the probability of a word given only its $n-1$ preceding words, instead of all preceding words, making the following assumption:

$$P(x^{(t)}|x^{(1:t-1)}) = P(x^{(t)}|x^{(t-n+1,t-1)}). \tag{2.2}$$

Then n-gram LMs compute the probability of a sequence by substituting Eq. 2.2 into Eq. 2.1 (Jurafsky, 2000):

$$P(\mathbf{x}) = \prod_{t=1}^{L} P(x^{(t)}|x^{(t-n+1,t-1)}), \tag{2.3}$$

where $P(x^{(t)}|x^{(t-n+1:t-1)})$ is computed by the method of *maximum likelihood estimation* given a training set and normalized as follows (Jurafsky, 2000):

$$\tilde{P}(x^{(t)}|x^{(t-n+1:t-1)}) = \frac{\text{Count}(x^{(t-n+1:t)})}{\text{Count}(x^{(t-n+1:t-1)})}, \tag{2.4}$$

where $\tilde{P}(\cdot)$ denotes the probability probabilities defined by the model, distinguishing them from the ground-truth probabilities $P(\cdot)$; $\text{Count}(x^{(t-n+1:t)})$ is the number of n-grams (i.e., $x^{(t-n+1:t)}$); and $\text{Count}(x^{(t-n+1:t-1)})$ is the number of (n-1)-grams (i.e., $x^{(t-n+1:t-1)}$).

An n-gram LM contains a probability distribution defined by itself after computing Eq. 2.4 on the training set. For example, Fig. 2.1 visualizes the probability distribution defined by unigram LMs ($n = 1$). This technique of visualizing an LM by sampling was first suggested very early on by Shannon (1951) and Miller and Selfridge (1950).

2.2.2 Limitations of Traditional Language Models

The simplicity of n-gram LMs is shown in Eq. 2.4, where they tabulate the frequency of n-grams in the training set. Despite the straightforwardness of this approach, which has proven to be a key driver of the effectiveness of n-gram LMs (Brants *et al.*, 2007), it has raised crucial issues that must be addressed.

One fundamental issue is the sparsity problem in n-gram LMs, stemming from their way of modeling the joint probability of sequences $P(\mathbf{x})$. As the length of a sequence (i.e., $L \in \mathbb{N}_+$) increases, the number of possible distinct sequences increases exponentially; this growth results in a large and complex search space, V^L, for LMs, leading to the *curse of dimensionality* (Bellman, 1966).

Despite n-gram LMs truncating a sequence into multiple n-grams, as the value of n increases, n-gram LMs still require an exponential increase in the dataset size. For example, if the vocabulary size of a dataset reaches 5000, there are $|V|^2 = 2.5 \times 10^7$ bigrams and $|V|^4 = 6.25 \times 10^{14}$ 4-grams. Thus, the exponential growth of the search space V^n leads to sparse n-grams; for example, an n-gram that appears in the training set may not appear in the test set (Bengio *et al.*, 2000). Consequently, n-gram LMs assign the probability of that n-gram as zero at test time, when multiplied with the other probabilities in Eq. 2.3, leading to the probability of the entire sequence being zero. Many techniques have been proposed to alleviate the issue of sparsity. For example, *smoothing* intends to properly shave off a bit of probability mass from some more frequent n-grams and give it to the n-grams that do not occur during training (Kneser and Ney, 1995; Chen and Goodman, 1999; Moore and Quirk, 2009).

Rather than relying on the n-grams alone, more traditional LMs using discrete representation were proposed, including models based on decision trees (Potamianos and Jelinek, 1998; Heeman, 1999) and maximum entropy-based techniques (Rosenfeld, 1994; Peters and Klakow, 1999; Wang *et al.*, 2005). These LMs typically incorporated manually engineered linguistic features such as part-of-speech tags. However, they could not essentially address the drawback associated with discrete representations (Bengio *et al.*, 2000). Specifically, when each discrete variable takes a large value,

most observed objects are almost maximally far from each other in the Hamming distance. Any change of these discrete variables may drastically impact the value of the function $P(\mathbf{x})$ to be estimated.

2.3 Continuous Representations

Neural network LMs learn *continous representation* of text by representing words as real-valued vectors (Hinton, 1984). The property is a defining feature that sets neural network LMs apart from traditional LMs (Mikolov, Yih, *et al.*, 2013). A critical advantage of real-valued vectors is that they can encapsulate word meaning and context, as evidenced by words with similar contexts having similar vector distributions (Pennington *et al.*, 2014). For example, Chen *et al.* (2013) showed that the vector representations capture word relations such as synonyms, antonyms, and regional spelling variations.

Neural network models have distinct architectures. Each architecture often accompanies a distinct optimization algorithm during training and a sampling method to generate new examples during testing. Notable architectures encompass feedforward-based (Bengio *et al.*, 2000), recurrent-based (Mikolov *et al.*, 2010), and transformer-based (Vaswani *et al.*, 2017). For example, recurrent-based models include the vanilla version of RNNs (Vanilla-RNNs; Mikolov *et al.*, 2010), Long Short-term Memory Networks (LSTMs; Hochreiter and Schmidhuber, 1997), and Gated Recurrent Units (GRUs; Chung *et al.*, 2014).

We organize this section in two aspects. In Sec. 2.3.1, we use Vanilla-RNNs as an example to introduce the continuous representations in neural network LMs. In Sec. 2.3.2, we shortly describe state-of-the-art LMs, providing an overview of the current paradigm in neural network LMs.

2.3.1 Recurrent Neural Network Language Models

In this section, we delve into the vanilla version of RNNs (Vanilla-RNNs) in-depth, introducing its model architecture, training process, and sampling method. This detailed introduction is necessary because certain RNNs hold mathematical relevance to our proposed models, as specified in Sec. 5.3.2.

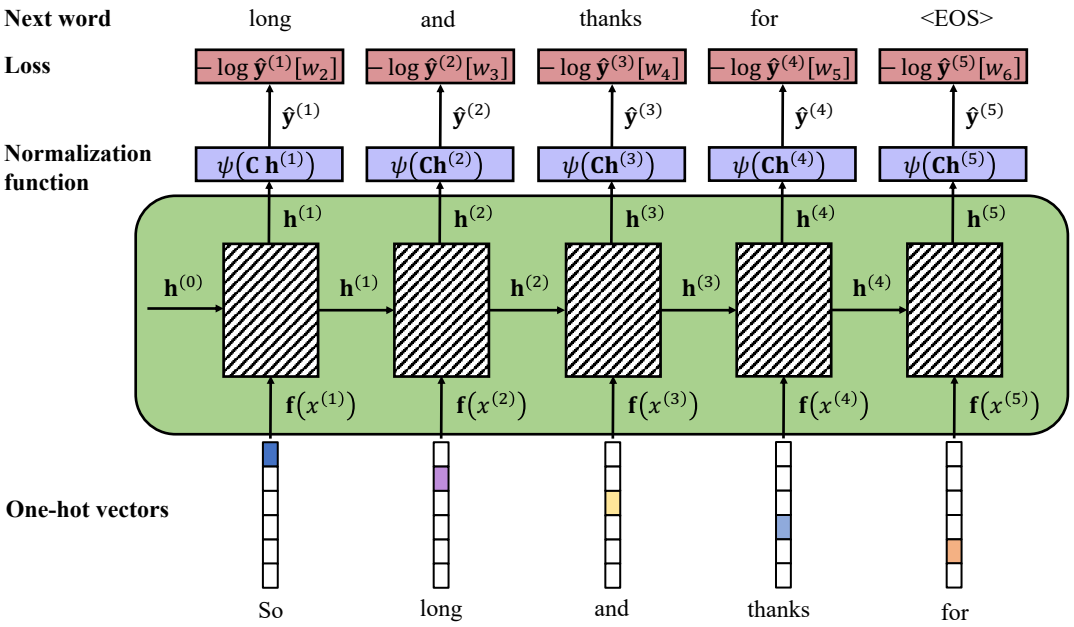

Figure 2.2 A toy example to show how Vanilla-RNNs predict the next word at time t during training (Jurafsky, 2000). The sequence is `[so, long, and, thanks, for, <EOS>]`, where <EOS> is a special token used to denote the end of the sequence (Here, $|V| = 6$). The rectangles with diagonal lines denote the computation of Eq. 2.5. Note that one-hot vectors $\mathbf{f}(x^{(t)})$ are commonly replaced by pre-trained word embeddings such as GolVe (Pennington *et al.*, 2014) in practice. The normalization function ψ is typically the `softmax` function. The initial hidden state $\mathbf{h}^{(0)}$ is typically the last hidden state of the previous sequence or initialized from the uniform distribution $\mathcal{U}(-\sqrt{k}, -\sqrt{k})$ where $k = \frac{1}{\text{number of hidden units}}$.

Thus, we include this material as a reference for readers who may not be well-versed in the deep learning community's terminology and concepts.

Model Architecture

Vanilla-RNNs process the input sequence one word at a time, attempting to predict the next word from the current word and previous hidden state as depicted in Fig. 2.2, which is defined as (Mikolov *et al.*, 2010):

$$\mathbf{h}^{(t)} = \phi(\mathbf{A}\mathbf{h}^{(t-1)} + \mathbf{B}\mathbf{f}(x^{(t)}) + \mathbf{b}), \tag{2.5}$$

where

- $\mathbf{h}^{(t)} \in \mathbb{R}^H$ is the *hidden state* of Vanilla-RNNs at time step t;

- $\mathbf{A} \in \mathbb{R}^{H \times H}$ is the *weight matrix* for the hidden-to-hidden connections, and $\mathbf{B} \in \mathbb{R}^{H \times |V|}$ is the weight matrix for the input-to-hidden connections;

- ϕ denotes a *nonlinear* activation function, often chosen to be a sigmoid or tanh activation;

- $\mathbf{b} \in \mathbb{R}^{H}$ is the *bias term* for the hidden state;

- $\mathbf{f}(x^{(t)})$ is one-hot encoding, which outputs a vector that has a single element that is equal to 1 and 0s in all other positions. The position of the "1" element in the vector corresponds to the index of the word $x^{(t)}$ in the vocabulary. In addition to one-hot vectors, the inputs can be word embeddings or intermediate layer computations that keep the sequential structure.

Once the hidden state has been updated, the output $\hat{\mathbf{y}}^{(t)} \in \mathbb{R}^{|V|}$ at time step t is defined as:

$$\hat{\mathbf{y}}^{(t)} = \psi(\mathbf{C}\mathbf{h}^{(t)}), \tag{2.6}$$

where $\mathbf{C} \in \mathbb{R}^{|V| \times H}$ is a weight matrix for the hidden-to-output connections; and ψ is often a nonlinear normalization function such as `softmax`, ensuring that components in $\hat{\mathbf{y}}^{(t)}$ are non-negative and sum to 1. The output $\hat{\mathbf{y}}^{(t)}$ is typically viewed as $P_{x^{(1:t-1)}} : V \to \mathbb{R}_{[0,1]}$ (i.e., the conditional probability distribution of the random variable $X^{(t)}$ given $X^{(1:t-1)} = x^{(1:t-1)}$ defined by the model (Jurafsky, 2000)).

Vanilla-RNNs possess a search space of V at each time step as shown in Eq. 2.6, unlike n-gram LMs having a search space of V^n. This characteristic mitigates the curse of dimensionality, enabling Vanilla-RNNs to manage longer sequences without the need to truncate them into n-grams. More important, the long sequence modeling capability of Vanilla-RNNs is empowered by recursively encoding the information of words and their previous content into real-valued vectors. Each word in the vocabulary corresponds to a column of $\mathbf{B}$, and the hidden state is designed to encapsulate some information about all preceding words from the beginning of the sequence. These real-valued vectors can partly capture the semantic and syntactic characteristics of sequences (Pennington *et al.*, 2014), making them a fundamental building block for developing neural network LMs.

Training Process

During training, Vanilla-RNNs need a *loss function* to gauge the error between the prediction output $\hat{\mathbf{y}}^{(t)}$and provided target value $\mathbf{y}^{(t)}$. Formally, suppose the indices of words in the sequence $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$ is $w_1, w_2, \cdots, w_L$, where $w_i \in [|V|] \coloneqq \{1, 2, \cdots, |V|\}$, the estimated probability of a particular word $x^{(t)}$ in the vocabulary being the next word is the tth component of $\hat{\mathbf{y}}^{(t)}$ (denoted as $\hat{\mathbf{y}}^{(t)}[w_t]$). Meanwhile, in language modeling, the groud-truth distribution $\mathbf{y}^{(t)}$ is assumed to be an *one-hot vector* having $\mathbf{y}^{(t)}[w_t] = 1$ and all the remaining $|V| - 1$ components being 0 (Jurafsky, 2000).

Vanilla-RNNs typically use *cross entropy* as the loss function $\mathcal{L}_{\text{CE}}$ at time t (Jurafsky, 2000):

$$\begin{aligned} \mathcal{L}_{\text{CE}} &= -\sum_{w \in [|V|]} \mathbf{y}^{(t)}[w] \log \hat{\mathbf{y}}^{(t)}[w] \\ &= -\log \hat{\mathbf{y}}^{(t)}[w_{t+1}]. \end{aligned} \tag{2.7}$$

When minimizing $\mathcal{L}_{\text{CE}}$ in Eq. 2.7, we are minimizing the negative log likelihood (NLL) of the probability the model assigns to the next word in the training sequence $\hat{\mathbf{y}}^{(t)}[w_{t+1}]$. Apart from the cross-entropy loss, we can use other loss functions in Vanilla-RNNs. Selecting an appropriate loss function is an important research topic in machine learning. Readers who wish to explore this topic further may refer to Wang *et al.* (2020), who have compared 31 classical loss functions.

Vanilla-RNNs are often trained using *teacher forcing* (Marcus, 1998). This technique provides the model with the correct historical sequence of words to predict the next word, instead of words predicted by the model. Specifically, during training, Vanilla-RNNs use the correct sequence of words $x^{(1:t)}$ to estimate the probability of the next word $x^{(t+1)}$, ignoring the output predicted by the model. Teacher forcing has also been widely used in Transformer-based models during training (Devlin *et al.*, 2018); and we will employ it in our proposed models in Ch. 5.

After selecting the loss function, Vanilla-RNNs use a two-pass algorithm for optimizing parameters. In the forward phase, they compute $\hat{\mathbf{y}}^{(t)}$, accumulate the loss $\mathcal{L}_{\text{CE}}$ at each step in time, save the value

of the hidden layer at each step for use at the next time step. In the backward phase, Vanilla-RNNs process the sequence in reverse, computing the required gradients, computing and saving the error term for use in the hidden layer for each step backward in time (Jurafsky, 2000). This two-pass approach is commonly referred to as the *Backpropagation Through Time* (BPTT) (Werbos, 1974; Rumelhart *et al.*, 1985; Werbos, 1990). BPTT is a particular type of *Backpropagation* algorithm (Rumelhart *et al.*, 1986) applied to the sequence data such as the time series. Most neural network models rely on the backpropagation algorithm to update their parameters, with gradients computed through this process (Hinton, 2022).

Sampling Method

Within the probabilities defined by an LM, we can use sampling techniques to generate new sequences during test time. Sampling from an LM refers to using it to generate new sequences based on its likelihood as defined by themselves (Jurafsky, 2000). For example, we can use the most commonly used approach, *autoregressive generation*, to generate sequences from Vanilla-RNNs. Specifically, we sample a word in $\hat{\mathbf{y}}^{(1)}$ from using the beginning of sentence marker, <BOS>, as the first input and use the word embedding for that first word as the input to the network at the next time step. Then, we sample the next word in the same way until the end of sentence marker, <EOS>, is sampled or a fixed length limit is reached (Jurafsky, 2000).

2.3.2 State-of-the-art Language Models

Since the advent of RNNs, the field of language modeling has seen a significant amount of development. Some models no longer follow the Markov assumption and chain rule, such as the Transformer architecture (Vaswani *et al.*, 2017) employing self-attention mechanisms to learn dependencies between words in a sequence, regardless of their position. More recently, large pre-trained Transformer-based LMs, such as the BERT (Devlin *et al.*, 2018) and GPT (Radford *et al.*, 2018) families of models, have achieved state-of-the-art performance on many tasks. These models with billions of parameters, trained

on vast amounts of data, have revolutionized the field and continue to drive breakthroughs in natural language understanding (Min *et al.*, 2021).

This section introduces these large pre-trained LMs' naming, categories, and applications. We only briefly touch upon these large LMs, as they are not directly relevant to our research goal of developing a novel LM with small-scale parameters. We recommend referring to Bommasani *et al.* (2021) for readers interested in this topic.

There is yet to be a universally accepted name or definition for the state-of-the-art LMs developed since 2018. For example, some people refer to them as the *large language models* (LLMs) (Carlini *et al.*, 2021), while others call them the *foundation models* (Bommasani *et al.*, 2021). The term *LLMs* emphasizes the technical aspect of these models, which involves neural network LMs with billions of parameters or more that have been pre-trained on vast amounts of unlabeled text using *self-supervised learning* techniques. The term *foundation models* highlights the sociological impact of these models and how they have conferred a broad shift in AI research and deployment (Bommasani *et al.*, 2021). In later sections, we will use the term *LLMs* to refer to this group of LMs.

The architecture and training process of LLMs can be broadly divided into three classes (Min *et al.*, 2021) as depicted in Fig. 2.3: autoregressive LMs, masked LMs, and encoder-decoder models. (1) Autoregressive LMs predict the next word $x^{(i)}$ given all previous words $x^{(1:t-1)}$. Popular examples include GPT (Radford *et al.*, 2018), GPT-2 (Radford *et al.*, 2019) and GPT-3 (Brown *et al.*, 2020). (2) Masked LMs predict a "masked" word conditioned on all other words in the sequence. Representative models include BERT (Devlin *et al.*, 2018) and RoBERTa (Liu *et al.*, 2019). (3) The encoder-decoder models generate a sequence of words $[z^{(1)}, z^{(2)}, \cdots, z^{(M)}]$ given an input sequence $[x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$. Typical models include BART (Lewis *et al.*, 2019) and T5 (Raffel *et al.*, 2020).

LLMs have revolutionized the applications of language modeling. **(1)** Before 2018, downstream tasks mainly used LMs to generate linguistic units or evaluate their fluency. For example, applications of LMs mainly focused on speech recognition (Kuhn and De Mori, 1990; Jelinek, 1998), spelling correction (Ahmed *et al.*, 2009), text generation (De Novais *et al.*, 2010), machine translation (Brown

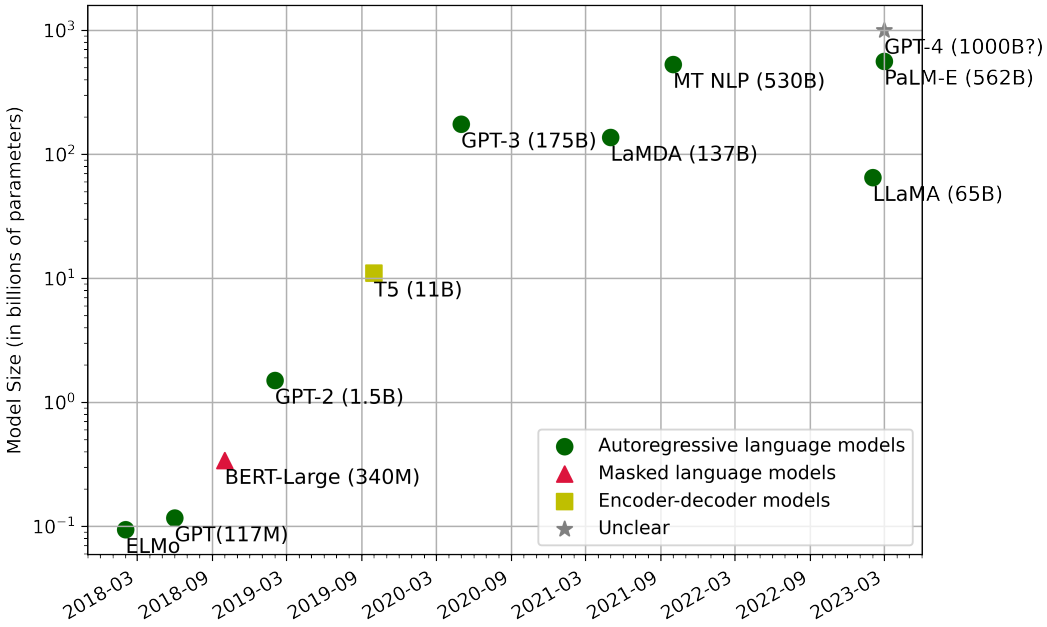

Figure 2.3 Comparing parameters of popular pre-trained language models: ELMO (Peters *et al.*, 2018), GPT (Radford *et al.*, 2018), BERT-Large (Devlin *et al.*, 2018), GPT-2 (Radford *et al.*, 2019), T5 (Raffel *et al.*, 2020), GPT-3 (Brown *et al.*, 2020), LaMDA (Thoppilan *et al.*, 2022), Megatron-Turing NLP (MT NLP) (Smith *et al.*, 2022), LLaMA (Touvron *et al.*, 2023), PaLM-E (Driess *et al.*, 2023), and GPT-4 (OpenAI, 2023). We extract the release date, number of parameters, and architecture of each model from its official website or earliest arXiv version. When writing this thesis, official information on GPT-4 regarding its architecture and number of parameters is unavailable.

et al., 1993; Schwenk *et al.*, 2006), semantic parsing (Deschacht *et al.*, 2012; Andreas *et al.*, 2013), and text summarization (Hermann *et al.*, 2015). **(2)** Since 2018, LLMs has been used as a powerful tool for tasks requiring high levels of text understanding (Jiang *et al.*, 2020; Bommasani *et al.*, 2021). For example, LLMs have been used to answer factoid questions (Radford *et al.*, 2019), answer common sense queries (Trinh and Le, 2018; Sap *et al.*, 2019), or extract factual knowledge about relations between entities (Petroni *et al.*, 2019; Soares *et al.*, 2019). Furthermore, the widely recognized model, GPT-4, has been used to solve novel and difficult tasks that span mathematics, coding, vision, medicine, law, and psychology (OpenAI, 2023). Since the effectiveness of GPT-4 is strikingly close to human-level, Bubeck *et al.* (2023) argued that it could reasonably be viewed as an early (yet still incomplete) version of an Artificial General Intelligence system.

2.4 Evaluation Metric: Perplexity

The performance of LMs can be evaluated extrinsically or intrinsically (Jurafsky, 2000). *Extrinsic evaluation* refers to applying LMs in a downstream task (e.g., machine translation) and measuring the extent to which the application is improved. By doing so, we can determine if a particular improvement to the LM truly benefits the application. However, performing end-to-end evaluations on large NLP systems can be prohibitively expensive. Therefore, an *intrinsic evaluation* metric is necessary to quickly assess potential improvements, which will be conducted independently of any downstream tasks or applications (Jurafsky, 2000). The most commonly used intrinsic metric is *perplexity* (PPL; Jelinek *et al.*, 1977), which we will employ in our experimental evaluation in Ch. 6. In this section, we provide an overview of its fundamental concepts.

Given the non-empty sequence $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$ in the test set, the *perplexity per word* computes the geometric mean of the inverse probability assigned by a model to each word, given previous words (Jurafsky, 2000):

$$\begin{aligned} \mathrm{PPL}(\mathbf{x}) &= \tilde{P}(\mathbf{x})^{-\frac{1}{L}} \\ &= \sqrt[L]{\prod_{t=1}^{L} \frac{1}{\tilde{P}\left(x^{(t)} \mid x^{(1)}, \cdots, x^{(t-1)}\right)}}. \end{aligned} \tag{2.8}$$

The metric is based on the principle that a model assigned higher probability to the test set is more accurate and better at predicting it. Thus, according to Eq. 2.8, lower perplexity scores mean higher conditional probabilities of sequences in the test set, which indicates more effective LMs.

The notion of the *perplexity per word* in Eq. 2.8 is defined from the perspective of natural language processing. This concept actually arises from the information-theoretic concept of cross entropy (Jurafsky, 2000). We now explain how the perplexity per word relates to the cross-entropy formula. Formally, let

us consider a language as a stochastic process ℓ that produces a sequence of words. The *cross-entropy rate* of ℓ (a.k.a per-word cross-entropy) is defined as (Jurafsky, 2000):

$$H(P,\tilde{P}) = \lim_{L\to\infty} -\frac{1}{L}\sum_{\mathbf{x}\in\ell} P(\mathbf{x})\log\tilde{P}(\mathbf{x}), \tag{2.9}$$

where P is the actual probability distribution that generated the data, and $\tilde{P}$ is the distribution defined by a model. The Shannon-McMillan-Breiman theorem (Algoet and Cover, 1988; Cover, 1999) states that if the language is stationary and ergodic (Jurafsky, 2000),

$$H(P,\tilde{P}) = \lim_{L\to\infty} -\frac{1}{L}\log\tilde{P}(\mathbf{x}). \tag{2.10}$$

That is, we can take a single sequence that is long enough instead of summing over all possible sequences when computing the cross-entropy rate (Jurafsky, 2000); the intuition behind the Shannon-McMillan-Breiman theorem is that a long-enough sequence of words will contain in it many other shorter sequences and that each of these shorter sequences will reoccur in the longer sequence according to their probabilities (Jurafsky, 2000). It is important to emphasize that natural language is not stationary (Jurafsky, 2000), as the probability of upcoming words can depend on arbitrarily distant and time-dependent events. Thus, Eq. 2.10 only gives an approximation to the correct distributions and entropies of natural language (Jurafsky, 2000).

Though the cross-entropy rate is defined in the limit as the length of the observed word sequence goes to infinity ($L \to \infty$), we often approximate it with a (sufficiently long) sequence of fixed length: $H(\mathbf{x}) = -\frac{1}{L}\log\tilde{P}(\mathbf{x})$ (Jurafsky, 2000). Thus, the cross-entropy rate can be linked to the perplexity per word:

$$\mathrm{PPL}(\mathbf{x}) = 2^{H(\mathbf{x})} = \tilde{P}(\mathbf{x})^{-\frac{1}{L}}. \tag{2.11}$$

In practical implementations, PPL is typically computed using Eq. 2.11 instead of Eq. 2.8 (i.e., taking the exponent of the cross-entropy rate).

We need to consider two deficiencies of PPL when using it in experiments. First, PPL is viewed as a quick evaluation measure that correlates with extrinsic improvements (Jurafsky, 2000). However, improving PPL does not necessarily guarantee an extrinsic improvement in downstream tasks such as speech recognition (Clarkson and Robinson, 2001; Iyer *et al.*, 1997; Martin *et al.*, 1997). Second, PPL depends not only on the LM but also on vocabulary size and text type used during training. Thus, valid comparisons can only be made when the same dataset is used (De Mulder *et al.*, 2015).

3 Preliminaries: Tensor Tools

This chapter is organized as follows: We present the notation, diagrams, and tensor operations in Sec. 3.1 to cater to readers with different background knowledge. Following this, we introduce tensor networks, mainly focusing on Tensor-Train decompositions in Sec. 3.2. Complete technical introductions can be found in standard textbooks (e.g., Itskov, 2009; Bi *et al.*, 2022).

3.1 Tensor Computations

We present the notation and diagrams in Sec. 3.1.1 and basic tensor operations used throughout the thesis in Sec. 3.1.2.

3.1.1 Notation and Diagrams

Table 3.1 The notations used throughout the thesis.

	`Numbers, Arrays, and Sets`
A	Scalar (integer or real)
$\mathbf{A}$	Vector, Matrix or Higher-order tensors (order three or higher)
$\mathbb{N}^+$	The shorthand for the natural without zeros (i.e., $\{1,2,\cdots\}$)
$\mathbb{R}_{[0,1]}$	The shorthand for the set: $\{x \in \mathbb{R} \vert 0 \leq x \leq 1\}$
$[K]$	The shorthand for the set, $\{1,\cdots,K\}$, where $K \in \mathbb{N}_+$
	`Indexing`
A_i	Element i of vector $\mathbf{A}$, with indexing starting at 1
$\mathbf{A}_{ij}$	Element i, j of matrix $\mathbf{A}$
$\mathbf{A}_{ijk}$	Element i, j, k of a third-order tensor $\mathbf{A}$
$\mathbf{A}_{i,:,:}$	2-D horizontal slice of a third-order tensor

We use the following definition of a tensor, in keeping with the aim of this thesis and existing research (Kolda and Bader, 2009; Orús, 2014; Bridgeman and Chubb, 2017):

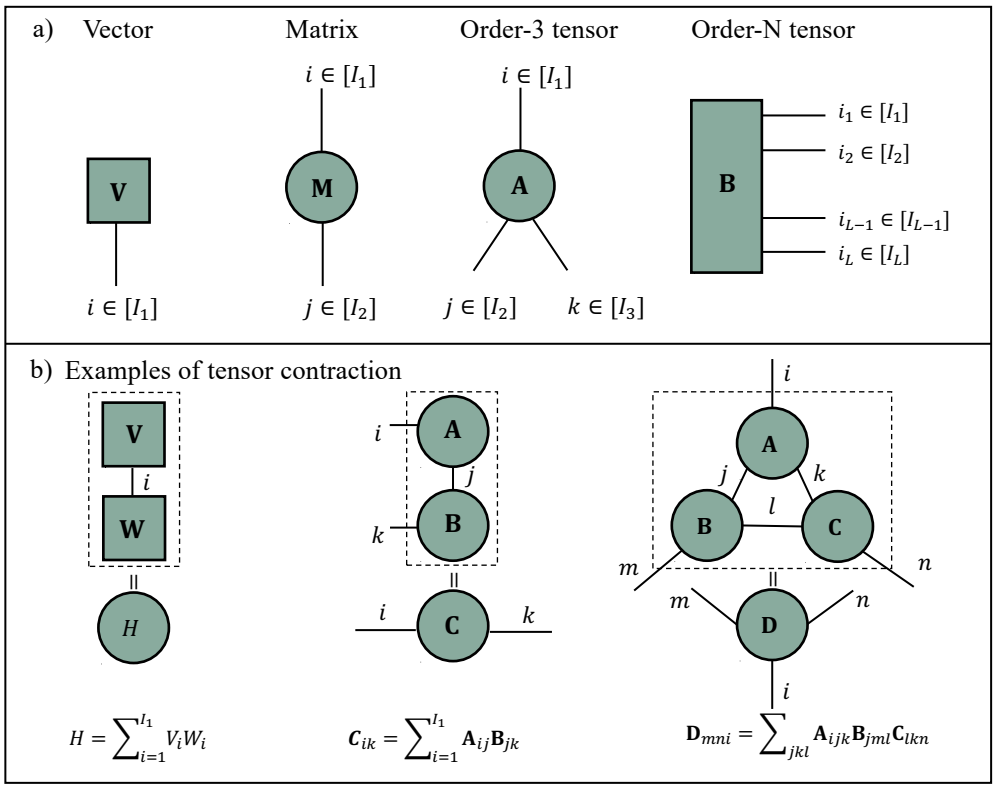

Figure 3.1 A quick introduction to *tensor diagram notation*. There are two rules of tensor diagrams: (1) solid shapes notate tensors with some 'legs' corresponding to their indices; and (2) connecting two index lines denotes a *contraction* (see Def. 3.1.2). We may augment our equations with these diagrams to make them easier to understand in this thesis.

Definition 3.1.1 (Orús, 2014)**.** *A tensor is a multidimensional array of real or complex numbers.*

For example, vectors and matrices are tensors. Given $L \in \mathbb{N}_+$ and $I_k \in \mathbb{N}_+$, $\quad \forall k \in [L]$, the elements of $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_L}$ is denoted as $\mathbf{A}_{i_1,\cdots,i_L} \in \mathbb{R}$, where $i_k \in [I_k]$. This thesis also denotes tensors using diagram notation (Penrose, 1971) as depicted in Fig. 3.1a. Note that though we define tensors as multidimensional arrays, it is also common to define them as the *tensor product* of vector spaces (Kolda and Bader, 2009).

The *order* of a tensor is the number of indexing entries in the tensor, and a particular indexing entry is called a *mode*. The term *dimension* represents the number of values an index can take in a particular mode. For example, a vector is a first-order tensor; a matrix is a second-order tensor; and $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_L}$ has order L and dimension I_i in mode i where $i \in [L]$. The two-dimensional sections of a tensor are usually called the *slices*, defined by fixing all but two indices (Kolda and Bader, 2009).

The slices of a third-order tensor $\mathbf{A}$ are denoted by $\mathbf{A}_{i,:,:}$ (horizontal slices), $\mathbf{A}_{:,j,:}$ (lateral slices), $\mathbf{A}_{:,:,k}$ (frontal slices).

We often use diagrams to describe *tensor contraction* as depicted in Fig. 3.1b, which is defined as:

Definition 3.1.2 (Orús, 2014)**.** *A tensor contraction is the sum of all the possible values of the repeated indices of a set of tensors.*

For example, the *matrix product*, $\mathbf{A}_{ij} = \sum_{k=1}^{D} \mathbf{B}_{ik}\mathbf{C}_{kj}$, is the contraction of index k, which amounts to the sum over its D possible values. One can also have more complicated contractions, such as this one (Orús, 2014):

$$\mathbf{F}_{\gamma\omega\rho\sigma} = \sum_{\alpha,\beta,\delta,\nu,\mu=1}^{D} \mathbf{A}_{\alpha\beta\delta\sigma}\mathbf{B}_{\beta\gamma\mu}\mathbf{C}_{\delta\nu\mu\omega}\mathbf{E}_{\nu\rho\alpha},$$

where $\alpha, \beta, \delta, \nu$ and $\mu \in [D]$ are contracted indices.

3.1.2 Tensor Operations

Now we introduce several tensor operations that will be used in this thesis. The first operation is *unfolding*, which will reorder the elements of an order-L tensor into a matrix (Kolda and Bader, 2009). In this thesis, we consider one type of unfolding of a tensor $\mathbf{A}$, which we refer to as the *reshape*.

Definition 3.1.3 (Oseledets, 2011)**.** *The reshape of a tensor* $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_L}$*, denoted here as* $\mathbf{A}_{[k]}$*, is a* $(I_1 I_2 \cdots I_k)$ *by* $(I_{k+1} I_{k+2} \cdots I_L)$ *matrix, whose elements are taken column-wise from* $\mathbf{A}$*, that is:*

$$(\mathbf{A}_{[k]})_{\overline{i_1 \cdots i_k}, \overline{i_{k+1} \cdots i_L}} = \mathbf{A}_{i_1, \cdots, i_k, i_{k+1}, \cdots, i_L}. \tag{3.1}$$

where the multi-index $\overline{i_1 \ldots i_L}$ *is defined as:*

$$\overline{i_1 \cdots i_L} = i_1 + (i_2 - 1)I_1 + \cdots + (i_L - 1)I_1 \cdots I_{L-1}.$$

Given an Lth-order tensor $\mathbf{A}$, it is possible to have L unfoldings and $L-1$ reshapes.

Another fundamental manipulation of tensors is the *permute* function. Permute allows the index ordering of a tensor to be changed (but does not change the number of indices). For example, given $\mathbf{A} \in \mathbb{R}^{2\times3\times4\times5}$, we can reorder its mode-1 and mode-2 to $\tilde{\mathbf{A}} \in \mathbb{R}^{3\times2\times4\times5}$, and $\mathbf{A}_{i_1,i_2,i_3,i_4} = \tilde{\mathbf{A}}_{i_2,i_1,i_3,i_4}$.

Definition 3.1.4 (Cohen *et al.*, 2016)**.** *The tensor product of two tensors* $\mathbf{A}$ *(order* L_1*) and* $\mathbf{B}$ *(order* L_2*) is denoted by* $\mathbf{A} \otimes \mathbf{B}$ *(order* $L_1 + L_2$*) and is defined as:*

$$(\mathbf{A} \otimes \mathbf{B})_{i_1,i_2,\cdots,i_{L_1+L_2}} = \mathbf{A}_{i_1,i_2,\cdots,i_{L_1}} \cdot \mathbf{B}_{i_{L_1+1},i_{L_1+2},\cdots,i_{L_1+L_2}}.$$

Notice that when $L_1 = L_2 = 1$, the tensor product reduces to an *outer product* between vectors.

Definition 3.1.5 (Kolda and Bader, 2009)**.** *The inner product of two same-sized tensors* $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{I_1\times I_2\times\cdots\times d_N}$ *is the sum of the products of their entries:*

$$\langle \mathbf{A}, \mathbf{B} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_L=1}^{I_L} \mathbf{A}_{i_1,i_2,\ldots,i_L} \mathbf{B}_{i_1,i_2,\ldots,i_L}.$$

Definition 3.1.6 (Kossaifi *et al.*, 2020)**.** *The generalized inner product of two tensors* $\mathbf{A} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_L\times I_x}$ *and* $\mathbf{B} \in \mathbb{R}^{I_1\times I_2\times\cdots I_L\times I_y}$ *along the same-sized L modes is defined as:*

$$\langle \mathbf{A}, \mathbf{B} \rangle_L = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_L=1}^{I_L} \mathbf{A}_{i_1,i_2,\ldots,i_L} \mathbf{B}_{i_1,i_2,\ldots,i_L}.$$

with $\langle \mathbf{A}, \mathbf{B} \rangle_L \in \mathbb{R}^{I_x\times I_y}$.

Definition 3.1.7 (Kolda and Bader, 2009). *The Frobenius norm of a tensor* $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_L}$ *is the square root of the sum of the squares of all its elements:*

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_L=1}^{I_L} \mathbf{A}_{i_1,i_2,\ldots,i_L}^2}.$$

Thus, $\|\mathbf{A}\|_F = \sqrt{\langle \mathbf{A}, \mathbf{A} \rangle}$. Note that the squared Frobenius norm is defined as: $\|\mathbf{A}\|_F^2 = \langle \mathbf{A}, \mathbf{A} \rangle$ (see Eq. 2.61 in Clerckx and Oestges, 2013).

Finally, we introduce the *tensor rank*, one of the most important concepts in tensor computations:

Definition 3.1.8 (Kolda and Bader, 2009). *An Lth-order tensor* $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_L}$ *is rank one if it can be written as the outer product of L vectors* $\mathbf{x}^{(1)} \in \mathbb{R}^{I_1}, \mathbf{x}^{(2)} \in \mathbb{R}^{I_2}, \cdots, \mathbf{x}^{(L)} \in \mathbb{R}^{I_L}$*:*

$$\mathbf{A} = \mathbf{x}^{(1)} \circ \mathbf{x}^{(2)} \circ \cdots \circ \mathbf{x}^{(L)},$$

where $\circ$ *means the outer product.*

This means that each element of the tensor is the product of the corresponding vector elements:

$$\mathbf{A}_{i_1,i_2,\cdots,i_L} = x_{i_1}^{(1)} x_{i_2}^{(2)} \cdots x_{i_L}^{(L)}, \quad \forall 1 \leq i_n \leq I_n, \quad n \in [L].$$

Definition 3.1.9 (De Lathauwer *et al.*, 2000; Definition 4). *The rank of* $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_L}$ *is the minimal number of rank-one tensors that yield* $\mathbf{A}$ *as a linear combination and we denote it by* $\text{rank}(\mathbf{A})$.

3.2 Tensor Networks

We briefly introduce tensor networks' definitions, properties, and applications in Sec. 3.2.1. Then, we illustrate the definition and main theorems of Tensor-Train (TT) decompositions in Sec. 3.2.2.

3.2.1 Basic Concepts and Terminology

Definition 3.2.1 (Orús, 2014)**.** *A tensor network is a non-empty set of tensors where some, or all, of its indices, are contracted according to some pattern.*

A tensor network (TN) should satisfy two basic concepts according to Def. 3.2.1: *contraction* and *pattern*. While Def. 3.1.2 outlines the definition of contraction, the term *pattern* has not been explicitly defined in a precise manner. For example, the pattern in a TN may refer to either its geometric or mathematical pattern. Other work, such as that proposed by Bridgeman and Chubb (2017), defined a TN as a diagram that tells us how to combine several tensors into a single composite tensor. This statement, unlike Def. 3.2.1, did not mention a particular pattern of contraction in TNs. Thus, although these statements provide an intuition about the term *tensor networks*, they do not constitute a strictly formalized definition, and further mathematical formalization is necessary.

TNs have proven an important tool in attempting to overcome *the curse of dimensionality* and have contributed to the theoretical understanding of quantum wave functions, particularly regarding quantum entanglement (Orús, 2019). This issue, discussed in Sec. 2.2.2, poses difficulties in developing LMs. It is also one of the biggest hindrances in the theoretical and numerical study of quantum many-body systems, owing to the exponential growth of the number of dimensions of the Hilbert space (Bridgeman and Chubb, 2017).

To gain a deeper understanding of TNs, let us consider a quantum many-body system formed from a composition of L individual systems of dimension D, and its description is governed by the following wave function:

$$|\Psi\rangle = \sum_{i_1,i_2,\cdots,i_L}^{D} \mathbf{C}_{i_1,i_2,\cdots,i_L} |i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_L\rangle, \tag{3.2}$$

where $\otimes$ denotes *tensor product* introduced in Def. 3.1.4, each $|i_t\rangle$ denotes a vector in a d-dimensional Hilbert space $\mathcal{H}$ with an orthogonal basis, and the dimension of the full Hilbert space $\mathcal{H}^{\otimes L}$ is D^L. Thus, the order-L tensor $\mathbf{C}_{i_1,i_2,\cdots,i_L}$ has D^L entries, giving it a space complexity of $\mathcal{O}(2^L)$. Consequently, the

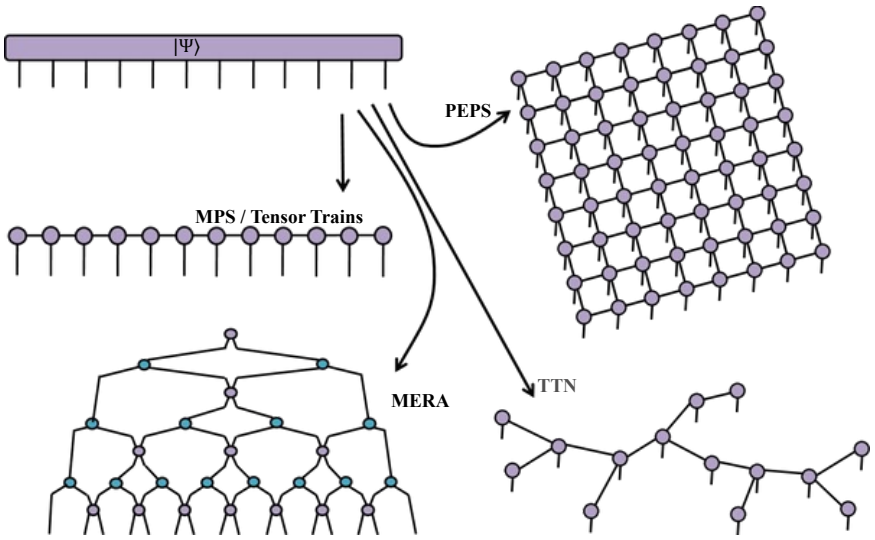

Figure 3.2 Examples of tensor network decompositions (Evenbly, n.d.): Matrix Product States (MPS) (Perez-Garcia *et al.*, 2006), Tree Tensor Networks (TTN) (Shi *et al.*, 2006), the Multi-scale Entanglement Renormalization Ansatz (MERA) (Vidal, 2007), and Projected Entangled Pair States (PEPS) (Verstraete and Cirac, 2004). The introduction of the diagrammatic notation is presented in Fig. 3.1.

complete description of the wave function $|\Psi\rangle$ is not achievable in polynomial time, rendering it an intractable task (Goldreich, 2008; Sipser, 1996).

A generic way to deal with the wave function is to decompose the higher-order tensor $\mathbf{C}_{i_1,i_2,\cdots,i_L}$ into a TN consisting of lower-order tensors (Verstraete *et al.*, 2008; Orús, 2014). This process, referred to as *tensor decompositions*, yields various TNs depending on the chosen decomposition, as exemplified in Fig. 3.2. The best choice of tensor decompositions depends on the geometry of the problem as well as its physical properties (Bridgeman and Chubb, 2017). Note that tensor decompositions rely on multiplicative interactions to represent complex correlations within data; this linearity gives us a convenient mathematical structure to represent and manipulate correlated data (Miller *et al.*, 2021).

3.2.2 Tensor-Train Decomposition

Let $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_L}$, its TT-decomposition is given by (Oseledets, 2011):

$$\mathbf{A}_{i_1,\cdots,i_L} = \mathbf{G}^{(1)}_{i_1,:}\mathbf{G}^{(2)}_{:,i_2,:}\cdots\mathbf{G}^{(L-1)}_{:,i_{L-1},:}\mathbf{G}^{(L)}_{:,i_L}, \tag{3.3}$$

where $\mathbf{G}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$, for $k \in [L]$ are called the *TT-cores*; R_k for $k \in [L]$ are called the *TT-ranks* and $R_0 = R_L = 1$; and $R = \max_{1 \leq k \leq L} R_k$ is called the maximal *TT-rank*.

The TT-decomposition of a tensor can be written as the *TT-format*:

Definition 3.2.2 (Oseledets, 2011)**.** *A tensor* $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_L}$ *is said to be in TT-format if its elements are given by:*

$$\mathbf{A}_{i_1,\cdots,i_L} = \sum_{\alpha_1=1}^{R_1} \sum_{\alpha_2=2}^{R_2} \cdots \sum_{\alpha_{L-1}=1}^{R_{L-1}} \mathbf{G}^{(1)}_{i_1 \alpha_1} \mathbf{G}^{(2)}_{\alpha_1 i_2 \alpha_2} \cdots \mathbf{G}^{(L)}_{\alpha_{L-1} i_L}, \tag{3.4}$$

We next recall one of the main theorems for the TT-decomposition that gives a recursive way to compute it.

Theorem 3.2.1 (Oseledets, 2011; Thm. 2.1)**.** *If for each unfolding matrix* $\mathbf{A}_{[k]}$ *of form* (3.1) *of a tensor* $\mathbf{A}_{i_1,\cdots,i_L}$ *with*

$$\operatorname{rank}(\mathbf{A}_{[k]}) = R_k, \tag{3.5}$$

then there exists a decomposition (3.3) *with TT-ranks not higher than* R_k*.*

Proof. Consider the first unfolding of $\mathbf{A}$ and its dyadic (skeleton) decomposition $\mathbf{A}_{[k]} = \mathbf{U}\mathbf{V}^\top$, which can be written in the index form

$$(\mathbf{A}_{[1]})_{i_1, \overline{i_2 \cdots i_L}} = \sum_{\alpha_1=1}^{R_1} \mathbf{U}_{i_1,\alpha_1} (\mathbf{V}^\top)_{\alpha_1, \overline{i_2 \cdots i_L}},$$

because $\operatorname{rank}(\mathbf{A}_{[1]}) = R_1$. Then, the first TT-core is given by the matrix $\mathbf{G}^{(1)}$ such that $\mathbf{G}^{(1)}_{i_1,\alpha_1} = \mathbf{U}_{i_1,\alpha_1}$. The matrix $\mathbf{V}$ can be expressed as:

$$\mathbf{V} = \mathbf{A}^\top_{[1]} \mathbf{U} (\mathbf{U}^\top \mathbf{U})^{-1} = \mathbf{A}^\top_{[1]} \mathbf{W},$$

or in the index form:

$$\mathbf{V}_{\overline{i_2\cdots i_L},\alpha_1} = \sum_{i_1=1}^{I_1} (\mathbf{A}_{[1]}^{\top})_{\overline{i_2\cdots i_L},i_1} \mathbf{W}_{i_1,\alpha_1}, \tag{3.6}$$

where $\mathbf{W} = \mathbf{U}(\mathbf{U}^{\top}\mathbf{U})^{-1}$.

Now we treat the matrix $\mathbf{V}$ as a $(L-1)$th-order tensor $\hat{\mathbf{V}}$ with $(\alpha_1 i_2)$ as one long index, which denotes its elements as $\hat{\mathbf{V}}_{\overline{\alpha_1 i_2},i_3,\cdots,i_L}$. To show that $\text{rank}(\hat{\mathbf{V}}_{[k]}) \leq R_k$, let us consider the unfoldings $(\hat{\mathbf{V}}_{[k]})_{\overline{\alpha_1 i_2\cdots i_k},\overline{i_{k+1}\cdots i_L}}$ for $k = 2,\cdots,L$. Since (3.5) holds, $\mathbf{A}$ can be reshaped and decomposed as:

$$(\mathbf{A}_{[k]})_{\overline{i_1\cdots i_k},\overline{i_{k+1}\cdots i_L}} = \sum_{\beta=1}^{R_k} \mathbf{F}_{\overline{i_1\cdots i_k},\beta} (\mathbf{J}^{\top})_{\beta,\overline{i_{k+1}\cdots i_L}}.$$

Using Eq. 3.6, we obtain:

$$\begin{aligned}(\hat{\mathbf{V}}_{[k]})_{\overline{\alpha_1 i_2\cdots i_k},\overline{i_{k+1}\cdots i_L}} &= \sum_{i_1=1}^{I_1}\sum_{\beta=1}^{R_k} \mathbf{F}_{\overline{i_1\cdots i_k},\beta} (\mathbf{J}^{\top})_{\beta,\overline{i_{k+1}\cdots i_L}} \mathbf{W}_{i_1,\alpha_1} \\ &= \sum_{\beta=1}^{R_k} \mathbf{H}_{\overline{\alpha_1 i_2\cdots i_k},\beta} (\mathbf{J}^{\top})_{\beta,\overline{i_{k+1}\cdots i_L}}\end{aligned} \tag{3.7}$$

where we treat $\mathbf{F}_{\overline{i_1\cdots i_k},\beta}$ as $\tilde{\mathbf{F}}_{\overline{i_2\cdots i_k},\beta,i_1}$ and thus $\mathbf{H}_{\overline{i_1\cdots i_k},\beta,\alpha_1} = \tilde{\mathbf{F}}_{\overline{i_2\cdots i_k},\beta,i_1}\mathbf{W}_{i_1,\alpha_1}$. From (3.7) we have $\hat{\mathbf{V}}_{[k]}$ whose row and column indices are now separated and $\text{rank}(\hat{\mathbf{V}}_{[k]}) \leq R_k, \forall k \in [L]$.

The process can be continued by induction. Now if we consider the unfolding $\hat{\mathbf{V}}_{[1]}$, we have:

$$(\hat{\mathbf{V}}_{[1]})_{\overline{\alpha_1 i_2},\overline{i_3\cdots i_L}} = \sum_{\alpha_2=1}^{R_2} \mathbf{K}_{\overline{\alpha_1 i_2},\alpha_2} (\mathbf{E}^{\top})_{\alpha_2,\overline{i_2\cdots i_L}}.$$

Then the second TT-core is given by the third-order tensor $\mathbf{G}^{(2)}$ such that $(\mathbf{G}^{(2)})_{[2]} = \mathbf{K}$. We can iterate this process to find the other core tensors $\mathbf{G}^{(k)}$ for $k = 3,\cdots,L$. □

The proof of Thm. 3.2.1 gives a recursive way to compute the TT-decomposition of an Lth-order tensor $\mathbf{A}$. Following Thm. 3.2.2, Oseledets (2011) considered an algorithm that computes the TT-

decomposition using the truncated *Singular Value Decomposition* (SVD) instead of the exact one; the introduced error can be estimated as shown in the following theorem.

Theorem 3.2.2 (Oseledets, 2011; Thm. 2.2)**.** *Suppose that the unfoldings* $\mathbf{A}_{[k]}$ *of the tensor* $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_L}$ *can be approximated by matrices of low rank* $\hat{\mathbf{A}}_k$*:*

$$\mathbf{A}_{[k]} = \hat{\mathbf{A}}_k + \mathbf{E}_k, \quad \operatorname{rank}(\hat{\mathbf{A}}_k) = R_k, \quad \|\mathbf{E}_k\|_F = \varepsilon_k, \quad k \in [L-1]. \tag{3.8}$$

Then truncated TT-SVD computes a tensor $\mathbf{B}$ *in the TT-format with TT-ranks* R_k *and*

$$\|\mathbf{A} - \mathbf{B}\|_F \leq \sqrt{\sum_{k=1}^{L-1} \varepsilon_k^2}. \tag{3.9}$$

This truncated TT-SVD is reported in Algorithm 1. Note that the numel$(\cdot)$ and reshape$(\cdot)$ functions in this algorithm refer to the MATLAB's functions (Oseledets, 2011).

Algorithm 1 Truncated TT-SVD [Oseledets (2011), Algorithm 1]

Require: Tensor $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_L}$, prescribed accuracy ε

Ensure: Cores $\mathbf{G}^{(1)}, \cdots, \mathbf{G}^{(L)}$ of the TT-approximation $\mathbf{B}$ to $\mathbf{A}$ in the TT-format with TT-ranks R_k equal to the δ-ranks of the unfoldings $\mathbf{A}_{[k]}$ of $\mathbf{A}$, where $\delta = \frac{\varepsilon}{\sqrt{L-1}}\|\mathbf{A}\|_F$. The computed approximation satisfies

$$\|\mathbf{A} - \mathbf{B}\|_F \leq \varepsilon \|\mathbf{A}\|_F.$$

1: Compute the truncation parameter $\delta = \frac{\varepsilon}{\sqrt{L-1}}\|\mathbf{A}\|_F$.
2: Create a copy of the original tensor $M_1 = \mathbf{A}$.
3: Set $R_0 = R_L = 1$.
4: **for** $k = 1, \cdots, L-1$ **do**
5: $\quad \mathbf{M}_k = \text{reshape}(\mathbf{M}_k, [R_{k-1} \cdot I_k, \frac{\text{numel}(\mathbf{M}_k)}{R_{k-1} \cdot I_k}])$
6: $\quad$ Compute the δ-truncated SVD, $\mathbf{M}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top + \mathbf{E}_k$ where $\|\mathbf{E}_k\|_F = \delta$ and $R_k = \text{rank}_\delta(\mathbf{U}_k)$;
7: $\quad$ New core: $\mathbf{G}^{(k)} = \text{reshape}(\mathbf{U}_k, [R_{k-1}, I_k, R_k])$.
8: $\quad \mathbf{M}_{k+1} = \mathbf{\Sigma}_k \mathbf{V}_k^\top$.
9: **end for**
10: Set the last TT-core $\mathbf{G}^{(L)} = \mathbf{M}_L$.
11: Return TT-cores $\mathbf{G}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$, for $k \in [L-1]$.

4 Related Work

We organize this chapter as follows: **(1)** In Sec. 4.1, we present a historical review of the tensor network language models (TNLMs), clarify their research objectives, and explain their paradigm. This section represents a pioneering effort to untangle and assess this research domain. **(2)** In Sec. 4.2, we seek to shed light on future research directions of TNLMs and offer our outlook to experienced researchers. To achieve this goal, we focus on recent advances in recurrent neural networks (RNNs). This direction is chosen because RNNs have similar research objectives and mathematical relationships with TNLMs, whose innovations can illuminate the development of TNLMs. **(3)** Building upon the two sections, we share our thoughts on the limitations and potential improvements of TNLMs in Sec. 4.3.

4.1 Advances in Tensor Network Language Modeling

The section is structured as follows: a historical review in Sec. 4.1.1, prior motivations and research objectives in Sec. 4.1.2, the current paradigm in Sec. 4.1.3.

4.1.1 Historical Review

In the past decade, applications of TNs underwent a surge of progress in various fields, including quantum gravity and holography (Han and Hung, 2017), error-correcting codes (Jahn and Eisert, 2021), classical data compression (Cichocki *et al.*, 2017) and big data analysis (Cichocki, 2014). In deep learning, prior research used TNs to analyze the theoretical properties of neural network models. A better understanding of feed-forward, convolutional and recurrent architectures has been gained, including parameters compression (Novikov *et al.*, 2015), expressive power (Cohen *et al.*, 2016;

Table 4.1 Timeline of tensor networks used in language modeling or density estimation. The term *theoretical proposals* refers to papers that put forward a theoretical framework without experimental evaluation. The term *small-scale vocabulary* denotes papers with experiments on sequence modeling with small-scale vocabulary size ($|V| \leq 38$). Tensor networks used in prior research include the Multi-scale Entanglement Renormalization Ansatz (MERA; Vidal, 2007), Matrix Product States (MPS; Perez-Garcia *et al.*, 2006), and Tree Tensor Networks (TTNs; Shi *et al.*, 2006).

Developing Stage	Types of Tensor Networks
Theoretical proposals	MERA (Pestun and Vlassopoulos, 2017) MPS (Pestun *et al.*, 2017)
Small-scale vocabulary	MPS (Han *et al.*, 2018) TTNs (Cheng *et al.*, 2019) MPS (Stokes and Terilla, 2019) MPS (Miller *et al.*, 2021) MPS (Novikov *et al.*, 2021) TTNs (Tang *et al.*, 2022) MPS (Hur *et al.*, 2022)

Khrulkov *et al.*, 2018), and depth efficiency for long-term memory (Levine *et al.*, 2018). In natural language processing, prior research used TNs to devise language models, which we refer to as the *Tensor Network Language Models* (TNLMs). The term *TNLMs* denotes the use of TNs to learn the joint probability distribution of sequences, with the result that TNLMs can generate new sequences by sampling from the learned probability distribution. This section focuses on previous research on TNLMs as listed in Table 4.1 and does not delve into the use of TNs in other fields.

The timeline of TNLMs can be categorized into two stages, *theoretical proposals* and *small-scale vocabulary*, from an experimental perspective. **(1)** As theoretical proposals, Pestun and Vlassopoulos, 2017 based the model on the Multi-scale Entanglement Renormalization Ansatz (MERA; Vidal, 2007), and Pestun *et al.*, 2017 used the Matrix Product States (MPS) (Perez-Garcia *et al.*, 2006). Neither of them has an experimental evaluation of their models. Note that MPS is also known as the TT-format (see Def. 3.4) (Oseledets, 2011); for consistency, we use "MPS" in this chapter. **(2)** The second category includes models that have been used to model sequences with small-scale vocabulary size ($|V| \leq 38$). Most of them were based on MPS (Han *et al.*, 2018; Stokes and Terilla, 2019; Novikov *et al.*, 2021; Miller *et al.*, 2021), except that Cheng *et al.* (2019) and Tang *et al.* (2022) used Tree Tensor Networks (TTNs).

4.1.2 Prior Research Objectives

This section clarifies the discrepancy between the research objectives of papers in the two categories presented in Table 4.1. In essence, *theoretical proposals* aimed to capture *long-range correlations* in language by TNs, while the models with *small-scale vocabulary* used TNs for diverse purposes.

As theoretical proposals

Pestun and Vlassopoulos (2017) and Pestun *et al.* (2017) aimed to exploit the capabilities of TNs to capture the "higher-order statistical properties of natural language" that were beyond the ability of current LMs. Specifically, they argued that LMs should capture the long-range correlations in human language. To elaborate on this point, we will introduce the concept of correlation function, as well as the definition of long-range correlations.

Definition 4.1.1 (Tanaka-Ishii, 2021)**.** *Let $S \in \mathbb{N}_+$ represent the number of elements between the first element of two sequences. A correlation function is a function $C(S)$ that measures the statistical correlation between the two sequences.*

For example, the sequence could be a list of ordered characters or words. Def. 4.1.1 generally describes the category of correlation functions. It appears that a rigorous mathematical definition of the entire category is yet to be established. Within this category, notable instances of correlation functions include the *mutual information function* $I(S)$ and *autocorrelation function* $ACF(S)$ (Tanaka-Ishii, 2021). For example, let us focus on $I(S)$ to understand how to use $C(S)$ in natural language. Let $\mathbf{x}$ represent a text composed of two sequences, each of length $L \in \mathbb{N}_+$, such that the distance between the first word of each sequence is $S \in \mathbb{N}_+$. The two sequences may overlap. $X^{(i)}$ refers to the random variable located at position i in $\mathbf{x}$. The averaged mutual information function, $I : \mathbb{N}_+ \to \mathbb{R}_{\geq 0}$, of two sequences is defined as (Tanaka-Ishii, 2021):

$$I(S) = \sum_{i=1}^{L} \sum_{x^{(i)}, x^{(i+S)} \in V} \tilde{P}\left(X^{(i)} = x^{(i)}, X^{(i+S)} = x^{(i+S)}\right) \log \frac{\tilde{P}\left(X^{(i)} = x^{(i)}, X^{(i+S)} = x^{(i+S)}\right)}{\tilde{P}\left(X^{(i)} = x^{(i)}\right) \tilde{P}\left(X^{(i+S)} = x^{(i+S)}\right)}, \tag{4.1}$$

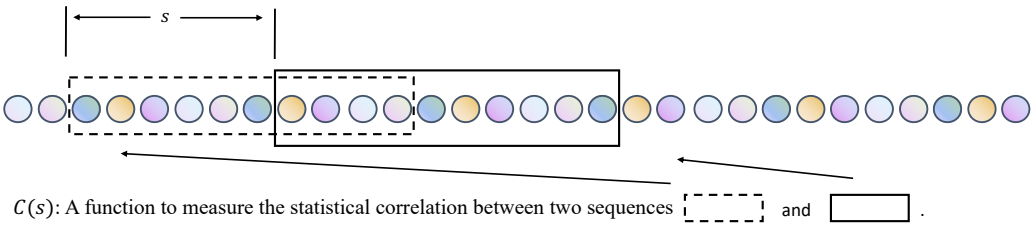

Figure 4.1 Schematic illustration of long-range correlation analysis (Tanaka-Ishii, 2021). Each solid circle represents a word or character in the text. $C(S)$ is a correlation function to measure the statistical correlation between two sequences, such that the distance between the first word of each sequence is S.

where V is the vocabulary set, and $x^{(i)}, x^{(i+S)} \in V$ denotes specific words at the corrsponding positions. The averaging can be accompanied by an oscillating function such as `sin` (Ebeling and Pöschel, 1994). In this type of experiment, the vocabulary V is typically collected in the entire dataset (Ebeling and Pöschel, 1994; Ebeling and Neiman, 1995; Montemurro and Pury, 2002; Altmann *et al.*, 2012); Hence, when computing Eq. 4.1, $\tilde{P}(X^{(i)} = x^{(i)}) \neq 0, \quad \forall x^{(i)} \in V$. If we use a large vocabulary set to assess a small dataset, we can use engineering techniques, such as smoothing, to avoid a word that has zero probability.

The definition of *long-range correlations* is based on how the value of a correlation function changes, as outlined below:

Definition 4.1.2 (Tanaka-Ishii, 2021)**.** *A text is long-range correlated if a correlation function, $C(S)$, follows a power function with respect to the distance S between two of its sequences as follows:*

$$C(S) \propto S^{-\gamma}, \quad S \in \mathbb{N}_+, \quad \gamma \in \mathbb{R}_{(0,1)}, \tag{4.2}$$

where γ is the power exponent indicating the degree of decay of $C(S)$ with respect to S.

For example, Fig. 4.1 illustrates a method of computing $C(S)$ in a text. This method places a dashed box at the beginning of the text and moves a solid box toward the end while calculating $C(S)$ at each step. If $C(S)$ can be characterized as a power function, the text is considered to have a long-range correlation according to Def. 4.1.2.

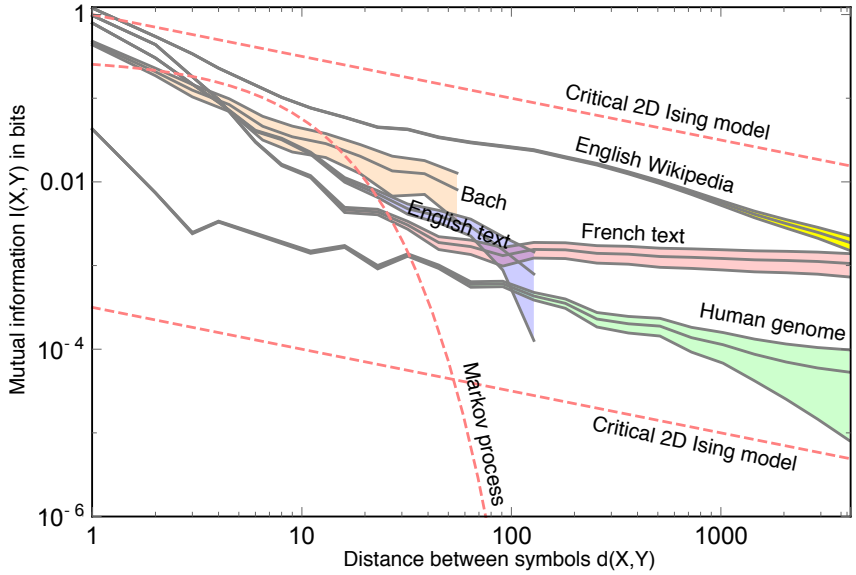

Figure 4.2 Decay of mutual information (Lin and Tegmark, 2017). The mutual information in bits per symbol is shown as a function of separation $d(X,Y) = |i-j|$, where the symbols $X$and Y are located at positions i and j in the sequence in question, and shaded bands correspond to $1-\delta$ error bars. These human genome data consist of 177,696,512 base pairs A, C, T, G from chromosome 5 from the National Center for Biotechnology Information, with unknown base pairs omitted. These Bach data consist of 5727 notes from Partita No. 2, with all notes mapped into a 12-symbol alphabet consisting of the 12 half-tones C, C#, D, D#, E, F, F#, G, G#, A, A#, B, with all timing, volume and octave information discarded. The three text corpora are 100 MB from Wikipedia (206 symbols), the first 114 MB of a French corpus (185 symbols), and 27 MB of English articles (143 symbols).

Prior research has reported long-range correlations in human language (Ebeling and Pöschel, 1994; Ebeling and Neiman, 1995; Montemurro and Pury, 2002; Altmann *et al.*, 2012). In addition, the power-law decay of $C(S)$ has also been reported in other domains, including biological systems (Mora and Bialek, 2011; Tagliazucchi *et al.*, 2012; Linkenkaer-Hansen *et al.*, 2001), music (Levitin *et al.*, 2012; Manaris *et al.*, 2005) and physics (Bak *et al.*, 1987; Bak *et al.*, 1988). For example, Fig. 4.2 illustrates the decay of mutual information in three text corpora, music, and human genome sequences.

In their TNLMs proposals, Pestun *et al.* (2017) and Pestun and Vlassopoulos (2017) emphasized the failure of capturing long-range correlations in human language by the existing LMs. To support their claim, they cited the work of Lin and Tegmark (2017), who tested the ability of LMs to capture long-range correlations in text. Lin and Tegmark (2017) showed that the mutual information of the generated text by bi-gram LMs decreased exponentially, as depicted in Fig. 4.3. Also, even the widely used neural network architecture, LSTMs (Hochreiter and Schmidhuber, 1997), encountered difficulties generating text with long-range correlations. Thus, the two proposals attempted to use TNs to propose novel architectures for language modeling. Distinct types of TNs exhibit diverse physical geometries,

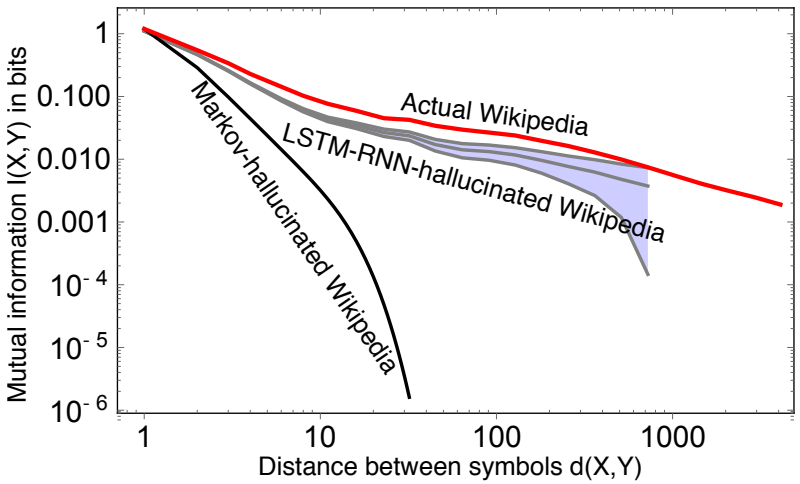

Figure 4.3 Comparison of mutual information between text generated by different models and real-word natural language (Lin and Tegmark, 2017). The red line is the mutual information of a 100 MB sample of English Wikipedia. In shaded blue is the mutual information of generated Wikipedia from a trained LSTM with 3 layers of size 256. The solid black line is the mutual information of a Markov process on single characters. Lin and Tegmark (2017) also measured the mutual information for generated text using bigram LMs, which underperforms the LSTMs. Note that the mutual information function used in (Lin and Tegmark, 2017) is at the character level and $N = 1$.

and certain types have advantageous theoretical properties for modeling human language (Evenbly and Vidal, 2011). For example, MERA is scale-invariant and has the asymptotic power-law decay of correlations (Vidal, 2008; Giovannetti *et al.*, 2008; Pfeifer *et al.*, 2009).

However, Pestun and Vlassopoulos (2017) and Pestun *et al.* (2017) did not experimentally implement their models, lacking empirical evidence to support their theoretical claims. This evidence is particularly critical when researchers endeavor to develop a new LM. Additionally, it is worth noting that Pestun *et al.* (2017) did not succeed in creating an LM with power-law decay correlations; they used MPS that has exponential decay correlations, as widely accepted among researchers (Fannes *et al.*, 1992; Östlund and Rommer, 1995; Rommer and Östlund, 1997). Although Stoudenmire and Schwab (2016) claimed that MPS could approximate power-law decays over relatively long distances w.r.t text, no solid empirical or theoretical analysis supports this claim.

With small-scale vocabulary

Building on the works of Pestun and Vlassopoulos (2017) and Pestun *et al.* (2017), an advancement in previous research is that they have evaluated their model on the datasets with small-scale vocabulary

size (Han *et al.*, 2018; Cheng *et al.*, 2019; Stokes and Terilla, 2019; Miller *et al.*, 2021; Novikov *et al.*, 2021; Tang *et al.*, 2022; Hur *et al.*, 2022).

Their motivations have shifted away from capturing long-range correlations. Here, we summarize three explanations suggested in prior literature as follows: **(1)** MPS has exponential decay correlations (Fannes *et al.*, 1992; Östlund and Rommer, 1995; Rommer and Östlund, 1997). From the perspective of Def. 4.1.2, this property fails to capture long-range correlations in text. Thus, it is hard for previous studies to demonstrate the long-range modeling abilities of their MPS-based models. **(2)** Def. 4.1.2 fails to measure the effectiveness of the model in long-context scenarios. There is a need for a benchmark or metric that can evaluate its effectiveness. The first benchmark, however, designed for this purpose emerged three years after the publication of the two theoretical proposals in 2017, introduced by Tay *et al.* (2020). **(3)** The definition of long-range correlations in Def. 4.1.2 reveals limitations when evaluating model capacity in long-range scenarios. We provide a detailed discussion of its limitations in Sec. 4.3.1.

The motivation or research objectives of TNLMs in the category of *small-scale vocabulary* varies. To gain a basic understanding of their objectives, we briefly introduce them from the perspective of deep learning, rather than physics. We recommend that readers refer to the original paper for a more detailed explanation.

- Han *et al.* (2018) aimed to build on the connection between unsupervised generative modeling and quantum physics. Han *et al.* (2018) used MPS as a *Born machine* where Born's rule in quantum physics is borrowed for representing the joint probability distribution of data with squared amplitude of a wave function (see Eq. 4.3). Compared with the Hopfield model (Hopfield, 1982) and inverse Ising model, Han *et al.* (2018) said MPS exhibited a much stronger learning ability; MPS also enjoyed a direct sampling method (Ferris and Vidal, 2012) much more efficient than the Boltzmann machines.

- Cheng *et al.* (2019) used TTNs as an extension to the Born machines. Compared with MPS, they stated that TTNs exhibited natural modeling on two-dimensional data, such as images; TTNs are more favorable than MPS in the growth of correlation length of pixels.

- Stokes and Terilla (2019) used MPS as generative models based on quantum circuits that could provide an *inductive bias* for sequence modeling tasks. They said this is an advantage of MPS compared with *restricted Boltzmann machines* (Montúfar *et al.*, 2011).

- Miller *et al.* (2021) stated that the linearity of MPS has yet to be leveraged to develop new *operational* abilities. Using "transfer operators," their model can process the sequential inputs in a parallel manner, with sequences of length L being evaluated in parallel time $\mathcal{O}(\log L)$. Note that the input sequence is assumed to be regular expressions since their developed techniques, transfer operators, link the structure of MPS to that of regular expressions.

- Novikov *et al.* (2021) believed there was a gap between simple, intuitive models for low-dimensional data and powerful, capable of solving most difficult tasks, yet very fragile and hard to theoretically analyze neural network models. To fill this gap, they aimed to build a new method of nonparametric density estimation using MPS. Their models have several features that other models do not have (at least not simultaneously), including tractable log-likelihood (unlike Generative Adversarial Networks (Goodfellow *et al.*, 2020)), exact sampling, ability to calculate the cumulative density function and exact calculation of the partition function.

4.1.3 Current Paradigm

While prior research has proposed TNLMs with diverse research objectives and motivations, these models share notable commonalities. These shared traits distinguish TNLMs from neural network LMs. To expound on this point, we illustrate the paradigm of TNLMs in four steps: **(1)** representing the joint probability distribution of sequences with squared amplitude of a wave function, **(2)** representing the wave function using a particular TN, **(3)** learning the parameters of the TN during training, and **(4)**

using the TN to sample new sequences during testing. Note that we have introduced the definition and main theorems of MPS in Sec. 3.2.2; thus, we use that terminology directly in this section.

Representing the Joint Probability Distribution

Given a sequence of $L \in \mathbb{N}_+$ words, denoted as $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$, where each $x^{(t)} \in V$ represents the t-th word in the sequence, we define V^L as the set of all possible length-L sequences (i.e., $\mathbf{x} \in V^L$). The first step to build TNLMs is encoding the joint probability of X into the wave function as follows:

$$\tilde{P}(\mathbf{x}) = \frac{|\Psi(\mathbf{x})|^2}{Z}, \tag{4.3}$$

where $Z = \sum_{\mathbf{x} \in V^L} |\Psi(\mathbf{x})|^2$ is the normalization factor, which is often referred to as the *partition function* in previous TNLMs to draw an analogy with the energy-based models (e.g., LeCun *et al.*, 2006). Note that V is a finite set, allowing us to calculate Z. Also, Eq. 4.3 ensures $\tilde{P}(\mathbf{x})$ is non-negative and that the sum of all $\tilde{P}(\mathbf{x})$ equals 1.

Several studies categorize their models as the "Born machines," where Born's rule is used to represent the joint probability distribution of data with squared amplitude of a wave function (Stokes and Terilla, 2019; Cheng *et al.*, 2019; Miller *et al.*, 2021). This interpretation is grounded in Eq. 4.3 satisfying Born's rule in quantum physics (i.e., the probability of finding a system in a given state is proportional to the square of the amplitude of the system's wave function at that state (Born, 1926)).

Though Eq. 4.3 naturally admits a quantum mechanical interpretation of $\Psi(\mathbf{x})$ as wave functions over L quantum spins, the potential benefits of this interpretation for devising LMs remain unclear. In other words, the property of $\Psi(\mathbf{x})$ has yet to be fully explored. For instance, despite $\Psi(\mathbf{x})$ being complex-valued in quantum mechanics, previous TNLMs are typically real-valued.

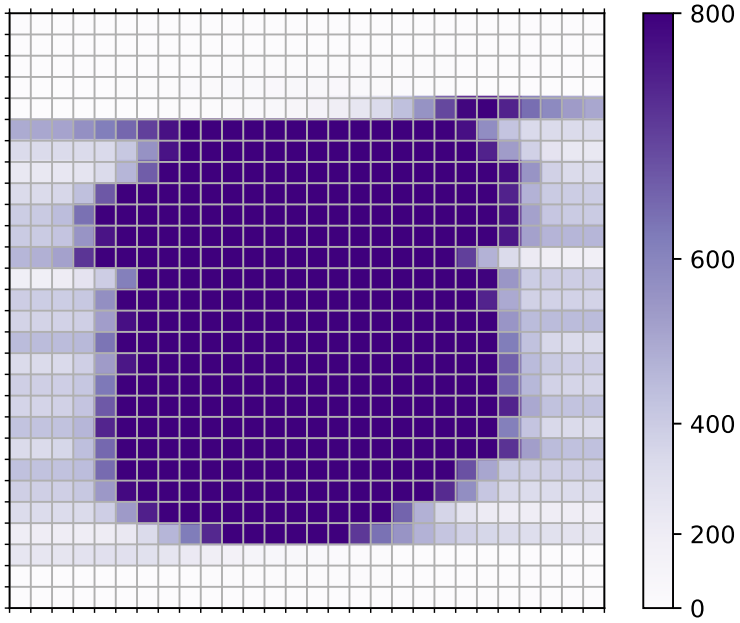

Figure 4.4 TT-ranks of MPS trained with $|T| = 1k$ MNIST samples, constrained to a maximum value ($R = 800$) (Han *et al.*, 2018). Each pixel in this figure corresponds to the TT-rank on the right side of the TT-core associated with the identical coordinate in the original image.

Model Architecture

The second step is selecting a particular TN to represent the wave function. Recall that the full description of wave functions has a space complexity of $\mathcal{O}(2^L)$ (see Eq. 3.2), and prior work used TNs to represent the wave function instead of storing all possible sequences. Most previous research developed their model architecture based on MPS. Here, we explain two key differences in how researchers used MPS in this step.

The first difference is the constraints on the TT-ranks. For example, Pestun *et al.* (2017) and Miller *et al.* (2021) assumed that all TT-ranks are equal; they called this form the *uniform MPS* (u-MPS). On the other hand, Han *et al.* (2018) used MPS with the prescribed maximal TT-rank, without any other restrictions on the rank of a particular TT-core. As depicted in Fig. 4.4, Han *et al.* (2018) displayed an example that the TT-ranks around the top and bottom edge of the images remain small, because those pixels are always inactivated in the images. Conversely, large TT-ranks are concentrated in the center of the images, where the variation of the pixels is complex. This finding is consistent with previous research on the applications of MPS in quantum many-body systems (Schollwöck, 2011; Orús, 2014), which states that as the TT-ranks increase, an MPS improves its ability to parameterize

more sophisticated functions. Therefore, we expect that the upper bound effectiveness of u-MPS used in Pestun *et al.* (2017) and Miller *et al.* (2021) to be inferior to that of MPS utilizing varying TT-ranks when handling long-range tasks.

Another major difference is the constraint on TT-cores. For instance, MPS has gauge degrees of freedom, which means that a TT-core can be invariant after inserting its identity $\mathbf{I} = \mathbf{M}\mathbf{M}^{-1}$. Exploiting the gauge freedom, Han *et al.* (2018) restricted TT-cores into canonical form; a TT-core is called *left-canonical* if it satisfies $\sum_{i_k \in [|V|]} (\mathbf{G}^{(k)}_{:,i_k,:})^T \mathbf{G}^{(k)}_{:,i_k,:} = \mathbf{I}$. The canonical form can reduce the computation cost of the normalization factor Z. More details about the canonical condition and calculation of Z can be found in Han *et al.* (2018).

Optimization

The third step is optimizing the parameters of a particular TN. Normally, this step requires a cost function and a gradient descent algorithm. For example, Han *et al.* (2018) used the negative log-likelihood (NLL) as the cost function, which is defined as:

$$\mathcal{L} = -\frac{1}{|T|} \sum_{\mathbf{x} \in V^L} \log \tilde{P}(\mathbf{x}), \tag{4.4}$$

where $|T| \in \mathbb{N}_+$ denotes the number of examples in the training set. When a model assigns zero to $\tilde{P}(\mathbf{x})$, prior TNLMs use engineering techniques such as smoothing to avoid this issue. It is worth emphasizing that minimizing $\mathcal{L}$ is equivalent to minimizing the Kullback-Leibler divergence (Kullback and Leibler, 1951) between the model probability distribution $\tilde{P}(\mathbf{x})$ and empirical distribution defined by the training set. Other cost functions are available; For example, Miller *et al.* (2021) used the *mean absolute error*.

After choosing a cost function, TNLMs require an optimization algorithm to update its parameters. For example, Han *et al.* (2018) employed an algorithm similar to the *Density Matrix Renormalization Group* (DMRG; White, 1992) for optimizing their model.

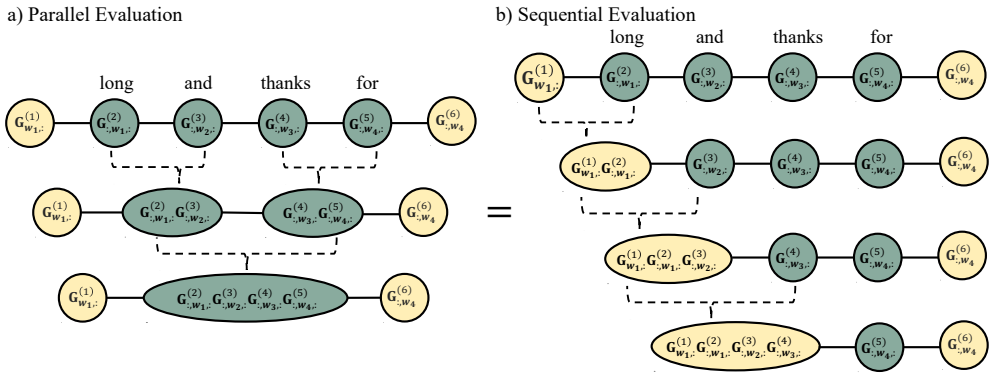

Figure 4.5 Illustration of parallel and sequential evaluation of the sequence `[long, and, thanks, for]` represented by MPS (Miller *et al.*, 2021), where the indices of words in the sequence is $w_1, w_2, \cdots, w_4$. Note that $\mathbf{G}^{(2)}_{:,w_1,:}, \cdots, \mathbf{G}^{(5)}_{:,w_4,:}$ are the matrix representations of each word. The $\mathbf{G}^{(1)}_{w_1,:}$ and $\mathbf{G}^{(6)}_{:,w_4}$ are denoted as boundary vectors, α and w, in Miller *et al.* (2021), without any interpretation. We use the notation $\mathbf{G}^{(1)}_{w_1,:}$ and $\mathbf{G}^{(6)}_{:,w_4}$ because we can naturally view the two vectors as <BOS> or <EOS> associated with the word w_1 or w_4.

Sampling

The fourth and final step is sampling new examples after training. Two distinct approaches have been proposed. The first approach is sequential and based on conditional probabilities. For example, given a sequence of $L \in \mathbb{N}_+$ words, $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$, the (k-1)-th word is sampled according to the conditional probability (Han *et al.*, 2018):

$$\tilde{P}(x^{(k-1)} | x^{(k)}, x^{(k-1)}, \cdots, x^{(L)}) = \frac{\tilde{P}(x^{(k-1)}, x^{(k)}, \cdots, x^{(L)})}{\tilde{P}(x^{(k)}, x^{(k+1)}, \cdots, x^{(L)})}. \tag{4.5}$$

This method is close to how RNNs theoretically generate sequences, with one key difference: RNNs calculate the conditional probability from left to right as introduced in Eq. 2.6, which is more natural for modeling sequences of human language.

In addition to sequential calculation, Miller *et al.* (2021) used the linearity of u-MPS to update its parameters in a parallel manner. Fig. 4.5 compares parallel and sequential evaluation of a sequence represented by MPS. As Miller *et al.* (2021) explained, after obtaining the matrix representations (i.e., $\mathbf{G}^{(2)}_{:,w_1,:}, \cdots, \mathbf{G}^{(5)}_{:,w_4,:}$), parallel evaluation involves repeated batch multiplications of nearest-neighbor pairs of matrices, with the boundary vectors (i.e., $\mathbf{G}^{(1)}_{w_1,:}$ and $\mathbf{G}^{(6)}_{:,w_4}$) only incorporated after the matrix product has been obtained. Sequential evaluation instead uses iterated matrix-vector multiplications starting with a boundary vector to contract this product. Parallel and sequential evaluation have respective

costs of $\mathcal{O}(LR^3)$ and $\mathcal{O}(LR^2)$, but the former can be carried out in $\mathcal{O}(\log L)$ parallel time (Miller *et al.*, 2021), where R is the maximal TT-rank and L is the number of TT-cores.

4.1.4 Comparing TNLMs with Neural Network LMs

Upon explaining the TNLMs paradigm, we observe that prior models with experimental evaluations mainly concentrate on the problem of *density estimation* (Silverman, 1986). Specifically, given $|T| \in \mathbb{N}_+$ independent samples, $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{|T|}$ where $\mathbf{y}_i = [y_i^{(1)}, y_i^{(2)}, \cdots, y_i^{(L)}]$ drawn from some ground truth density $P : \mathbb{R}^L \to \mathbb{R}$, their goal is to estimate P from the empirical distribution (Tang *et al.*, 2022). In other words, most TNLMs aim to minimize the dissimilarity between the joint probability distribution $\tilde{P}$ defined by themselves and the empirical distribution defined by the training set. For instance, Han *et al.* (2018) used NLL as the cost function to learn the empirical distribution of $1k$ images.

TNLMs are closer to density estimation models than to neural network LMs from the perspective of their paradigms. This assertion is supported by two observations. First, TNLMs have specific characteristics that neural network LMs lack, such as explicit tractable probability density estimation. Second, the experimental evaluation of TNLMs, including baselines, datasets, and metrics, typically differs from that used in language modeling. For instance, TNLMs have been benchmarked against density estimation models rather than LMs. Representative neural network models in density estimation include Variational AutoEncoders (VAE; Kingma and Welling, 2013), realNVP (Dinh *et al.*, 2016), PixelCNN (Van den Oord *et al.*, 2016), and GANs (Goodfellow *et al.*, 2020).

TNLMs are close to traditional LMs in language modeling, instead of neural network LMs. **(1)** Both TNLMs and most traditional LMs aim to learn the empirical distribution of sequences defined by the training set. For example, n-gram LMs estimate P using the maximum likelihood estimation on the empirical distribution of n-grams in the training set (see Sec. 2.2.1). The difference between TNLMs and traditional LMs is that TNLMs employ continuous representations as learnable parameters (e.g., TT-cores in tensor trains) to optimize $\tilde{P}$, whereas traditional LMs use discrete representations (e.g., frequency of n-grams). **(2)** In contrast, neural network LMs aim to learn the *distributed representation* of words (Jurafsky, 2000), instead of the empirical distribution defined by the training set. To

achieve this goal, Vanilla-RNNs are trained to predict the next word based on the previous context, as elaborated in Sec. 2.3.1. Other neural network LMs may predict masked words in a sequence or translate a sequence to the target text during training, as described in Sec. 2.3.2. Rarely do they explicitly compute or learn the joint probability distribution of data during training.

This discrepancy between TNLMs and neural network LMs results in distinct properties and applications. We will elaborate on how the paradigm of TNLMs may restrict their effectiveness in language modeling tasks in Sec. 4.3.2.

4.2 Advances in Recurrent Neural Networks

The effectiveness of TNLMs at the moment lags behind neural network LMs in language modeling tasks. To illuminate potential research avenues for TNLMs, we draw insights from recent RNN advancements. This choice is motivated by two main reasons: **(1)** TNLMs are proposed to harness the potential benefits of TNs to capture long-range correlations in human language (Pestun and Vlassopoulos, 2017; Pestun *et al.*, 2017). Similarly, RNNs aim to address *long-range dependencies* in human language (Bengio and Frasconi, 1994; Hochreiter and Schmidhuber, 1997). Recently, Orvieto *et al.* (2023) showed that careful design of Vanilla-RNNs can recover the impressive effectiveness of SOTA models on long-range reasoning tasks. By closely examining RNN progress, we can uncover crucial theoretical attributes essential for handling long-range scenarios. **(2)** MPS, as the simplest TNs, has a mathematical relationship with some RNNs (see Sec. 5.3.2). Thus, because of their similar mathematical structure, TNLMs could face similar issues that RNNs have encountered, such as vanishing and exploding gradients problems.

Our focus in this section is to present the work of Orvieto *et al.* (2023) in Sec. 4.2.2, with the aim of gaining insights into the innovations of TNLMs. Before delving into their work, we briefly present the recent progress in RNNs in Sec. 4.2.1, catering to readers who are new to RNNs. For those seeking a comprehensive review of RNNs, we recommend referring to Salehinejad *et al.* (2018).

Table 4.2 Timeline of some RNNs at a glance, organized according to three aspects of their innovations: gating mechanisms, connections, and initialization.

Models	Gating	Connections	Initializations
LSTMs (Hochreiter and Schmidhuber, 1997)	✔	✗	✗
GRUs (Chung *et al.*, 2014)	✔	✗	✗
MGUs (Zhou *et al.*, 2016)	✔	✗	✗
QRNNs (Bradbury *et al.*, 2017)	✔	✗	✗
RANs (Lee *et al.*, 2017)	✔	✗	✗
NARX-RNNs (Lin *et al.*, 1996)	✗	✔	✗
TKRNNs (Sutskever and Hinton, 2010)	✗	✔	✗
CW-RNNs (Koutnik *et al.*, 2014)	✗	✔	✗
DilatedRNNs (Chang *et al.*, 2017)	✗	✔	✗
IRNNs (Le *et al.*, 2015)	✗	✗	✔
np-RNNs (Talathi and Vartak, 2015)	✗	✗	✔
uRNNs (Arjovsky *et al.*, 2016)	✗	✗	✔
RIN (Hu *et al.*, 2018)	✗	✗	✔
IndRNNs (Li *et al.*, 2018)	✗	✗	✔
LRUs (Orvieto *et al.*, 2023)	✗	✗	✔
ON-LSTMs (Shen *et al.*, 2019)	✔	✔	✗
SRUs (Lei *et al.*, 2018)	✔	✗	✔
URLSTMs (Gu *et al.*, 2020)	✔	✗	✔

4.2.1 Recent Progress in RNNs

Training RNNs by the backpropagation through time algorithm (Rumelhart *et al.*, 1986) can be difficult in practice; for example, it suffers from vanishing and exploding gradients (Pascanu *et al.*, 2013), which limits the ability of RNNs to learn, especially on tasks with long input sequences (Bengio *et al.*, 1994; Hochreiter *et al.*, 2001). Various methods have emerged to tackle these challenges, spanning model architecture, input data, and model parameters. For the purposes of this thesis, we classify these methods into three rough categories according to their main contributions: **(1)** gating mechanisms, **(2)** connections, and **(3)** initialization. We lists key papers for each area in Table 4.2.

We briefly illustrate the development of these methods in the following paragraphs. For readers seeking insights into developing TNLMs, we suggest focusing on papers in the "initialization" category. These techniques center on model parameters instead of the architecture, making them more adaptable to tensor networks without modifying their theoretical architectures.

Gating Mechanism

Bengio and Frasconi (1994) observed that the vanilla version of RNNs is hard to capture long-term dependencies because the gradients tend to either vanish or explode. This issue makes gradient magnitudes vary and the effect of long-term dependencies hidden, hindering the gradient-based optimization method (Chung *et al.*, 2014). Focusing on the gating mechanism, many variants of RNNs have been proposed to optimize gradient flow, such as MGUs with one gate (Zhou *et al.*, 2016), GRUs with two gates (Chung *et al.*, 2014), and LSTMs with three gates (Hochreiter and Schmidhuber, 1997). At every time step, the values of the gates, which are the coefficients of the weighted combination of the previous states, control the length of temporal dependencies that can be addressed (Gu *et al.*, 2020).

The inner mechanism of gated RNNs varies. For example, the Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber, 1997) has input, output, and forget gates to remember or forget information from previous time steps. Another popular variant of gated RNNs is the Gated Recurrent Unit (GRU; Chung *et al.*, 2014), simplifying the LSTM architecture by merging input and forget gates into a single update gate. Instead of sequential computing, Quasi-RNNs (QRNNs; Bradbury *et al.*, 2017) use convolutions and a minimalist recurrent pooling function, achieving significant speed-up over LSTMs.

Connections

The second direction focuses on the time steps in RNNs to alleviate gradients vanishing or exploding. Such direction mainly intends to define or learn useful constraints that allow the network to skip certain parts of the input sequence, which are not taken into account during training (Alpay, 2021). Representative methods include skipping connections across multiple timestamps (Hihi and Bengio, 1995; Zhang *et al.*, 2016) or designing multi-timescale layers (Koutnik *et al.*, 2014; Chung *et al.*, 2016). For those interested in delving deeper into this field, we recommend the study by Alpay (2021).

To exemplify this research area, we briefly introduce the three relevant studies: NARX-RNNs (Lin *et al.*, 1996) introduce an additional set of recurrent connections with time lags of $2, 3 \cdots, k$ time steps. The

additional connections help to bridge long-time lags, but introduce two problems: NARX-RNNs are k times slower than Vanilla-RNNs (per iteration) and have k times more parameters than Vanilla-RNNs with the same number of hidden units. To overcome these problems, Sutskever and Hinton (2010) used leaky integrators for every unit and parameter sharing in Temporal-Kernel RNNs (TKRNNs; Sutskever and Hinton, 2010). Another attempt is Clockwork RNN (CW-RNN; Koutnik *et al.*, 2014), which partitions its hidden layer into separate modules, each processing inputs at its own temporal granularity, making computations only at its prescribed clock rate.

Initializations

Rather than on model architecture, a major research direction focuses on optimizing parameter initialization and maintenance to address gradient vanishing or exploding. Well-known methods include identity projection (Talathi and Vartak, 2015), unitary or orthogonal matrices (Arjovsky *et al.*, 2016; Helfrich *et al.*, 2018), and controlling the range of parameters (Li *et al.*, 2018; Kanai *et al.*, 2017).

For example, the unitary evolution RNNs (uRNNs; Arjovsky *et al.*, 2016) introduce unitary hidden-to-hidden matrices, where eigenvalues are restricted to live on the unit circle. While this restriction stabilizes the training process of RNNs considerably, it causes two problems (Orvieto *et al.*, 2023): smaller function approximation class and expensive training cost (since a projection on the Stiefel manifold is required at each gradient step). To resolve the second issue, many works designed reparameterization of the hidden-to-hidden matrix as the Givens rotations (Jing *et al.*, 2017), Householder reflections (Mhammedi *et al.*, 2017), or as the exponentials of skew-symmetric matrices (Hyland and Rätsch, 2017; Lezcano-Casado and Martınez-Rubio, 2019).

4.2.2 Design Space of Vanilla-RNNs

The recent advancement presented by Orvieto *et al.* (2023) holds significant importance for designing TNLMs. Our exploration of their work is motivated by two compelling factors. Primarily, their

recurrent model defies convention by embracing linearity, a departure from the prevailing belief that RNNs should exhibit nonlinearity (Siegelmann, 2012; Pascanu *et al.*, 2013; Erichson *et al.*, 2020). When tensor networks have been used to analyze the theoretical properties of neural networks (Cohen and Shashua, 2016; Khrulkov *et al.*, 2018; Levine *et al.*, 2018), the validity of such analyses has always been a point of contention. These doubts are due to the contradiction between the nonlinearity of neural network models and the linearity of tensor networks. On the experimental side, the linearity within tensor networks has presented challenges in training, a notion supported by our experimental findings (see Sec. 6.3.1). Consequently, we shall investigate the impact of linearity on model effectiveness and understand how Orvieto *et al.* (2023) overcame the hurdles stemming from this linearity.

Distinct from our earlier discussions on gating mechanisms or connections in Sec. 4.2.1, the innovation in Orvieto *et al.* (2023)centered on the parameter initializations in the complex number domain. In contrast, prior TNLMs have predominantly operated within the domain of real numbers (see Sec. 4.1.3, though tensor networks are naturally suited for complex numbers. Thus, the work of Orvieto *et al.* (2023) allows us to use their theorems to parameterize tensors while preserving the underlying architecture of tensor networks, as well as integrating complex numbers into the model (see Sec. 5.4).

In the following sections, we illustrate their insights into model development and theorem formulation. We use their notations for consistency.

Background, Motivation and Research Objective

Deep state-space models (SSMs) have achieved SOTA performance on the Long Range Arena benchmark (LRA; Tay *et al.*, 2020). These models are inspired by continuous-time linear SSMs, which is a well-established component of modern control systems (Orvieto *et al.*, 2023). Representative deep SMMs include S4 (Gu, Goel, and Ré, 2022) and its variants: DSS (Gu, Goel, Gupta, *et al.*, 2022), S4D (Gupta *et al.*, 2022), and S5 (Smith *et al.*, 2023). SSMs offer a crucial advantage over Transformers by addressing a limitation: the computational and memory costs of attention layers scale quadratically as $\mathcal{O}(L^2)$ with the sequence length L. SSMs and Vanilla-RNNs are superficially similar. Specifically,

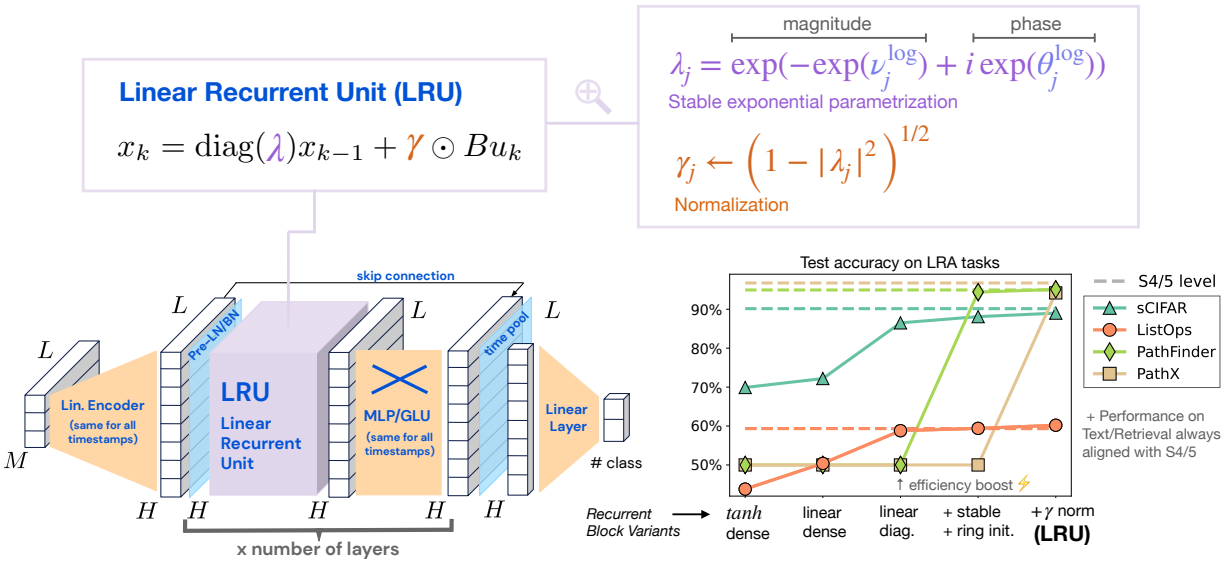

Figure 4.6 **(Left)** Deep Linear Recurrent Units (LRUs) architecture (Orvieto *et al.*, 2023). The model is a stack of 6 LRU blocks, with skip connections and batch/layer normalization (Ba *et al.*, 2016; Ioffe and Szegedy, 2015). **(Right)** Summary of effects for the main steps to build LRUs, starting from `RNNs-Tanh`. Shown is the average performance (3 seeds) of the recurrent module at each step on LRA tasks, compared with the average performance of deep SSMs.

let $[u_1, u_2, \cdots, u_L]$ be a sequence of H_{in}-dimensional inputs. An RNN/SSM layer with H-dimensional hidden state computes a sequence of N_{out}-dimensional outputs $[y_1, y_2, \cdots, y_L]$ through a recurrent computation using learnable parameters $A \in \mathbb{R}^{H\times H}, B \in \mathbb{R}^{H\times N_{\text{in}}}, C \in \mathbb{R}^{N_{\text{out}}\times H}, D \in \mathbb{R}^{N_{\text{out}}\times N_{\text{in}}}$:

$$x_k = \phi(Ax_{k-1} + Bu_k), \quad y_k = Cx_k + Du_k, \tag{4.6}$$

where several differences need to be emphasized: **(1)** Vanilla-RNNs include a nonlinear normalization function, such as the `softmax` function, on the output $y_k = \psi(Cx_k + Du_k)$ with $D = 0$. Having $D \neq 0$ basically introduces a skip connection in its architecture. **(2)** ϕ represents a nonlinearity function in Vanilla-RNNs, such as the `Tanh` function. In contrast, the recurrence of SSMs is linear (i.e., ϕ is the identity function). **(3)** Bias parameters are not denoted in Eq. 4.6 as they can be incorporated into the MLP blocks following the RNN/SSM layer. **(4)** Compared with Vanilla-RNNs, A and B are parameterized in a peculiar way in SSMs (e.g., Gu, Goel, and Ré, 2022).

Motivated by these similarities and differences, Orvieto *et al.* (2023) argued that it was unclear where the effectiveness enhancement of deep SSMs boost over Vanilla-RNNs came from. Thus, they used the same architecture as S4, DSS, and S5, but replaced the SSM layer in the recurrent core with an RNN. Then, they studied which steps need to be taken to gradually retrieve S4-like effectiveness

Table 4.3 The effect of removing the nonlinearity from the vanilla recurrent cell (see Eq. 4.6) on test accuracy. The term RNN-LIN refers to the identity function ϕ. ✗denotes that the models did not exceed random guessing.

Cell	sCIFAR	LISTOPS	TEXT	RETRIEVAL	PATHFINDER
RNN-RELU	69.7 (0.2)	37.6 (8.0)	88.0 (0.1)	88.5 (0.1)	✗
RNN-TANH	69.9 (0.3)	43.9 (0.1)	87.2 (0.1)	88.9 (0.2)	✗
RNN-LIN	**72.2** (0.2)	**50.4** (0.2)	**89.1** (0.1)	**89.1** (0.1)	✗

on LRA tasks, leading to the architecture presented in Fig. 4.6. Their main goal was to answer the question: "*Can we match the effectiveness and efficiency of deep continuous-time SSMs using deep RNNs?*" In other words, Orvieto *et al.* (2023) sought to demonstrate that Vanilla-RNNs can achieve SOTA performance on LRA tasks when properly initialized and parameterized.

Orvieto *et al.* (2023) divided the development of their models into four main steps, each supported by theoretical considerations and empirical observations. In the remainder of this section, we will elaborate on the fundamental principles and discoveries underlying each step.

Linear Recurrences

Linear RNN outperformed nonlinear RNN variants in the same architecture, as depicted in Table 4.3. These results were unexpected because recurrent nonlinearities were believed to be a key component for the success of RNNs (Siegelmann, 2012; Pascanu *et al.*, 2013; Erichson *et al.*, 2020). For example, single-layer sigmoidal and Tanh RNNs are Turing completeness, which cannot be achieved by the linear variant (Chung and Siegelmann, 2021). In addition to experiment results, Orvieto *et al.* (2023) leveraged a spectral analysis and Koopman operator theory (Koopman and Neumann, 1932), showing that linear RNN layers with nonlinear feedforward blocks are sufficient to approximate highly nonlinear systems.

With their empirical findings and theoretical analysis, Orvieto *et al.* (2023) claimed that the use of linear recurrence in RNNs when coupled with nonlinear MPS does not harm the model expressivity. In contrast, linearity recurrence empowers the ability to parallelize training and inference, as well as to

control how quickly the gradients might vanish or explode directly. Thus, Orvieto *et al.* (2023) dropped nonlinearities in the following steps.

Complex Diagonal Recurrent Matrices

Diagonalizing linear recurrences for computational efficiency has been a dominating feature of all deep SSMs since the introduction of DSS. Correspondingly, Orvieto *et al.* (2023) unrolled the linear recurrence $x_k = Ax_{k-1} + Bu_k$ using the assumption that $x_{-1} = 0 \in \mathbb{R}^H$:

$$\begin{aligned} x_0 &= Bu_0 \\ x_1 &= ABu_0 + Bu_1 \\ x_2 &= A^2Bu_0 + ABu_1 + Bu_2 \\ &\vdots \\ x_k &= \sum_{j=0}^{k-1} A^j Bu_{k-j}. \end{aligned} \tag{4.7}$$

Then, Orvieto *et al.* (2023) decomposed the recurrence as $A = P\Lambda P^{-1}$, where $P \in \mathbb{C}^{H\times H}$ is an invertible matrix and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_H) \in \mathbb{C}^{H\times H}$. Note that the use of the complex field allows representing non-symmetric matrices A in diagonal form. Next, Orvieto *et al.* (2023) plugged the decomposition $A = P\Lambda P^{-1}$ into Eq. 4.7 and multiplied both sides by p^{-1}:

$$\bar{x}_k = \sum_{j=0}^{k-1} \Lambda^j \bar{B} u_{k-j}, \tag{4.8}$$

where $\bar{x}_k := P^{-1}x_k$ and $\bar{B} := P^{-1}B$. Note that the output function in Eq. 4.6 is $y_k = Cx_k + Du_k$. Here, Orvieto *et al.* (2023) defined the output function as:

$$y_k = \mathcal{R}[\bar{C}\bar{x}_k] + Du_k, \tag{4.9}$$

where $\bar{C} := CP^{-1}$ and $\mathcal{R}[\cdot]$ denotes a vector consisting of the real part of the complex-valued vector $\bar{C}\bar{x}_k$. Thus, instead of learning (A, B, C, D), one can equivalently learn $\Lambda, \bar{B}, \bar{C}, D$, where $\Lambda, \bar{B}, \bar{C}$ are complex valued, and Λ is a diagonal matrix.

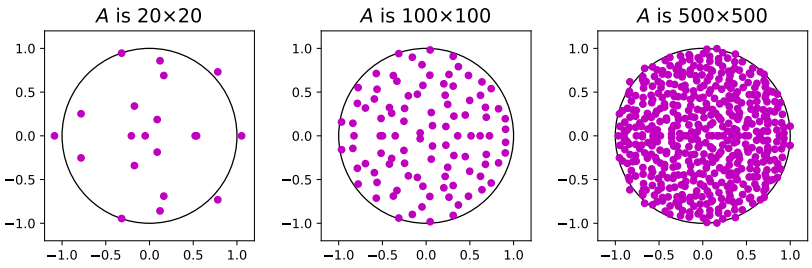

Figure 4.7 Eigenvalues of $A \in \mathbb{R}^{H\times H}$ following Glorot initialization: each entry of A is sampled independently from a Gaussian with mean 0 and variance $1/H$ (Orvieto *et al.*, 2023). The eigenvalues are complex and are represented on the complex plane. The black circle is the unit disk $\{|z| = 1\} \subseteq \mathbb{C}$. The limit behavior (uniform initialization) is predicted by Thm. 4.2.1.

Learning recurrent linear RNNs speed up training and inference. For example, computing the powers of diagonal matrices Λ^j is much easier than exponentiating hidden-to-hidden matrices A^j. Meanwhile, the linearity of recurrences allows the use of parallel scans algorithm (Martin and Cundy, 2017). Moreover, the model is more effective on the `sCIFAR` and `LISTOPS` tasks, as demonstrated in the second row of Table 4.4.

Initialization and Stable Exponential Parameterization

A stable initialization and parameterization are important to model performance. Orvieto *et al.* (2023) mainly achieved this goal in three steps. As a first step, they kept the eigenvalues of the recurrence unchanged when comparing Eq. 4.8 with Eq. 4.6, where A followed Glorot initialization (Glorot and Bengio, 2010). They used a classical result from *random matrix theory*:

Theorem 4.2.1 (Ginibre, 1965; Strong circular law)**.** *Let μ_H be the empirical spectral measure of A_H, where A_H is a real $H \times H$ matrix with i.i.d. Gaussian entries, each with zero mean and variance $1/H$. Then, μ_H converges weakly almost surely as $H \to \infty$ to the uniform probability measure on $|z| \leq 1 \subseteq \mathbb{C}$.*

Supported by Thm. 4.2.1, Fig. 4.7 shows that the spectrum of non-symmetric A is sampled from the unit disk in $\mathbb{C}$ under Glorot initialization. Thus, sampling the eigenvalues of Λ uniformly on the unit disk is equivalent to the Glorot initialization of A. Using the definition of exponential of a complex

number: $\exp(-\nu + i\theta) := e^{-\nu}(\cos(\theta) + i\sin(\theta))$, Orvieto *et al.* (2023) parameterized the diagonal matrix as:

$$\Lambda = \text{diag}(\exp(-\nu + i\theta)), \tag{4.10}$$

where $\nu \in \mathbb{R}^H$ and $\theta \in \mathbb{R}^H$ are the learnable parameters (instead of the real and imaginary parts of Λ).

As the second step, Orvieto *et al.* (2023) kept the norm of eigenvalues of Λ to be close to 1 (at least at initialization). This step is important to avoid quickly vanishing or exploding gradients (Gu, Goel, Gupta, *et al.*, 2022; Gupta *et al.*, 2022). Specifically, since $\bar{x}_k = \sum_{i=0}^{k-1} \Lambda^i \bar{B} u_{k-i}$ in Eq. 4.8, the norm of component j of $\bar{x}$ at timestamp k evolves such that $|x_{k,j}| = \mathcal{O}(\bar{x}_{k,j}) = \mathcal{O}(|\lambda_j|^k)$. Therefore, a sufficient condition to ensure stability (i.e., x_k does not explode or vanish) is therefore $|\lambda_j| = 1, \forall j$.

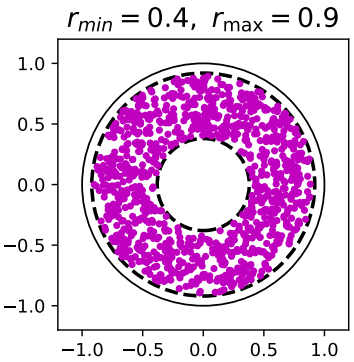

Figure 4.8 Eigenvalues of a diagonal matrix A with entries sampled using Lem. 4.2.1 (Orvieto *et al.*, 2023). For $r_{\min} = 0$, $r_{\max} = 1$, the distribution coincides with Glorot initialization in the limit.

To achieve this goal, Orvieto *et al.* (2023) sampled λ_j uniformly on an annulus in between circles with radii $r_{\min}$ and $r_{\max}$ in $\mathbb{C}$, as shown in Fig. 4.8:

Lemma 4.2.1 (Orvieto *et al.*, 2023; Lem. 3.2)**.** *Let u_1, u_2 be independent uniform random variables on the interval $[0,1]$. Let $0 \leq r_{\min} \leq r_{\max} \leq 1$. Compute $\nu = -\frac{1}{2}\log\left(u_1(r_{\max}^2 - r_{\min}^2) + r_{\min}^2\right)$ and $\theta = 2\pi u_2$. Then $\exp(-\nu + i\theta)$ is uniformly distributed on the annulus in $\mathbb{C}$ between circles of radii $r_{\min}$ and $r_{\max}$.*

By setting $r_{\min} = 0$ and $r_{\max} = 1$, this exponential parameterization takes the effectiveness of `PathFinder` above random chance (50%), as depicted in Table 4.4 (third row).

Table 4.4 Test accuracy of a linear diagonal complex RNNs under different parametrizations of the transition matrix (Orvieto *et al.*, 2023). The term DENSE A denotes linear RNNs in Eq. 4.6. The term Λ REAL + IM denotes the complex diagonal RNNs in Eq. 4.8. The term Λ EXP denotes the parameterization $\Lambda = \text{diag}(\exp(-\nu + i\theta))$ with $r_{\min} = 0, r_{\max} = 1$ in Eq. 4.10. The term Λ STABLE EXP denotes Eq. 4.11. The term + RING INIT denotes a changed initialization where $r_{\min}$ and $r_{\max}$ are tuned. ✗ denotes that the models did not exceed random guessing.

Cell	sCIFAR	LISTOPS	PATHFINDER	Path-X
DENSE A	72.2 (0.2)	50.4 (0.2)	✗	✗
Λ REAL + IM	86.5 (0.1)	58.8 (0.3)	✗	✗
Λ EXP	85.4 (0.7)	**60.5** (0.3)	65.4 (9.0)	✗
Λ STABLE EXP	87.2 (0.4)	59.4 (0.3)	93.5 (0.5)	✗
+ RING INIT	**88.1** (0.0)	59.4 (0.3)	**94.4** (0.3)	✗

As the third step, Orvieto *et al.* (2023) enforced numerical stability by reparameterizing Λ as:

$$\Lambda = \text{diag}\left(\exp\left(-\exp(\nu^{\log}) + i\exp(\theta^{\log})\right)\right), \tag{4.11}$$

where $\nu^{\log}, \theta^{\log} \in \mathbb{R}^H$ are the learnable parameters (instead of the ν, θ in Eq. 4.10), and $\nu^{\log} := \log(\nu), \theta^{\log} := \log(\theta)$ at initialization. As listed in the fourth row of Table 4.4, this change drastically improves the model effectiveness on `Pathfinder`. It seems that the transition from Eq. 4.10 to Eq. 4.11 is primarily driven by empirical considerations: **(1)** The power of exponential parameterization makes training with Adam optimizer (Kingma and Ba, 2014) easier. **(2)** The exponential nonlinearity can increase granularity around the eigenvalues $|\lambda_j| = 1$, stabilizing the training process. **(3)** Orvieto *et al.* (2023) interpreted $\nu^{\log}$ as the vector of log eigenvalue magnitudes, encoding the distance to the origin, and $\theta^{\log}$ as the vector of log eigenvalue *phases*, representing the angle from the vector $1 + 0i$. Empirically, tuning the initialization range of $\nu^{\log}$ and $\theta^{\log}$ can improve model effectiveness, as shown in Table 4.4 (last row). For example, for longer sequence lengths, tuning $r_{\min}$ and $r_{\max}$ or reducing the initialization phase (i.e., tuning the range of $\theta^{\log}$ from $[0, 2\pi]$ to $[0, \pi/50]$) improved convergence on the PathX task.

Normalization

Up to this point, Orvieto *et al.* (2023) still did not solve `PathX`—the hardest task among LRA tasks, with a sequence length of $16k$ tokens (see Table 4.4). During training, they identified an issue

that $\lim_{\varepsilon\to 0}\frac{\mathbb{E}[\|x_{k\to\infty}\|_2^2]}{\mathbb{E}[\|Bu\|_2^2]}=\frac{1}{1-r^2}$, where $\varepsilon=r_{\max}^2-r_{\min}^2$. To obtain this finding, they first compute the magnitude of $\mathbb{E}[\|x_\infty\|_2^2]$:

Proposition 4.2.1 (Orvieto *et al.*, 2023; Proposition 3.3)**.** *Let Λ be diagonal with eigenvalues sampled uniformly on the annulus in $\mathbb{C}$ between circles of radii $r_{\min}<r_{\max}<1$. Then, under constant or white-noise input and Glorot input projection, the squared norm of the state x_k converges as $k\to\infty$ to the following quantity.*

$$\mathbb{E}[\|x_{k\to\infty}\|_2^2]=\frac{1}{r_{\max}^2-r_{\min}^2}\log\left(\frac{1-r_{\min}^2}{1-r_{\max}^2}\right)\mathbb{E}[\|Bu\|_2^2].$$

Let $\varepsilon=r_{\max}^2-r_{\min}^2$ and $\rho=1-r_{\max}^2$, Orvieto *et al.* (2023) found that:

$$\lim_{\varepsilon\to 0}\frac{\mathbb{E}[\|x_\infty\|_2^2]}{\mathbb{E}[\|Bu\|_2^2]}=\lim_{\varepsilon\to 0}\left[\frac{1}{\varepsilon}\log\left(1+\frac{\varepsilon}{\rho}\right)\right]=\lim_{\varepsilon\to 0}\left[\frac{1}{\varepsilon}\left(\frac{\varepsilon}{\rho}+\mathcal{O}(\varepsilon^2)\right)\right]=\frac{1}{\rho}=\frac{1}{1-r^2}. \tag{4.12}$$

The magnitude of x_k will explode if r is close to 1 according to Eq. 4.12. To mitigate this issue, Orvieto *et al.* (2023) tended to normalize the hidden states during training. They considered the one-dimensional setting under white-noise input: let $\Lambda=\lambda\in\mathbb{C}$ and $B=1$:

$$\begin{aligned}\mathbb{E}|x_k|^2&=\left(\sum_{i=0}^{k-1}\lambda^i\mathbb{E}[u_{k-i}]\right)\left(\sum_{j=0}^{k-1}\lambda^j\mathbb{E}[u_{k-j}]\right)^*\\&=\sum_{i,j=0}^{k-1}\lambda^i(\lambda^j)^*\mathbb{E}[u_{k-i}u_{k-j}]\\&=\sum_{i=0}^{k-1}|\lambda|^{2i}\overset{\infty}{\to}\frac{1}{1-|\lambda|^2}.\end{aligned} \tag{4.13}$$

where $*$ denotes conjugation. Therefore, Orvieto *et al.* (2023) used learnable parameter γ initialized element-wisely as $\sqrt{1-|\lambda_j|^2}$. An interesting observation is that Orvieto *et al.* (2023) also tried to set $\gamma_j=\sqrt{1-|\lambda_j|^2}$ in each training iteration and found it to work similarly in practice to a trainable γ.

With this final modification, the hidden states of Linear Recurrent Units (LRUs) are defined as (Orvieto *et al.*, 2023):

$$x_k = \text{diag}(\lambda)x_{k-1} + \gamma \odot Bu_k, \tag{4.14}$$

where $\lambda_j = \exp\left(-\exp(\nu_j^{\log}) + i\exp(\theta_j^{\log})\right)$ and $\gamma_j = \sqrt{1-|\lambda_j|^2}$ at initilization. LRUs finally matched the effectiveness and efficiency of deep SSMs on all LRA tasks, as shown in Fig. 4.6.

4.3 Limitations and Potential Improvements

This section is structured as follows: In Sec. 4.3.1, we discuss the limitations of long-range correlations from the perspective of language modeling. In Sec. 4.3.2, we analyze why previous TNLMs have not been tested on well-accepted NLP datasets and why there exists an effectiveness gap between TNLMs and SOTA LMs. In Sec. 4.3.3, we share our thoughts on possible enhancements for TNLMs. It should be noted that the content of this section is from the author's perspective and may contain inaccuracies.

4.3.1 Long-Range Correlations

The research objectives of Pestun and Vlassopoulos, 2017; Pestun *et al.*, 2017 hinged on the definition of long-range correlations (see Def. 4.1.2), which has twofold shortcomings.

Determining whether the correlation function $C(S)$ follows a power decay is susceptible to itself and experimental factors. **(1)** The correlation functions themselves may cause issues. Not all correlation functions can precisely capture the long-range correlations that underlie language, even when such phenomena are present (Tanaka-Ishii, 2021). Evidence of this is that the power-law type phenomenon is sometimes better characterized by distribution families other than power laws (Clauset *et al.*, 2009). **(2)** This procedure involves fitting the parameter γ using an appropriate method, with the validation of a goodness-of-fit test, such as the Kolmogorov-Smirnov test (Massey Jr, 1951). During this process,

Table 4.5 Example of the datasets used in previous TNLMs. Column headings are as follows: "Paper" are works that used tensor networks (see Table 4.1); "Dataset" refers to the training datasets used in the paper; $|V|$ denotes the vocabulary size; L denotes the maximum length of samples; and $|T|$ denotes the number of training samples. The notation "-" indicates that the vocabulary size in each position of the samples varies.

Paper	Dataset	$\|V\|$	L	$\|T\|$
Han *et al.* (2018)	Bars and Stripes (MacKay and Mac Kay, 2003)	2	16	30
	MNIST (LeCun *et al.*, 1998)	2	784	$1k$
Miller *et al.* (2021)	Tomita grammars (Tomita, 1982)	2	30	$10k$
	Motzkin grammars (Alexander *et al.*, 2021)	3	50	$10k$
	Email addresses (Radev, 2008)	38	30	$4k$
Novikov *et al.* (2021)	POWER (Dua and Graff, 2017)	-	6	1659k
	GAS (Fonollosa *et al.*, 2015)	-	8	$852k$
	HEPMASS (Baldi *et al.*, 2016)	-	21	$315k$
	MINIBOONE (Roe *et al.*, 2005)	-	43	$29k$
	BSDS300 (Martin *et al.*, 2001)	-	63	$1000k$

the goodness-of-fit test could be affected by experimental factors, including the text type, text length, and selected sequence length (see Fig. 4.1).

The second shortcoming is that Def. 4.1.2 has limited applicability to LMs. There are three reasons for this statement. **(1)** We can view Def. 4.1.2 as a binary metric that states whether or not a text generated by a model has long-range correlations. However, using a metric that indicates how much a model can capture long-range correlations would be more natural for deep learning communities. **(2)** According to Def. 4.1.2, we cannot conclude that if an LM generates text with power-law decay correlations, it can always produce text that more closely resembles human language in every scenario. This uncertainty is due to not all text written by humans having power-law decay correlations. For instance, a corpus consisting of only $1k$ words may exhibit a different decay pattern of correlations than that of a corpus containing billions of words. **(3)** It is unclear if better modeling of long-range correlations can improve model effectiveness in long-context scenarios. No theoretical or empirical evidence exists to establish this relationship. Deep learning communities typically employ "benchmarks" to ascertain whether a model can perform effectively in long-context scenarios. For example, LMs have been frequently tested on LRA tasks (Tay *et al.*, 2020) regarding their long-range reasoning abilities (see Sec. 4.2.2).

4.3.2 Effectiveness Gap

Prior TNLMs lacked comparisons with state-of-the-art (SOTA) LMs on well-accepted NLP datasets (e.g., PTB; Marcinkiewicz, 1994). The reasons for this absence have yet to be explicitly described in prior research. We attribute the absence to the properties of TNLMs, and this section analyzes two possible contributing factors.

Computational and Memory Cost

One distinction between NLP datasets and those used to evaluate TNLMs is the size of their vocabularies. TNLMs have only been evaluated on the datasets with small-scale vocabulary size ($|V| < 38$), as exemplified in Table 4.5. On the other hand, NLP datasets tend to possess a larger vocabulary size; for example, WikiText-2 (Merity *et al.*, 2016) has a vocabulary size of $30k$. Unexpectedly, our observations indicate that modeling the joint probability of sequences with large-scale vocabulary sizes is theoretically plausible for TNLMs. In other words, the vocabulary size does not impede prior research applying TNLMs to NLP datasets.

To support this statement, we have two observations. **(1)** The increase in the vocabulary size will only lead to a linear growth in the computation cost of Z in MPS. Recall that after representing the joint probability of the sequence as a wave function $\mathbf{x}$ (see Eq. 4.3), previous TNLMs employ a particular TN to represent $\mathbf{x}$; otherwise, the computation cost of $\mathbf{x}$ is $\mathcal{O}(|V|^L)$, which is intractable (see Sec. 3.2.1). If the model architecture is based on MPS, the joint probability $\tilde{P}(\mathbf{x})$ has the computation cost of $\mathcal{O}(LR^2)$ (Miller *et al.*, 2021), and the normalization factor Z has the cost of $\mathcal{O}(|V|LR^3)$ (Orús, 2014), where R is the maximum TT-rank. **(2)** $\tilde{P}(\mathbf{x})$ is the result of tensor contractions of TT-cores in the TT-format (see Eq. 3.4); elements of TT-cores are learnable parameters. Thus, increasing large-scale vocabulary sizes does not cause data underflow when computing $\tilde{P}(\mathbf{x})$. Taking the two observations together, we suggest that modeling datasets with large-scale vocabulary size is plausible for TNLMs.

Mismatched Paradigm

We conjecture that the primary factor is the mismatch between the paradigm of TNLMs and neural networks. We highlight that, unlike neural network LMs, TNLMs aim to learn the empirical distribution defined by the training set in Sec. 4.1.4. Their training objective may cause TNLMs to underperform neural network LMs on most NLP tasks. To support this claim, let us consider an example that a model has fully learned the empirical distribution of the training set and can generate sequences that exactly follow this distribution. However, the model will generalize poorly if the empirical distribution of the training and test sets differs slightly. For example, it would encounter out-of-vocabulary problems (i.e., the occurrence of words in the test set is absent in the training set). In contrast, neural network models, such as the Transformers, demonstrate their generalization capabilities on the compositional benchmarks (Ontanon *et al.*, 2022). Behind the success of neural network models, predicting masked words or the next word in a sentence during training should be a significant contributor. For instance, this approach may encourage the LMs to acquire contextual information about words (Devlin *et al.*, 2018).

Learning the empirical distribution of sequences defined by the training set may only be suitable for certain language modeling tasks. Such tasks require models to generate sequences that adhere strictly to a predefined set of rules, and the training and test set must strictly obey these rules. In this way, models may fully grasp the rigid rules from modeling the empirical distribution of the training set, and be effective on the test set. For instance, Miller et al. (2021) trained their model on $4k$ email addresses and evaluated its ability by generating addresses in the correct format. Additionally, they used the model to generate sequences compliant with Tomita grammars (Tomita, 1982) or Motzkin grammars (Alexander *et al.*, 2021). Their model outperformed Transformers and LSTMs in these tasks.

We speculate that the training objective of TNLMs should align with that of SOTA LMs, to achieve SOTA effectiveness in language modeling. However, altering the training objectives necessitates a shift in the paradigm for establishing TNLMs. In Ch. 5, we will introduce our innovations in this paradigm shift.

4.3.3 Potential Improvements

Besides the mismatched paradigm, we outline four possible directions for enhancing the effectiveness of TNLMs in language modeling tasks. The first two directions will undergo evaluation in our experimental assessments in Ch. 6. The latter two directions remain as prospects for future endeavors. We explain the rationale behind our selection of these directions in the following paragraphs. Note that these suggestions are conjectural and stem from preliminary observations, which should evolve with further research and experiments.

Thoughtful parameterization of tensor networks. Effective parameterization and initialization strategies can mitigate gradients vanishing or exploding, allowing models to efficiently and effectively capture long-range dependencies (see Sec. 4.2). Just as careful parameterization and initializations are vital for RNNs, similar attention must be directed toward TNs. For example, if we use tensor trains, the parameterization of TT-cores demands careful consideration. In Ch. 6, we shall modify the parametrization and initializations of LRUs for our models.

Treating tensor networks as layers. RNNs have been integrated as layers within models rather than stand-alone entities, similar to the role of attention layers in Transformers. While challenging for theoretical analysis, this approach substantially enhances model effectiveness. The current paradigm of TNLMs, however, uses one type of TNs as the entire model architecture (see Sec. 4.1.3). Therefore, we propose treating TNs as integral building blocks within neural architectures. While TNs inherently exhibit linearity, their integration with nonlinear blocks can better capture complex patterns. In Ch. 6, we shall try to use the S4 architectures and replace the RNN layers with our proposed models.

Diverse roles of tensor networks. The role of TNs within the broader architecture offers multiple possibilities. This thesis focuses on using tensor trains as replacements for RNNs within a larger structure. Alternatively, TNs could be integrated into other model components, such as word embedding layers. Recent research has explored tensorized word embeddings, treating a word embedding as a composite of morpheme vectors through tensor products (Gan *et al.*, 2022). Thus, we could use

TNs, such as TTNs or MERA (see Sec. 3.2.1), to encode linguistic hierarchies, enhancing context comprehension. As this direction diverges from our current emphasis on tensor trains replacing RNNs, we consider it as prospective future work.

Leveraging normalization functions. The normalization factor (i.e., Z) for joint probabilities is a defining characteristic of TNLMs compared with neural network LMs, as outlined in Sec. 4.1.3. However, its practical benefits remain largely unexplored, despite its theoretical significance. In neural network models, various techniques such as layer/batch normalization (Ba *et al.*, 2016; Ioffe and Szegedy, 2015) are employed for normalization. Similarly, in LRUs (discussed in Sec. 4.2.2), the introduction of the normalization factor γ significantly enhances model effectiveness. Therefore, investigating the impact of Z on model effectiveness could be viewed as a research direction. Integrating Z to manage input lengths within TNLMs could potentially mitigate challenges arising from longer sequences, thereby enhancing generalization and robustness. However, given our model's emphasis on learning conditional probabilities (as introduced in Ch. 5), the integration of Z remains unclear. We defer this aspect for further research.

5 Language Modeling with Tensor Trains

We organize this chapter as follows: Sec. 5.1 introduces our research objectives when devising tensor network LMs. We propose a class of tensor-train LMs (TTLMs) in Sec. 5.2. We illustrate the computation process of TTLMs and their relationship with earlier recurrent-based models in Sec. 5.3. Last, we specify our attempts at developing an effective class of TTLMs in Sec. 5.4.

5.1 Research Objectives

Tensor networks have been widely used to analyze neural network models, including parameters compression (Novikov *et al.*, 2015), expressive power (Cohen *et al.*, 2016; Khrulkov *et al.*, 2018), and depth efficiency for long-term memory (Levine *et al.*, 2018). On the experimental side, despite arguments about their potential advantage in capturing long-range correlations in human language (Pestun and Vlassopoulos, 2017; Pestun *et al.*, 2017), the effectiveness of tensor networks remains unverified (see Sec. 4.1.2). In Ch. 4, we argue that the current paradigm of tensor network LMs diverges from that of neural network LMs, primarily attributing the effectiveness gap. Specifically, tensor network LMs typically learn the empirical frequency of training instances (see Sec. 4.3.2), but current neural network LMs often learn to predict a single word in each instance (see Sec. 2.3.1 and Sec. 2.3.2).

Our research objective is to *narrow the effectiveness gap between tensor networks and neural network models in language modeling tasks*. To achieve this goal, we introduce a class of tensor-train LMs (i.e., TTLMs) in Sec. 5.2. TTLMs represent sequences in tensor space, encode joint probabilities as wave functions, and employ tensor trains to learn conditional probabilities. Thus, TTLMs inherit the fundamental properties of tensor networks, such as *linearity* and *multiplicativity*.

However, TTLMs in its primary stage could not rival SOTA models. For instance, the initializations and parametrizations of RNNs in long-range scenarios have been extensively studied, as discussed in Sec. 4.2. Thus, our subsequent step is designing an effective class of TTLMs in long-range scenarios. This exploration is detailed in Sec. 5.4.

5.2 Tensor Train Language Models

This section is structured as follows: In Sec. 5.2.1, we recap the notation and concepts of probability distributions used throughout this thesis. In Sec. 5.2.2, we propose a class of tensor-train language models (TTLMs), a model architecture that uses tensor trains to learn conditional probabilities of sequences during training.

5.2.1 Notation and Basic Concepts

Before presenting TTLMs, we provide a concise reminder of the notation and terminology used throughout this thesis. We refer to Ch. 3 for a detailed introduction to the notation, diagrams, and tensor operations.

We denote a sequence of $L \in \mathbb{N}_+$ discrete random variables as $\mathbf{X} = [X^{(1)}, X^{(2)}, \cdots, X^{(L)}]$, where each $X^{(i)} \in V$ for all $i \in [L]$ and V denotes the vocabulary set defined by a training set. The probability distribution of $\mathbf{X}$ is denoted as $P : V^L \to \mathbb{R}_{[0,1]}$. Let $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$ be a fixed-length sample according to $\mathbf{X}$. We use $P(\mathbf{x})$ to represent the probability mass function that takes the value $\mathbf{x}$, which is short for $P\left(X^{(1)} = x^{(1)}, X^{(2)} = x^{(2)}, \cdots, X^{(L)} = x^{(L)}\right)$.

Let $t \leq L$ be a positive integer. We use $x^{(1:t-1)}$ as shorthand for the ordered list $[x^{(1)}, x^{(2)}, \cdots, x^{(t-1)}]$ in $\mathbf{x}$. We decompose the joint probability into conditional probabilities using the chain rule of probability (Bahl *et al.*, 1983) as follows: $P(\mathbf{x}) = \prod_{t=1}^{L} P(x^{(t)} | x^{(1:t-1)})$, where $P(x^{(t)} | x^{(1:t-1)})$ is shorthand for the conditional probability $P(X^{(t)} = x^{(t)} | X^{(1:t-1)} = x^{(1:t-1)})$. Note that the conditional probability distribution of $X^{(t)}$ given $X^{(1:t-1)} = x^{(1:t-1)}$ is denoted as $P_{x^{(1:t-1)}} : V \to \mathbb{R}_{[0,1]}$.

The ground-truth labels for probability distributions vary across research fields. Density estimation models typically assume that the ground-truth label for P is the empirical frequencies of $\mathbf{X}$ defined by a training set (see Sec. 4.1.3). Neural network LMs often assume that the ground-truth label for $P_{x^{(1:t-1)}}$ is the one-hot vector representing $x^{(t)}$ (see Sec. 2.3.1). To differentiate the ground-truth labels from those defined by a language model, we denote the probability distribution of $\mathbf{X}$ as $\tilde{P}$ and the conditional probability distribution of $X^{(t)}$ given $X^{(1:t-1)} = x^{(1:t-1)}$ as $\tilde{P}_{x^{(1:t-1)}}$.

5.2.2 Model Architecture.

TTLMs encode the joint probability distribution (i.e., $\tilde{P}$) into a quantum wave function. Measurement will collapse $\tilde{P}$ and generate a fixed-length sample $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$ with a probability proportional to $|\Psi(\mathbf{x})|^2$ as follows:

$$\tilde{P}(\mathbf{x}) = \psi\left(|\Psi(\mathbf{x})|^2\right), \tag{5.1}$$

where ψ is a normalization function that ensures the nonnegativity of $\tilde{P}(\mathbf{x})$ and that the probabilities sum to 1. For example, prior studies typically define $\psi(|\Psi(\mathbf{x})|^2) = \frac{|\Psi(\mathbf{x})|^2}{Z}$, where $Z = \sum_{\mathbf{x} \in V^L} |\Psi(\mathbf{x})|^2$ (see Sec. 4.1.3).

We then present two propositions, accompanied by their proofs, that demonstrate how TTLMs compute the joint probability in Prop. 5.2.1, as well as the conditional probability distribution in Prop. 5.2.2.

Proposition 5.2.1. *Let* $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$ *be represented as* $\Phi(\mathbf{x})$ *in tensor space. Given a set of tensors* $\mathbf{G}^{(t)} \in \mathbb{R}^{R_{t-1} \times |V| \times R_t}$ $(t \in [L], \quad R_0 = R_L = 1)$*, the Tensor Train Language Models compute the probability of* $\mathbf{x}$ *as follows:*

$$\tilde{P}(\mathbf{x}) = \psi\left((\mathbf{f}^{(1)})^\top \mathbf{G}^{(1)} (\mathbf{f}^{(2)})^\top \mathbf{G}^{(2)} \cdots (\mathbf{f}^{(L)})^\top \mathbf{G}^{(L)}\right). \tag{5.2}$$

Proof. We define the representation of $\mathbf{x}$ in tensor space. Given $\mathbb{V} = \mathbb{R}^{|V|}$ and the tensor space $\mathbb{V}^{\otimes L} = \underbrace{\mathbb{V} \otimes \cdots \otimes \mathbb{V}}_{L}$, a sample $\mathbf{x}$ according to $\mathbf{X}$ is represented as follows:

$$\begin{aligned}\Phi(\mathbf{x}) &= \mathbf{f}(x^{(1)}) \otimes \mathbf{f}(x^{(2)}) \cdots \otimes \mathbf{f}(x^{(L)}) \\ &= \bigotimes_{i=1}^{L} \mathbf{f}(x^{(i)}),\end{aligned} \tag{5.3}$$

where $\mathbf{f} \in \mathbb{V}$ is one-hot encoding and $\Phi(\mathbf{x}) \in \mathbb{V}^{\otimes L}$. For the sake of notation simplicity, we use $\mathbf{f}^{(i)}$ to denote $\mathbf{f}(x^{(i)})$ throughout the rest of the thesis.

Suppose $\mathcal{A}$ is a tensor of the same shape as $\Phi(\mathbf{x})$ in the tensor space $\mathbb{V}^{\otimes L}$. We define the measurement of $\Psi(\mathbf{x})$ as:

$$\begin{aligned}|\Psi(\mathbf{x})|^2 &= \langle \mathcal{A}, \Phi(\mathbf{x}) \rangle \\ &= \sum_{i_1, i_2, \cdots, i_L = 1}^{|V|} \mathcal{A}_{i_1, \cdots, i_L} \cdot \Phi(\mathbf{x})_{i_1, \cdots, i_L}\end{aligned} \tag{5.4}$$

where $\langle \cdot \rangle$ denotes the inner product of two same-sized tensors (see Def. 3.1.5). Note that TTLMs learn $\tilde{P}(\mathbf{x})$ by tuning the parameters of $\mathcal{A}$ during training. At this step, the computation and memory cost of $\mathcal{A}$ is $\mathcal{O}(|V|^L)$. Our next step is representing $\mathcal{A}$ in the TT-format.

Let a sequence of $L \in \mathbb{N}_+$ words denote as $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$, where each $x^{(i)} \in V$ for all $i \in [L]$. We denote the word indices in $\mathbf{x}$ as $w_1, w_2, \cdots, w_L$, where $w_i \in [|V|]$. The TT-format of $\mathcal{A}_{w_1 w_2 \cdots w_L}$ is represented as follows (Oseledets, 2011):

$$\begin{aligned}\mathcal{A}_{w_1 w_2 \ldots w_L} &= \underbrace{\mathbf{G}^{(1)}_{:, w_1}}_{1 \times R_1} \underbrace{\mathbf{G}^{(2)}_{:, w_2, :}}_{R_1 \times R_2} \cdots \underbrace{\mathbf{G}^{(L)}_{:, w_L}}_{R_{L-1} \times 1} \\ &= \sum_{\alpha_1, \cdots, \alpha_{L-1}}^{R_t} \mathbf{G}^{(1)}_{w_1 \alpha_1} \mathbf{G}^{(2)}_{\alpha_1 w_2 \alpha_2} \cdots \mathbf{G}^{(L)}_{\alpha_{L-1} w_L},\end{aligned} \tag{5.5}$$

where the tensors $\mathbf{G}^{(t)} \in \mathbb{R}^{R_{t-1} \times |V| \times R_t}$ ($t = 1, \cdots, L$, $R_0 = R_L = 1$ by definition) are called the *TT-cores*, and R_k for $k = 0, \cdots, L$ are called the *TT-ranks* (see Sec. 3.2.2 for a detailed introduction of TT-decompositions).

Since $\mathbf{f}^{(t)}$ is a one-hot vector, we represent the elements of TT-cores in Eq. 5.5 as:

$$\mathbf{G}^{(t)}_{w_t \alpha_{t-1} \alpha_t} = \sum_{i_t=1}^{|V|} f^{(t)}_{i_t} \mathbf{G}^{(t)}_{i_t \alpha_{t-1} \alpha_t}, \tag{5.6}$$

where we permute the shape of $\mathbf{G}^{(t)}$ from $R_{t-1} \times |V| \times R_t$ in Eq. 5.5 to $|V| \times R_{t-1} \times R_t$ in Eq. 5.6. We refer to Sec. 3.1.2 for an explanation of the permutation operation.

Using Eq. 5.4, Eq. 5.5 and Eq. 5.6, we derive Eq. 5.2 from Eq. 5.1 as follows:

$$\begin{aligned}
\tilde{P}(\mathbf{x}) &= \psi\left(|\Psi(\mathbf{x})|^2\right) \\
&= \psi\left(\langle \mathcal{A}, \Phi(\mathbf{x}) \rangle\right) \\
&= \psi\left(\mathcal{A}_{w_1 w_2 \ldots w_L}\right) \\
&= \psi\left(\sum_{i_1,\cdots,i_L=1}^{|V|} \sum_{\alpha_1,\cdots,\alpha_{L-1}=1}^{R_t} f^{(1)}_{i_1} \mathbf{G}^{(1)}_{i_1 \alpha_1} f^{(2)}_{i_2} \mathbf{G}_{\alpha_1 i_2 \alpha_2} \cdots f^{(L)}_{i_L} \mathbf{G}^{(L)}_{\alpha_{L-1} i_L}\right) \\
&= \psi\left((\mathbf{f}^{(1)})^\top \mathbf{G}^{(1)} (\mathbf{f}^{(2)})^\top \mathbf{G}^{(2)} \cdots (\mathbf{f}^{(L)})^\top \mathbf{G}^{(L)}\right).
\end{aligned}$$

Note that $\langle \mathcal{A}, \Phi(\mathbf{x}) \rangle = \mathcal{A}_{w_1 w_2 \ldots w_L}$ because we define $\Phi(\mathbf{x})$ as the tensor product of one-hot vectors in Eq. 5.3. Thus, the inner product yields a single nonzero element $\mathcal{A}_{w_1 w_2 \ldots w_L}$.

□

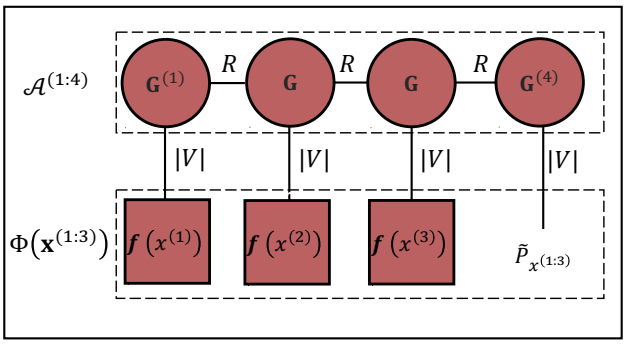

Figure 5.1 Example of Recursive calculation of conditional probability in TTLMs. According to Eq. 5.9, the conditional probability distribution of word $x^{(4)}$ is computed as: $\tilde{P}_{x^{(1:3)}} = \psi((\mathbf{h}^{(3)})^\top \mathbf{G}^{(4)})$.

An essential property of Prop. 5.2.1 is that it naturally combines the learnable parameters $\mathcal{A}$ and input data $\Phi(\mathbf{x})$ together, thereby defining a basic model architecture to compute $\tilde{P}(\mathbf{x})$ in the TT-format. However, as discussed in Sec. 4.3.2, previous tensor network LMs typically use the empirical frequency of $\mathbf{x}$ defined by a training set as the ground-truth label for $\tilde{P}(\mathbf{x})$. In contrast, current neural network LMs often learn to predict a single word in $\mathbf{x}$ (see Sec. 2.3.1 and Sec. 2.3.2).

To align with the paradigm of neural network LMs, we leverage the recursive property of TTLMs and devise the models to learn the conditional probability distributions during training, in Prop. 5.2.2. We provide an example of a recursive calculation of conditional probability in Fig. 5.1.

Proposition 5.2.2. *Let* $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$ *be represented as* $\Phi(\mathbf{x})$ *in tensor space, and the time step index* $t \in [L]$ *be the number of iterations when recursively unfolding* $\mathcal{A}$ *to obtain* $\mathbf{G}^{(t)}$ *during the TT-decomposition given Thm. 3.2.1. Given a set of tensors* $\mathbf{G}^{(t)} \in \mathbb{R}^{R_{t-1} \times |V| \times R_t}$ *(*$t \in [L], \quad R_0 = R_L = 1$*), the conditional probability distribution of* $X^{(t)}$ *given* $X^{(1:t-1)} = x^{(1:t-1)}$ *in the Tensor Train Language Models satisfies:*

$$\tilde{P}_{x^{(1:t-1)}} = \psi\left((\mathbf{f}^{(1)})^\top \mathbf{G}^{(1)} (\mathbf{f}^{(2)})^\top \mathbf{G}^{(2)} \cdots (\mathbf{f}^{(t-1)})^\top \mathbf{G}^{(t-1)} \mathbf{G}^{(t)}\right). \tag{5.7}$$

Proof. When recursively unfolding the calculation of TTLMs in Prop. 5.2.1, $\mathbf{G}^{(t)}$ has two sources of "input": the information from the previous recursive unfolding and input data $\mathbf{f}^{(t)}$ (see Eq. 5.12 for a detailed explanation of this recursive property).

From this perspective, $\mathbf{G}^{(t)}$ acts as a bilinear map $\mathbf{G}^{(t)} : \mathbb{R}^{|V|} \times \mathbb{R}^{R_{t-1}} \to \mathbb{R}^{R_t}$. We regard the information in the previous step as a hidden state $\mathbf{h}^{(t)}$, given by:

$$\mathbf{h}^{(t)} = (\mathbf{h}^{(t-1)})^\top \mathbf{G}^{(t)} \mathbf{f}^{(t)}, \tag{5.8}$$

where $\mathbf{f}^{(t)}$, $\mathbf{G}^{(t)}$, and $\mathbf{h}^{(t-1)}$ are contracted and the indices of $\mathbf{G}^{(t)}$ are permuted from $\mathbb{R}^{R_{t-1} \times |V| \times R_t}$ to $\mathbb{R}^{R_{t-1} \times R_t \times |V|}$.

We decompose $\tilde{P}(\mathbf{x})$ into conditional probabilities using the chain rule (Bahl *et al.*, 1983) as $\tilde{P}(\mathbf{x}) = \prod_{t=1}^{L} \tilde{P}(x^{(t)} | x^{(1:t-1)})$. The conditional probability distributions in TTLMs are defined as follows:

$$\tilde{P}_{x^{(1:t-1)}} = \psi\left((\mathbf{h}^{(t-1)})^\top \mathbf{G}^{(t)}\right), \tag{5.9}$$

where $\mathbf{G}^{(t)} \in \mathbb{R}^{|V| \times R_{t-1}}$ is the last TT-core in TT-format at time t. Note that Eq. 5.9 can be derived from Eq. 5.4, if we unfold its $\mathbf{h}^{(t-1)}$:

$$\begin{aligned} \tilde{P}_{x^{(1:t-1)}} &= \psi\left(\langle \mathcal{A}^{(1:t)}, \Phi(\mathbf{x}^{(1:t-1)}) \rangle_{t-1}\right) \\ &= \psi\left(\underbrace{(\mathbf{f}^{(1)})^\top \mathbf{G}^{(1)} (\mathbf{f}^{(2)})^\top \mathbf{G}^{(2)} \cdots (\mathbf{f}^{(t-1)})^\top \mathbf{G}^{(t-1)}}_{\mathbf{h}^{(t-1)}} \mathbf{G}^{(t)}\right), \end{aligned} \tag{5.10}$$

where $\langle \cdot \rangle_{t-1}$ denotes the "generalized inner product" (see Def. 3.1.6); $\mathcal{A}^{(1:t)} \in \mathbb{V}^{\otimes t}$; and $\Phi(\mathbf{x}^{(1:t-1)}) = \bigotimes_{i=1}^{t-1} \mathbf{f}(x^{(i)}) \in \mathbb{V}^{\otimes t-1}$; ψ is a normalization function that ensures that $\tilde{P}_{x^{(1:t-1)}}$ is nonnegative and that the conditional probabilities sum to 1. Note that ψ in Eq. 5.9 differs from that of Eq. 5.1. Although both functions ensure nonnegative elements summing up to 1, their choices differ. **(1)** In Eq. 5.1, ψ normalizes the joint probability and is commonly defined as $\psi(|\Psi(\mathbf{x})|^2) = \frac{|\Psi(\mathbf{x})|^2}{Z}$, where $Z = \sum_{\mathbf{x} \in V^L} |\Psi(\mathbf{x})|^2$ according to prior research (see Sec. 4.1.3). At this stage, the nonlinearity of ψ should not affect the linearity of tensor trains. **(2)** In Eq. 5.9, ψ is used to normalize the conditional probabilities. For the use of TTLMs as a component in a larger architecture, ψ can be chosen as a constant scaling function to preserve linearity; for stand-alone use of TTLMs, ψ can be chosen to be any appropriate activation function—in the remainder of the thesis, we use the `softmax` function (Bridle, 1990).

□

The similarities and differences between the two propositions are concluded as follows: **(1)** Prop. 5.2.1 defines how TTLMs calculate the probability of an instance $\mathbf{x}$ during training and inference. **(2)** Prop. 5.2.2 shows how TTLMs compute the conditional probability distribution of $X^{(t)}$ given $X^{(1:t-1)} = x^{(1:t-1)}$. **(3)** Both propositions represent $\mathbf{x}$ in the tensor space $\mathbb{V}^{\otimes L}$ and encode the probability distribution of $\mathbf{x}$ into a wave function.

5.3 Worked Computations and Relation to Previous Models

In this section, we aim to assist readers in understanding TTLM: Sec. 5.3.1 presents an illustrative example of the computation of TTLMs. Sec. 5.3.2 explains TTLM's relation to previous models. This section serves pedagogical purposes; readers interested in effective TTLM variants can skip it.

5.3.1 Example of Computation in TTLMs

This section illustrates TTLM's calculation of the probability of $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(L)}]$ in Eq. 5.2, in order to help readers understand the roles of TT-cores.

For pedagogical reasons, we make the following assumptions: **(1)** We adopt the convention of considering a special class of tensor trains (Khrulkov *et al.*, 2018; Miller *et al.*, 2021), where all intermediate TT-cores are equal to each other, denoted as $\mathbf{G} = \mathbf{G}^{(2)}, \ldots, \mathbf{G}^{(t-1)} \in \mathbb{R}^{R \times |V| \times R}$ (which suggests that all TT-ranks equal to R). **(2)** We assume that $\mathbf{f}(x^{(1)})$ is a one-hot vector, with $f(x^{(1)})_1 = 1$ and all other elements being 0. This setup allows us to clearly show the first part of Eq. 5.2. For the sake of simplicity in notation, we use $\mathbf{f}^{(t)}$ to represent $\mathbf{f}(x^{(t)})$; and $f_i^{(t)}$ represents the ith component of $\mathbf{f}^{(t)}$ where $i \in [|V|]$.

The computation of Eq. 5.2 at time t is composed of three parts. The first part is $\mathbf{f}^{(1)^\top}\mathbf{G}^{(1)}$, which is computed as follows:

$$\begin{aligned}
\mathbf{f}^{(1)^\top}\mathbf{G}^{(1)} &= \left[f_1^{(1)}, f_2^{(1)}, \dots f_{|V|}^{(1)}\right] \begin{bmatrix} \mathbf{G}_{11}^{(1)} & \mathbf{G}_{12}^{(1)} & \dots & \mathbf{G}_{1R}^{(1)} \\ \mathbf{G}_{21}^{(1)} & \mathbf{G}_{22}^{(1)} & \dots & \mathbf{G}_{2R}^{(1)} \\ \dots & \dots & \dots & \dots \\ \mathbf{G}_{|V|1}^{(1)} & \mathbf{G}_{|V|2}^{(1)} & \dots & \mathbf{G}_{|V|R}^{(1)} \end{bmatrix} \\
&= \left[\mathbf{G}_{11}^{(1)}, \mathbf{G}_{12}^{(1)}, \cdots, \mathbf{G}_{1R}^{(1)}\right]^\top \\
&= \mathbf{h}^{(1)},
\end{aligned}$$

where $\mathbf{G}^{(1)} \in \mathbb{R}^{|V| \times R}$ and $\mathbf{h}^{(1)} \in \mathbb{R}^R$. Note that we set $f_1^{(1)} = 1$ while $f_i^{(1)} = 0, \quad \forall i \in \{2, \cdots, |V|\}$.

The second part of Eq. 5.2 is the intermediate states: $(\mathbf{f}^{(i-1)})^\top \mathbf{G}\mathbf{h}^{(i)}$ where $i \in \{2, 3, \cdots, t-1\}$ defined in Eq. 5.8. For example, $\mathbf{h}^{(2)}$ is computed as follows:

$$\mathbf{h}^{(2)} = (\mathbf{h}^{(1)})^\top \mathbf{G}\mathbf{f}^{(2)},$$

where the indices of $\mathbf{G}$ are permuted from $\mathbb{R}^{R \times |V| \times R}$ to $\mathbb{R}^{R \times R \times |V|}$. Note that the intermediate TT-cores $\mathbf{G}$ are third-order tensors, unlike the matrices $\mathbf{G}^{(1)}$ and $\mathbf{G}^{(t)}$.

The last part of Eq. 5.2 outputs $\tilde{P}_{x^{(1:t-1)}}$ at time t, which is defined in Eq. 5.9 as follows:

$$
\begin{aligned}
\tilde{P}_{x^{(1:t-1)}} &= \psi\left((\mathbf{h}^{(t-1)})^{\top}\mathbf{G}^{(t)}\right) \\
&= \psi\left(\left[h_1^{(t-1)}, h_2^{(t-1)}, h_R^{(t-1)}\right]\begin{bmatrix} \mathbf{G}_{11}^{(t)} & \mathbf{G}_{12}^{(t)} & \cdots & \mathbf{G}_{1R}^{(t)} \\ \mathbf{G}_{21}^{(t)} & \mathbf{G}_{22}^{(t)} & \cdots & \mathbf{G}_{2R}^{(t)} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{G}_{|V|1}^{(t)} & \mathbf{G}_{|V|2}^{(t)} & \cdots & \mathbf{G}_{|V|R}^{(t)} \end{bmatrix}\right) \\
&= \psi\left(\begin{bmatrix} \sum_{i=1}^{R}\mathbf{G}_{i1}^{(t)}h_1^{(t-1)} \\ \sum_{i=1}^{R}\mathbf{G}_{i2}^{(t)}h_2^{(t-1)} \\ \cdots \\ \sum_{i=1}^{R}\mathbf{G}_{iR}^{(t)}h_R^{(t-1)} \end{bmatrix}\right).
\end{aligned}
$$

Observing the calculation, $\mathbf{G}^{(1)}$, $\mathbf{G}$ and $\mathbf{G}^{(t)}$ theoretically have no parameters in common. Furthermore, their roles in TTLMs are different: $\mathbf{G}^{(1)}$ is an input-to-hidden matrix for words at the first position in the sequence $\mathbf{X}$; Intermediate TT-cores deal with two sources of information (i.e., previous hidden states and input data); and $\mathbf{G}^{(t)}$ is a hidden-to-output matrix, extracting the evidence provided in $\mathbf{h}^{(t-1)}$ and generates a probability distribution over vocabulary. **(2)** The equivalence between Eq. 5.7 and Eq. 5.10 assumes that the hidden-to-output matrix is always the last TT-core, instead of other TT-cores. In other words, we shall always use the last TT-core as the hidden-to-output matrix during training, to satisfy the definition of TTLMs.

5.3.2 Relation to Previous Models

This section illustrates the relationship between TTLMs and previous models.

TTLMs and tensor network LMs share the similarity of representing the joint probability of sequences as wave functions with a normalization factor ψ. The key advantage of TTLMs over previous tensor network LMs lies in their training paradigm. We argue that LMs should predict individual words in a sequence rather than learning the empirical frequency of sequences in a training set (see Sec. 4.3.2). To achieve this, we leverage the recursive property of tensor trains, devising TTLMs to learn the

conditional probability of sequences. This paradigm shift allows us to use "teacher forcing" (Marcus, 1998) to learn the parameters of TT-cores. By adopting this training approach, our model is expected to achieve improved effectiveness on natural language datasets.

TTLMs and RNNs exhibit both similarities and differences. **(1)** As for differences, the dimensions of hidden states are typically fixed in RNNs, and a shared hidden-to-hidden matrix is used. In contrast, TTLMs have varying intermediate TT-cores: more parameters and potentially higher expressivity in the model. More important, RNNs do not represent sequences in tensor space and theoretically lack a normalization factor for the joint probability distribution. **(2)** Regarding similarities, we argue that some RNNs could be seen as a special case of TTLMs. We will clarify TTLM's relations to certain RNNs in this section, including Second-order RNNs (Lee *et al.*, 1986; Giles *et al.*, 1989; Maupomé and Meurs, 2020), Recurrent Arithmetic Circuits (Levine *et al.*, 2018), and Multiplicative Integration RNNs (Wu *et al.*, 2016).

Let us consider a class of TTLMs that all intermediate TT-cores are equal to each other, denoted as $\mathbf{G} = \mathbf{G}^{(2)}, \ldots, \mathbf{G}^{(t-1)} \in \mathbb{R}^{R \times |V| \times R}$, following the convention of Khrulkov *et al.* (2018) and Miller *et al.* (2021). To simplify notation across different models, we use the following representations: $\mathbf{f}^{(t)}$ as the one-hot vector of the word $x^{(t)}$, $\mathbf{h}^{(t)} \in \mathbb{R}^R$ for the vector of hidden states, $\mathbf{A} \in \mathbb{R}^{R \times R}$ for the hidden-to-hidden matrix, $\mathbf{B} \in \mathbb{R}^{R \times |V|}$ for the input-to-hidden matrix, and $\phi(\cdot)$ as an element-wise nonlinear activation function.

Relation to Second-order RNN

Second-order RNNs (2-RNNs) use a third-order tensor, $\mathbf{T}$, to combine hidden states and input data in a *multiplicative* manner (Lee *et al.*, 1986; Giles *et al.*, 1989; Maupomé and Meurs, 2020). The i-th coordinate of the hidden states in 2-RNNs is defined as follows:

$$h_i^{(t)} = \phi\left((\mathbf{h}^{(t-1)})^\top \mathbf{T}_{:,i,:}(\mathbf{f}^{(t)}) + \mathbf{b}\right), \tag{5.11}$$

where $\mathbf{T}_{:,i,:} \in \mathbb{R}^{R \times |V|}$ is the ith slice of tensor $\mathbf{T} \in \mathbb{R}^{R \times R \times |V|}$, and $\mathbf{b}$ is a bias vector. For simplicity, we will ignore $\mathbf{b}$ for other variants of RNNs since $\mathbf{b}$ can be seen as 0th component of $\mathbf{f}(x^{(t)})$ that equals

to 1. Rabusseau *et al.* (2019) have shown that tensor trains can generalize linear 2-RNNs. We here illustrate this equivalent using the recursive property of TTLMs.

Claim 5.3.1. *The third-order tensor* $\mathbf{T}$ *in 2-RNNs equals the TT-cores in TTLMs. There is a nonlinear activation* ϕ *such that the hidden states of 2-RNNs are identical to that of TTLMs when they are accompanied by* ϕ.

Proof. The proof is based on the following observation: We recursively unfold the calculation of TTLMs in Eq. 5.2:

$$\begin{aligned}
\tilde{P}(\mathbf{x}) &= \psi\left(\sum_{i_1=1}^{|V|} f_{i_1}^{(1)} \mathbf{G}_{i_1\alpha_1}^{(1)} \cdots\right) \\
&= \psi\left(\sum_{i_1,i_2=1}^{|V|} \sum_{\alpha_1=1}^{R} f_{i_1}^{(1)} \mathbf{G}_{i_1\alpha_1}^{(1)} f_{i_2}^{(2)} \mathbf{G}_{\alpha_1 i_2 \alpha_2} \cdots\right) \\
&\quad\vdots \\
&= \psi\left(\sum_{i_1,\cdots,i_L=1}^{|V|} \sum_{\alpha_1,\cdots,\alpha_{L-1}=1}^{R} f_{i_1}^{(1)} \mathbf{G}_{i_1\alpha_1}^{(1)} f_{i_2}^{(2)} \mathbf{G}_{\alpha_1 i_2 \alpha_2} \cdots f_{i_L}^{(L)} \mathbf{G}_{\alpha_{L-1} i_L}^{(L)}\right),
\end{aligned} \tag{5.12}$$

Observe in the above, we find that $\mathbf{G}$ has two sources of "input" at each time step: the information from the previous recursive unfolding (e.g., in the second line, the first line is the previous information) and input data $\mathbf{f}^{(t)}$.

From this perspective, $\mathbf{G}$ acts as a bilinear map $\mathbf{G} : \mathbb{R}^{|V|} \times \mathbb{R}^R \to \mathbb{R}^R$. We can regard the information in the previous line as a hidden state, given by:

$$h_{\alpha_t}^{(t)} = \sum_{i_t=1}^{|V|} \sum_{\alpha_t=1}^{R} h_{\alpha_{t-1}}^{(t-1)} \mathbf{G}_{\alpha_{t-1}\alpha_t i_t} f_{i_t}^{(t)}, \tag{5.13}$$

where we permute the indices of $\mathbf{G}_{\alpha_{t-1} i_t \alpha_t}$ as $\mathbf{G}_{\alpha_t \alpha_{t-1} i_t}$.

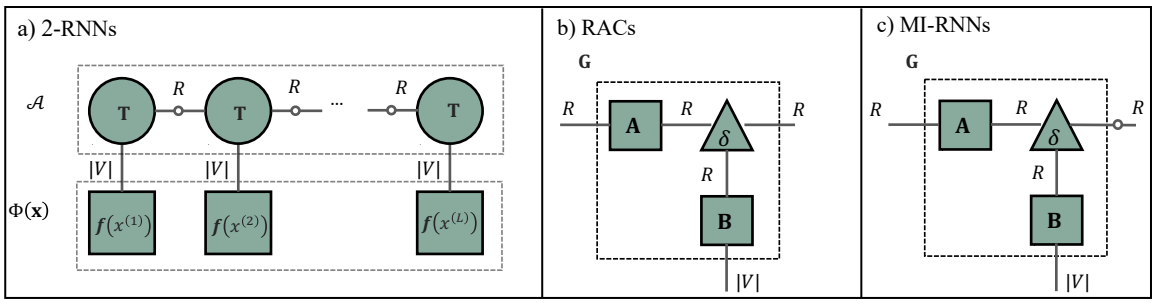

Figure 5.2 Hidden states of 2-RNNs, RACs, and MI-RNNs, draw from the perspective of TTLMs. The dashed line in the square denotes $\mathcal{A}, \Phi(\mathbf{x})$ or $\mathbf{G}$. The small hollow circles denote nonlinear activations.

After representing the hidden states in TTLMs, we can also represent the hidden states in 2-RNNs shown by Eq. 5.11 in an element-wise fashion:

$$\begin{aligned} h_i^{(t)} &= \phi\left((\mathbf{h}^{(t-1)})^\top \mathbf{T}_{:,i,:} \mathbf{f}^{(t)}\right) \\ &= \phi\left(\sum_{j=1}^{|V|}\sum_{k=1}^{R} h_j^{(t-1)} \mathbf{T}_{jik} f_k^{(t)}\right), \end{aligned} \tag{5.14}$$

where j,k are the dummy indices as i_t, α_t. Thus, $\mathbf{T} \in \mathbb{R}^{R\times R\times |V|}$ and $\mathbf{G} \in \mathbb{R}^{R\times |V|\times R}$ are two same-sized bi-linear maps.

After demonstrating that the third-order tensor (i.e., $\mathbf{T}$) in 2-RNNs equals the TT-cores (i.e., $\mathbf{G}$), the only difference between the hidden states in Eq. 5.14 and Eq. 5.13 is the nonlinear activation ϕ. If we add ϕ for $\mathbf{h}^{(t)}$ in TTLMs, the hidden states of 2-RNNs and TTLMs are identical, as shown in Fig. 5.2a.

□

Relation to RACs and MI-RNNs

Multiplicative Integration (MI) connects two sources of inputs by the Hadamard product '$\odot$'. MI has been used in RACs, Multiplicative RNNs (Sutskever *et al.*, 2011) and MI-RNN (Wu *et al.*, 2016):

Recurrent Arithmetic Circuits (RACs; Levine *et al.*, 2018)**.** The hidden states of RACs are defined as:

$$\mathbf{h}^{(t)} = \mathbf{A}\mathbf{h}^{(t-1)} \odot \mathbf{B}\mathbf{f}^{(t)}, \tag{5.15}$$

where these hidden states are also used as an *intermediate term* in M-RNNs.

Multiplicative Integration RNNs (MI-RNNs; Wu *et al.*, 2016)**.** The hidden states of MI-RNNs are defined as:

$$\mathbf{h}^{(t)} = \phi(\mathbf{A}\mathbf{h}^{(t-1)} \odot \mathbf{B}\mathbf{f}^{(t)}). \tag{5.16}$$

Claim 5.3.2. *Given the condition the TT-scores:* $\mathbf{G} = \mathbf{A} \odot \mathbf{B}$*. The hidden states of RACs are identical to that of TTLMs. There is a nonlinear function* ϕ *such that the hidden states of MI-RNNs are identical to that of TTLMs if they are accompanied by* ϕ*.*

Proof. The proof is based on the following observation: In the language of tensor contractions, Eq. 5.15 involves contracting the input weights matrix $\mathbf{B}$ with the input vector $\mathbf{f}(x^{(t)})$, and contracting the hidden weights matrix $\mathbf{A}$ with $\mathbf{h}^{(t-1)}$. The Hadamard product of the two is a third-order diagonal tensor $\delta \in \mathbb{R}^{R\times R\times R}$ such that $\delta_{ijk} = 1$ iff the $i = j = k$, and $\delta_{ijk} = 0$ otherwise. Thus, we can write Eq. 5.15 in element-wise fashion:

$$\begin{aligned} h_{\alpha_t}^{(t)} &= \sum_{i_t=1}^{|V|} \sum_{\alpha_t=1}^{R} \mathbf{A}_{j\alpha_{t-1}} h_{\alpha_{t-1}}^{(t-1)} \delta_{j\alpha_t k} \mathbf{B}_{ki_t} f_{i_t}^{(t)} \\ &= \sum_{i_t=1}^{|V|} \sum_{\alpha_t=1}^{R} h_{\alpha_{t-1}}^{(t-1)} \mathbf{G}_{\alpha_{t-1}\alpha_t i_t} f_{i_t}^{(t)}, \end{aligned} \tag{5.17}$$

where $\mathbf{G} = \mathbf{A} \odot \mathbf{B}$. In this case, the hidden state of TTLMs in Eq. 5.13 equals that of RACs in Eq. 5.17, as shown in Fig. 5.2b. Similarly, if Eq. 5.13 is accompanied by a nonlinear activation ϕ, Eq. 5.13 is equal to the hidden state of MI-RNNs in Eq. 5.16, as shown in Fig. 5.2c. □

Given Claim 5.3.1 and 5.3.2, the three models should be simulated by TTLMs with a nonlinear activation. We leave theoretical proof of this simulation to future work.

5.4 TTLMs Variant: Linear and Multiplicative Models

Up to this point, we have presented the overall architecture of TTLMs and discussed their recursive property to compute conditional probabilities. Empirically, we have assessed the impact of fundamental properties of tensor networks—linearity and multiplicativity—on model effectiveness, in Sec. 6.3. In essence, the gradients of hidden states easily vanish or explode because of the exponentiation of $\mathbf{A}^\top \mathrm{diag}\left(\mathbf{B}\mathbf{f}^{(t)}\right)$ (see Eq. 6.8). More important, hidden states of TTLMs exhibit exponential decay at each training step. For instance, with a sequence length of 784, values extend beyond data presentations at early recurrent steps. These dual issues substantially limit model effectiveness.

Thus, we attempt to devise an effective class of TTLMs that could address the aforementioned issues. We focus on the simplest TTLM variant, known as RACs (Levine *et al.*, 2018), with TT-cores $\mathbf{G} = \mathbf{A} \odot \mathbf{B}$ (see Claim. 5.3.2). Let $[\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(L)}]$ be a sequence of N-dimensional inputs. A RAC layer with H-dimensional hidden states is defined as:

$$\mathbf{h}^{(t)} = \mathbf{A}\mathbf{h}^{(t-1)} \odot \mathbf{B}\mathbf{x}^{(t)}, \tag{5.18}$$

where $\mathbf{h}^{(t)} \in \mathbb{R}^H$ is the vector of hidden states, $\mathbf{A} \in \mathbb{R}^{H \times H}$ is the hidden-to-hidden matrix, and $\mathbf{B} \in \mathbb{R}^{N \times H}$ is the input-to-hidden matrix.

We categorize our efforts to enhance model effectiveness into the following two sections. **(1)** When dealing with a new batch of sequences, the model initializes parameters according to its parameterization and initialization schemes at the outset. Therefore, in order to aid the model in learning the initial part of the sequences, we explore the *parameterization and initialization* scheme in Sec. 5.4.1. **(2)** As sequence length increases, the impact of parameterization and initialization diminishes in handling the latter part of sequences. Hence, we need a technique capable of maintaining stability and improving the computation of Eq. 5.18 throughout the entire process. We term techniques for this stage as the *normalization*, and discuss them in Sec. 5.4.2.

5.4.1 Parameterization and Initialization

Our initial step involves diagonalizing $\mathbf{A}$, which offers two key advantages: First, compared with a matrix with most non-zero elements (i.e., an *dense matrix*), eigenvalues of $\mathbf{A}$ can be easier managed to address challenges arising from exponentiations. Second, taking powers of diagonal matrices is faster than exponentiating dense matrices. We leverage a spectral analysis: up to an arbitrarily small perturbation of the entries, every matrix $\mathbf{A}$ is diagonalizable (Axler, 1997). Thus, one can write $\mathbf{A} = \mathbf{P}\mathbf{\Lambda}\mathbf{P}^{-1}$, where $\mathbf{P} \in \mathbb{C}^{H\times H}$ is an invertible matrix and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_H) \in \mathbb{C}^{H\times H}$. Plugging the decomposition $\mathbf{A} = \mathbf{P}\mathbf{\Lambda}\mathbf{P}^{-1}$ into Eq. 5.18 and multiplying both sides by $\mathbf{P}^{-1}$,

$$\bar{\mathbf{h}}^{(t)} = \mathbf{\Lambda}\bar{\mathbf{h}}^{(t-1)} \odot \mathbf{B}\mathbf{x}^{(t)}, \tag{5.19}$$

where $\bar{\mathbf{h}}^{(t)} = \mathbf{P}^{-1}\mathbf{h}^{(t)} \in \mathbb{C}^H$. In the following equations, we rename $\bar{\mathbf{h}}^{(t)}$ back to $\mathbf{h}^{(t)}$ for convenience. It is important to note that, unlike the symmetric setting where eigenvalues and eigenvectors are real, one has to allow for complex entries to achieve full equivalence in the non-symmetric case. Furthermore, in order to circumvent matrix multiplication and easier control the magnitude of the $\mathbf{h}$ over time, we diagonalize $\mathbf{B} = \mathbf{V}\mathbf{\Gamma}\mathbf{V}^{-1}$ and replace $\mathbf{B}$ with its diagonal matrix $\mathbf{\Gamma} = \text{diag}(\gamma_1, \gamma_2, \cdots, \gamma_H) \in \mathbb{C}^{H\times H}$:

$$\mathbf{h}^{(t)} = \mathbf{\Lambda}\mathbf{h}^{(t-1)} \odot \mathbf{\Gamma}\mathbf{x}^{(t)}, \tag{5.20}$$

where we assume $H = N$ for simplicity (i.e., $\mathbf{B} \in \mathbb{R}^{H\times H}$ and $\mathbf{x}^{(t)} \in \mathbb{C}^H$). Future work could explore *singular value decomposition* (Klema and Laub, 1980) and non-square diagonal matrices $\mathbf{\Gamma}$ to enhance model effectiveness.

Our second step focuses on Glorot initialization (Glorot and Bengio, 2010). The issue of vanishing and exploding gradients pertains to the initialization of neural networks (Vorontsov *et al.*, 2017). According to Thm. 4.2.1 introduced by Ginibre (1965), when each entry of a non-symmetric matrix is independently sampled from a Gaussian distribution with mean 0 and variance $1/N$, its eigenvalues are distributed across the complex unit plane. Thus, if we parameterize the entries of $\mathbf{\Lambda}$ and $\mathbf{\Gamma}$ in the complex unit plane, this indicates that the learnable parameters in Eq. 5.20 essentially are the

eigenvalues of both $\mathbf{A}$ and $\mathbf{B}$ under Glorot initialization. To achieve this goal, let us first parameterize $\mathbf{\Lambda}$ and $\mathbf{\Gamma}$ as follows:

$$\begin{aligned}\mathbf{\Lambda} &= \operatorname{diag}(\exp(-\boldsymbol{\nu}_\Lambda + i\boldsymbol{\theta}_\Lambda)) \\ \mathbf{\Gamma} &= \operatorname{diag}(\exp(-\boldsymbol{\nu}_\Gamma + i\boldsymbol{\theta}_\Gamma)),\end{aligned} \tag{5.21}$$

where $\boldsymbol{\nu}_\Lambda, \boldsymbol{\theta}_\Lambda, \boldsymbol{\nu}_\Gamma, \boldsymbol{\theta}_\Gamma \in \mathbb{C}^H$ are the learnable parameters. Then, to ensure that the entries of $\mathbf{\Lambda}$ and $\mathbf{\Gamma}$ are sampled uniformly on the unit disk, we use Lem. 4.2.1 introduced by Orvieto *et al.* (2023). Specifically, we initialize each tuple $(\boldsymbol{\nu}, \boldsymbol{\theta})$ as follows (omitted subscripts for clarity): Let $\mathbf{u}_1, \mathbf{u}_2$ be independent uniform random variables on the interval $[0, 1]$. Let $0 \leq r_{\min} \leq r_{\max} \leq 1$. Then, compute $\boldsymbol{\nu} = -\frac{1}{2}\log\left(\mathbf{u}_1(r_{\max}^2 - r_{\min}^2) + r_{\min}^2\right)$ and $\boldsymbol{\theta} = 2\pi\mathbf{u}_2$. In this way, $\exp(-\boldsymbol{\nu} + i\boldsymbol{\theta})$ is uniformly distributed on the annulus in $\mathbb{C}$ between circles of radii $r_{\min}$ and $r_{\max}$ (see Fig. 4.8).

Therefore, instead of learning $\mathbf{A}$ and $\mathbf{B}$, LMMs learn $\boldsymbol{\nu}_\Lambda, \boldsymbol{\theta}_\Lambda, \boldsymbol{\nu}_\Gamma, \boldsymbol{\theta}_\Gamma$ during training . The number of training parameters decreases from two dense matrices (i.e., $2H^2$) to four vectors (i.e., $4H$). More important, this parameterization removes dense matrix multiplications in Eq. 5.18 (cf. Eq. 5.20), enabling us to control the exponential growth of hidden states more easily. We term the model defined by Eq. 5.20 and Eq. 5.21 as the **Linear Multiplicative Models** (LMMs). We shall explore possible normalization techniques for LMMs in the following section.

5.4.2 Normalization

The component $j \in [N]$ of $\mathbf{h}^{(t)}$ at timestamp $t \in [L]$ according to Eq. 5.20 evolves such that:

$$h_j^{(t)} = (\lambda_j \gamma_j)^t \prod_{i=1}^{t} x_j^{(i)}, \tag{5.22}$$

where we assume that $\mathbf{h}^{(0)} = \mathbf{1}$. Because of Glorot initialization and the norm of λ_j, γ_j and $x_j^{(i)}$ being less than 1, Eq.5.22 clearly shows the exponential growth of hidden states in LMMs. To counteract this exponential behavior, we introduce three strategies to normalize Eq. 5.20 in the subsequent sections.

These strategies each bring distinct benefits but also inherent limitations, which are assessed and further explained in Sec. 6.4.2.

LMM-MLP. One approach involves adding trainable parameters to learn to counteract the exponential decay of hidden states in Eq. 5.20 during recurrent steps. To achieve this goal, we introduce a multi-layer perceptron to Eq. 5.20 and call it the LMM-MLP, with its hidden states at time t defined as:

$$\mathbf{h}^{(t)} = \mathrm{MLP}\left(\mathbf{\Lambda}\mathbf{h}^{(t-1)} \odot \mathbf{\Gamma}\mathbf{x}^{(t)}\right), \tag{5.23}$$

where $\mathrm{MLP}(\mathbf{z}) = \mathbf{C}\mathbf{z} + \mathbf{D}\mathbf{x}^{(t)}$ and $\mathbf{C} \in \mathbb{C}^{H\times H}, \mathbf{D} \in \mathbb{C}^{H\times H}$ are learnable parameters. From the perspective of tensor networks, a notable drawback of LMM-MLP is its disruption of multiplicativity. If we intend to leverage a linear and multiplicative model for theoretical analysis, it might be necessary to avoid employing LMM-MLP.

LMM-MMn. Another straightforward approach involves applying an activation to Eq. 5.20 that can rescale the hidden states within a pre-defined boundary during each recurrent step, as well as maintaining the inherent relationships within the original data. A paradox, however, arises due to the inherent linearity of tensor networks, while the commonly used activation functions, such as ReLU (Agarap, 2018), are nonlinear. Here, to preserve the linearity, we opt for the widely adopted linear transformation method known as Min-Max normalization (MMn; Sahu, 2015). Within MMn, we term the model as LMM-MMn, with its hidden states at time t defined as:

$$\mathbf{h}^{(t)} = \mathrm{MMn}\left(\mathbf{\Lambda}\mathbf{h}^{(t-1)} \odot \mathbf{\Gamma}\mathbf{x}^{(t)}\right), \tag{5.24}$$

where $\mathrm{MMn}(\mathbf{z}) = a + \frac{(\mathbf{z}-\min(\mathbf{z}))(b-a)}{\max(\mathbf{z})-\min(\mathbf{z})} \in \mathbb{R}^{H}_{[a,b]}$ and a, b are the min-max boundary values. For example, by setting $a = -1$ and $b = 1$, the components of $\mathbf{h}^{(t)}$ will be rescaled to $[-1, 1]$ during each recurrent step.

In Sec. 6.4, we will evaluate the effectiveness of LMMs, LMM-MLP, and LMM-MMn.

6 Experimental Evaluation

This chapter is organized as follows: Sec. 6.1 introduces our experimental objectives. Second, Sec. 6.2 outlines the experimental setup. Third, Sec. 6.3 investigates the impact of two fundamental properties of tensor networks—linearity and multiplicativity—on model effectiveness. Last, Sec. 6.4 evaluates the impact of introducing complex numbers and the effectiveness of our proposed TTLM variants.

6.1 Objectives of the Experiments

One of our research objectives is to narrow the performance gap between tensor network LMs and neural network LMs in long-range scenarios (see Sec. 5.1). In Ch. 4, we identify this gap mainly stemming from a training paradigm mismatch. To address this, we propose a class of tensor-train LMs (TTLMs) in Ch. 5. TTLMs represent sequences in tensor space, encode joint probabilities as wavefunctions, and employ tensor trains for learning conditional probabilities. It inherits two fundamental properties—*linearity* and *multiplicativity*—from tensor networks.

While these two properties offer theoretical advantages, they also present training challenges. On the theoretical side, prior work used tensor networks to analyze neural network models, including parameters compression (Novikov *et al.*, 2015), expressive power (Cohen *et al.*, 2016; Khrulkov *et al.*, 2018), and depth efficiency for long-term memory (Levine *et al.*, 2018). However, on the pragmatic side, the effectiveness of linear models is limited; for instance, nonlinearities in RNNs are believed to be crucial for their success (Siegelmann, 2012; Pascanu *et al.*, 2013; Erichson *et al.*, 2020). Moreover, most recurrent models use an additive operator, not a multiplicative one. Despite Wu *et al.* (2016) arguing the benefits of multiplicative integration, they incorporated additive operations in multiplicative integration (see Sec. 6.3.2).

Thus, our core objective in this chapter is to assess the impact of linearity and multiplicativity on model effectiveness. Accompanied by experiments and statistical methods, we delve into challenges stemming from linearity, pinpoint environments conducive to training linear recurrent models, and dissect the contributions of individual model components to the training process. Furthermore, we uncover distinct training challenges posed by multiplicativity, highlighting differences from those posed by linearity. This objective yields two benefits. **(1)** Prior work, exemplified by Khrulkov *et al.* (2018), argued a link between RNNs and Tensor-Train decompositions, using tensor trains to prove the expressivity for RNNs. Nonetheless, their theories lacked comprehensive empirical validation. Our experiments will uncover the specific effects arising from linearity and multiplicativity during training, shedding light on aspects that may have been overlooked. **(2)** These insights will provide a deeper understanding of the challenges of applying tensor networks in long-range scenarios. This will offer valuable guidance to researchers developing performant tensor network LMs.

Following analyzing these properties, we take a further step toward enhancing the effectiveness of tensor network LMs. As we have devised TTLMs in Sec. 5.4, we shall assess their effectiveness in Sec. 6.4.

6.2 Experimental Setup

This section provides a detailed description of our experimental setup. We present the tasks, datasets, and metrics in Sec. 6.2.1. We briefly introduce the baselines used in each task in Sec. 6.2.2. Sec. 6.2.3 outlines the model architecture used in the experiments. Last, Sec. 6.2.4 lists the model hyperparameters.

6.2.1 Tasks, Datasets, and Metrics

To examine model effectiveness in long-range scenarios, we focus on two specific challenges: (i) classification problems designed to test sequence models and (ii) language modeling datasets with

Table 6.1 Table of datasets used in this thesis. L denotes the maximum length of the samples in each task. $|T|$ denotes the number of training examples.

Task	Dataset	$\|V\|$	$\|T\|$	L	Epochs
Classification	sMNIST	256	54k	784	100
	sCIFAR	256	45k	1024	200
	LISTOPS	18	96k	2,048	100
	TEXT	135	25k	4,096	100
	RETRIEVAL	98	147k	4k	100
	PATHFINDER	2	480k	1,024	100
	PATH-X	2	480k	16,384	100
Language modeling	WikiText-2	50264	48k	1024	200

increasing segment length. Their relevance and importance for our experiments will be clarified in subsequent sections. Table 6.1 shows an overview of the tasks and datasets used in the chapter.

Classification

Pixel-level image classification. In this task, the model learns to classify images by processing the image pixels sequentially; the pixels are read one at a time in scanline order, starting from the top left corner and ending at the bottom right corner. The model predicts the image category only after observing all pixels. We evaluate models on the MNIST (LeCun *et al.*, 1998) and CIFAR-10 datasets (Krizhevsky, Hinton, *et al.*, 2009), containing images with 784 and 1024 pixels, respectively.

Both datasets have 10 image categories. Prior research used the two datasets to test whether models can capture long-term temporal dependencies, as the model has long time steps (i.e., 784 or 1024). For instance, the sequential MNIST (sMNIST) has been used in Le *et al.* (2015) and Arjovsky *et al.* (2016), and the sequential CIFAR (sCIFAR) has been employed in Gu, Goel, and Ré (2022) and Orvieto *et al.* (2023). We use the two datasets to analyze the linearity and multiplicativity of models in Sec. 6.3.

Long Range Arena (LRA; Tay *et al.*, 2020). LRA consists of 6 challenging classification tasks with varying lengths of 1K-16K steps. These tasks involve diverse modalities and objectives, demanding similarity, structural, and visuospatial reasoning. This benchmark is more difficult than the pixel-level image classification task. For instance, in the Path-X task, with images containing 16K pixels, many

models do not perform better than random guessing (Tay *et al.*, 2020). Thus, LRA tasks serve as a popular option to assess the long-range modeling capabilities of SOTA models (Gu, Goel, and Ré, 2022; Dao *et al.*, 2022; Orvieto *et al.*, 2023). We will not use the `image` task in LRA, as it is the gray-scaled (single channel) version of sCIFAR. We shall compare our proposed models with SOTA models on the remaining five tasks in Sec. 6.4, provided that our models demonstrate competitive performance on the `sMNIST` and `sCIFAR` datasets.

Language modeling

Prior research, such as Orvieto *et al.* (2023), only evaluated models on classification tasks. However, to gain a comprehensive understanding of model capabilities, it is crucial to gauge its ability to predict subsequent natural language words in long-range scenarios. Thus, as part of our evaluation, we test models on two widely-used language datasets, utilizing perplexity (PPL) (Meister and Cotterell, 2021) as the evaluation metric. A lower PPL indicates superior model performance; we refer to Sec. 2.4 for an introduction to PPL.

Our experiment underscores an aspect often overlooked in prior research: we adapt dataset segment lengths to better evaluate the long-range capacity of models. To comprehend the rationale behind this setup, readers should grasp two key facets: **(1)** Text data preprocessing involves segmenting text into fixed-length sequences comprising L tokens. This value, L, is usually treated as a hyperparameter, typically ranging from 30 to 512 tokens. **(2)** During language modeling training, recurrent-based models predict and store predicted information about subsequent tokens. After the model "reads" the entire sequence, perplexity is computed by averaging the whole prediction over the sequence. For instance, with a segment length of 1024 tokens, the model is tasked with predicting the 1024th word based on the preceding 1023 words. Consequently, as segment length and recurrent steps increase, the model will face heightened difficulty predicting sentence endings.

We conduct experiments on the word-level language model dataset: WikiText-2 (WT2; Merity *et al.*, 2016). WT2 is a dataset derived from Wikipedia articles, comprising 2088k training tokens, 217k validation tokens, and 45k test tokens, with a vocabulary of over 30k types. We use byte pair

encoding (Sennrich *et al.*, 2016) to increase the vocabulary size to 50,264, following the convention in Melis *et al.* (2019) and Wang *et al.* (2019). This setup can help models learn the patterns of low-frequency tokens (Melis *et al.*, 2019).

6.2.2 Baselines

We categorize our baselines into **(1)** linear and multiplicative analysis and **(1)** SOTA models. In the upcoming sections, we elaborate on our selection criteria and their relevance to our experimental objectives.

Linearity and Multiplicativity Analysis

The baselines in this group are used to investigate how the fundamental properties of TTLMs—linearity and multiplicativity—affect model effectiveness.

Linearity analysis. Orvieto *et al.* (2023) introduced a novel perspective that highlights the capacity of linear RNNs to outperform their nonlinear counterparts (see Sec. 4.2.2), contradicting prior research where nonlinearity is commonly considered pivotal for RNNs success (Siegelmann, 2012; Pascanu *et al.*, 2013; Erichson *et al.*, 2020). In order to study the impact of linearity on model effectiveness, we compare the vanilla version of RNNs (vRNNs; Mikolov *et al.*, 2010) with nonlinear activations and with identity activations. The hidden states of vRNNs are defined as:

$$\mathbf{h}^{(t)} = \phi\left(\mathbf{A}\mathbf{h}^{(t-1)} + \mathbf{B}\mathbf{f}^{(t)}\right), \tag{6.1}$$

where $\mathbf{f}^{(t)} \in \mathbb{R}^{|V|}$ denotes the one-hot vector of the word $x^{(t)}$, $\mathbf{h}^{(t)} \in \mathbb{R}^{R}$ is hidden states, $\mathbf{A} \in \mathbb{R}^{R\times R}$ is the hidden-to-hidden matrix, $\mathbf{B} \in \mathbb{R}^{R\times|V|}$ is the input-to-hidden matrix, and $\phi(\cdot)$ is an element-wise activation function.

Multiplicativity analysis. In this group, we consider Multiplicative Integration RNNs (MI-RNNs; Wu *et al.*, 2016) and Recurrent Arithmetic Circuits (RACs; Levine *et al.*, 2018), since they can be viewed

as special cases of TTLMs. Our Claim 5.3.2 states that *under the condition of TT-scores:* $\mathbf{G} = \mathbf{A} \odot \mathbf{B}$, *the hidden states of RACs are identical to that of TTLMs. There is a nonlinear function* ϕ *such that the hidden states of MI-RNNs are identical to that of TTLMs if they are accompanied by* ϕ. Therefore, MI-RNNs and RACs can help us comprehend the impact of tensor network multiplicativity on model effectiveness. The hidden states of RACs and MI-RNNs are computed as follows:

$$\mathbf{h}^{(t)} = \phi\left(\mathbf{A}\mathbf{h}^{(t-1)} \odot \mathbf{B}\mathbf{f}^{(t)}\right), \tag{6.2}$$

where $\phi(\cdot)$ denotes nonlinear activations in MI-RNNs and identity activations in RACs.

SOTA models

We shall assess LMMs (see Sec. 5.4) against SOTA models using the same training setting in this stage. Our chosen baselines span three domains: transformer-based, state-space, and recurrent-based models.

Transformer (Vaswani *et al.*, 2017)**.** The vanilla Transformer has substantially advanced the effectiveness of tasks in natural language processing. With its attention mechanism, the Transformer captures dependencies and exhibits parallelism during training. Its effectiveness in various domains underscores its position as an important benchmark in our evaluation.

Structured State Space Sequence model (S4; Gu, Goel, and Ré, 2022)**.** State-space models (SSMs) have achieved SOTA performance on LRA tasks. These models are inspired by continuous-time linear SSMs, a well-established component of modern control systems (Gu, Goel, and Ré, 2022). Representative deep SMMs include S4 and its variants (DSS (Gu, Goel, Gupta, *et al.*, 2022), S4D (Gupta *et al.*, 2022), S5 (Smith *et al.*, 2023)).

Linear Recurrent Units (LRUs; Orvieto *et al.*, 2023)**.** SSMs and Vanilla-RNNs are superficially similar, as discussed in Sec. 4.2.2. Motivated by these similarities and differences, Orvieto *et al.* (2023) argued that it is unclear where the performance enhancement of deep SSMs boost over Vanilla-RNNs comes from. They show that the careful design of deep RNNs using standard signal propagation arguments

can recover the impressive performance of deep SSMs on long-range reasoning tasks, while also matching their training speed.

6.2.3 Architecture

We consider the standard S4 architecture (Gu, Goel, and Ré, 2022) and replace the S4 layers with recurrent units listed in Sec. 6.2.2. The term *recurrent unit* refers to using a recurrent-based model, such as Vanilla-RNN, as a single layer within a larger model architecture. During training, the recurrent unit plays a role similar to the attention layer in Transformer (Vaswani *et al.*, 2017).

We examine the impact of individual model components within the S4 architecture on recurrent units in Sec. 6.3. We adopt the S4 architecture for two main reasons. **(1)** A larger architecture is crucial for achieving higher model performance. Standalone recurrent layers struggle to compete in deep learning benchmarks, prompting current research to optimize the single recurrent layer when integrated into a larger architecture (Gu, Goel, and Ré, 2022; Orvieto *et al.*, 2023; Smith *et al.*, 2023; Gupta *et al.*, 2022; Gu, Goel, Gupta, *et al.*, 2022). Thus, we believe that demonstrating the usefulness of a novel recurrent unit within a larger architecture is essential. **(2)** The S4 architecture proves beneficial for recurrent linearities. Orvieto *et al.* (2023) showcased that using linear recurrent units (removing nonlinear activation in the vanilla RNN) led to substantial improvements in model performance. Orvieto *et al.* (2023), however, used the S4 architecture mainly to compare their model strictly with S4, potentially overlooking the distinct role each model component may play in supporting recurrent linearities. As TNLMs inherently have linearity, we analyze how this larger architecture aligns with our approach in Sec. 6.3.

The S4 architecture comprises three components: encoder, decoder, and residual block. The input data undergo encoding, resulting in N_{in} features, which are then processed by a stack of residual blocks, projecting them to N_{out} features. Subsequently, the decoder maps the N_{out} features to the desired number of predicted classes. We set $N_{\text{in}} = N_{\text{out}}$ for simplicity in all tasks, which will be denoted as *model size* (N) in subsequent references.

The encoder is a linear transformation that maps the input data from input size (denoted as M) to N. The decoder varies depending on the task: for the classification task, the decoder is a linear transformation that maps the data from size N to the number of output classes. For the language model tasks, the decoder is the softmax function in order to compute perplexity.

Each residual block consists of identity skip connection, and the residual path containing the following ordered components: **(1)** a recurrent unit, **(2)** post layer/batch normalization (Ba *et al.*, 2016; Ioffe and Szegedy, 2015), **(3)** the Gaussian Error Linear Unit (GELU) activation (Hendrycks and Gimpel, 2016) with dropout, and **(4)** position-wise linear transformation with the Gated Linear Unit activation (Dauphin *et al.*, 2017), as well as dropout.

Regarding dropout, we discuss the impact of two implementations of dropout in Sec. 6.3. The implementation of dropout used in S4 (Gu, Goel, and Ré, 2022) and LRUs (Orvieto *et al.*, 2023) differs. In Gu, Goel, and Ré (2022), when `tie_dropout` is set to true, the layer randomly zeros out an entire dimension of hidden states across the segment length. On the other hand, when `tie_dropout` is set to false, it randomly zeros some elements of the input data with probability p. Thus, the `tie_dropout=True` setting provides a stronger regularization, commonly used in sequence models.

6.2.4 Hyperparameters

We list the hyperparameters used throughout the experiments in Table 6.2. Hyperparameters are chosen through grid search, considering the best performance on the validation set.

For our implemented models, we use AdamW optimizer (Loshchilov and Hutter, 2017). We use warmup for the learning rate, starting from a value of 10^{-7} and increasing the learning rate linearly up a specified value for the first 10% of training epochs. This is followed by cosine annealing for the rest of the training down to 10^{-7}. The specified value is called the *base rate* in Table 6.2, which is tuned on a logarithmic grid of 10 within the range from 3×10^{-5} to 3×10^{-2} choose the optimal value.

Table 6.2 Table of hyperparameters. The italicized terms indicate that they are tunable, but other terms vary according to the different task settings (see Sec. 6.2.4 for a detailed explanation).

	Component	Hyperparameter	Value
Encoder / Decoder	Linear layer	Input size (M) *Model size* (N) Output size	{1, 3} {128, 256, 512} [2, 10]
Residual Block	Recurrent unit	*Hidden size* (H) Dropout Activation	{192, 256, 384} 0.0 {None, Tanh}
	Position-wise FFN	*Dropout* Activation	{0.1, 0.25} GLU
Training	Learning rate	*Base rate* LR factor Schedule Warm-up	$[3 \times 10^{-5} - 3 \times 10^{-2}]$ 0.25 Cosine Annealing Num of epochs * 0.1
		Num of epochs Batch size Normalization Optimizer Weight Decay	{50, 100, 200} 50 Batch/Layer AdamW {0, 0.05, 0.1}

We employ a reduced learning rate and no weight decay for the parameters in recurrent units (i.e., **A** and **B**). Given the recursive computations in long-range scenarios, the learning rate for **A** and **B** is generally below that of other model parameters (Gu, Goel, and Ré, 2022; Orvieto *et al.*, 2023). This lower learning rate was determined by multiplying the base learning rate by a factor (< 1).

6.3 Linearity and Multiplicativity Analysis

Recurrent nonlinearities are believed to be a key component for the success of RNNs — both in theory and in practice (Siegelmann, 2012; Pascanu *et al.*, 2013; Erichson *et al.*, 2020). For example, a strong property of single-layer sigmoidal and Tanh RNNs is Turing completeness, which cannot be achieved by the linear variant (Chung and Siegelmann, 2021). Surprisingly, the S4 architecture used by Orvieto *et al.* (2023) challenges this notion by demonstrating that linear recurrent units can outperform nonlinear ones, as discussed in Sec. 4.2.2. In other words, accompanied by nonlinear activations outside these units, the S4 architecture may allow us to employ linear recurrent units. Given

that tensor networks are inherently linear, we aim to comprehend the specific reasons behind their success, helping us to use TTLMs as recurrent units and potentially enhance model effectiveness.

Unfortunately, the S4 architecture's impact on recurrent linearities remains incompletely understood. Orvieto *et al.* (2023) compared the expressivity of recurrent linearities and nonlinearities, using spectral analysis and Koopman operator theory (Koopman and Neumann, 1932). They demonstrated that placing linear recurrent units alongside nonlinear feedforward blocks is sufficient to approximate highly nonlinear systems. Their studies, however, did not examine other model components, including residual connections, optimizers, dropout types, normalization, and nonlinear activations. These components could have distinct impacts on recurrent linearities. Furthermore, another fundamental property of tensor networks is *multiplicativity*. We need to examine whether the advantages of the S4 architecture apply to both additive linearities and multiplicative linearities.

In this section, we perform the following two tasks: **(1)** conduct an ablation study to evaluate the impact of the S4 architecture using the sMNIST and sCIFAR datasets, and **(2)** test model capabilities to predict the next word in long-range scenarios using the WT2 dataset. We aim to understand how each component in the S4 architecture empirically contributes to training linear recurrent units and whether this support remains consistent for both additive units and multiplicative units. To manage the heavy load of experiments, we avoid extensive hyperparameter fine-tuning for optimal performance in each task. Instead, we only tune learning rates for all models on a logarithmic grid of 10 to prevent training issues arising from inappropriate rates.

6.3.1 Additive Recurrent Units

This section focuses on the performance of additive recurrent units on the sMNIST, sCIFAR, and WT2 datasets. We compare two types of units: the vanilla version of RNNs using Tanh activations (vRNN-Tanh) as specified in Eq. 6.1, and the other with identity activations (vRNN-Lin). We explain our rationale for choosing the two models to study the impact of recurrent linearities on model effectiveness in Sec. 6.2.2.

Table 6.3 Test accuracy (in %) of one-layer additive recurrent linearities on sMNIST and sCIFAR. BN and LN refer to Batch Normalization and Layer Normalization. We refer to Sec. 6.2.3 for a detailed explanation of the terms in the column "Architecture." ✗ signifies numerical instability during training, causing loss values to become NaNs.

Model	Dataset		Architecture					
	sMNIST	sCIFAR	Residual	Optimizer	Tie_dropout	Norm	GLU	GELU
vRNN-Tanh	96.82	62.21	On	AdamW	Off	BN	On	On
	96.80	62.03	Off	AdamW	Off	BN	On	On
	91.63	22.28	On	SGD	Off	BN	On	On
	96.62	62.29	On	AdamW	On	BN	On	On
	96.43	60.53	On	AdamW	Off	LN	On	On
	96.91	62.19	On	AdamW	Off	BN	Off	On
	97.05	62.15	On	AdamW	Off	BN	On	Off
vRNN-Lin	97.16	**70.79**	On	AdamW	Off	BN	On	On
	97.09	31.05	Off	AdamW	Off	BN	On	On
	34.34	30.33	On	SGD	Off	BN	On	On
	96.94	24.91	On	AdamW	On	BN	On	On
	✗	✗	On	AdamW	Off	LN	On	On
	84.61	70.45	On	AdamW	Off	BN	Off	On
	97.47	24.69	On	AdamW	Off	BN	On	Off

Table 6.3 displays the test accuracy of vRNN-Tanh and vRNN-Lin on pixel-level image classification tasks. The column "Architecture" contains the five major components of the S4 architecture. Each row displays the accuracy after removing one specific component from the architecture. As depicted in Table 6.3, vRNN-Lin exhibits superior performance compared with vRNN-Tanh on sMNIST and sCIFAR, when the S4 architecture is kept intact. However, vRNN-Tanh is more robust than vRNN-Lin when subjected to changes in the S4 architecture. In the subsequent sections, we identify two factors contributing to linearity vulnerability: gradient vanishing/exploding and exponential growth in hidden states.

Gradients Exploding/Vanishing

The issue of vanishing or exploding gradients, as described by Bengio and Frasconi (1994) and Pascanu *et al.* (2013), is one barrier to training RNNs with gradient descent. Theoretically, in the vanilla version of RNNs with identity activations, the gradient of hidden states is computed as follows:

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-n)}} = (\mathbf{A}^{\top})^{n}, \tag{6.3}$$

where $\mathbf{h}^{(t)} = \mathbf{A}\mathbf{h}^{(t-1)} + \mathbf{B}\mathbf{f}^{(t)}$ and $n \in [L]$. The computation of the gradient of hidden-to-hidden matrices entails $\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-n)}}$, and as a result, the exponentiations of the matrix $\mathbf{A}$ within the aforementioned equation are the root cause of the unstable gradient flow.

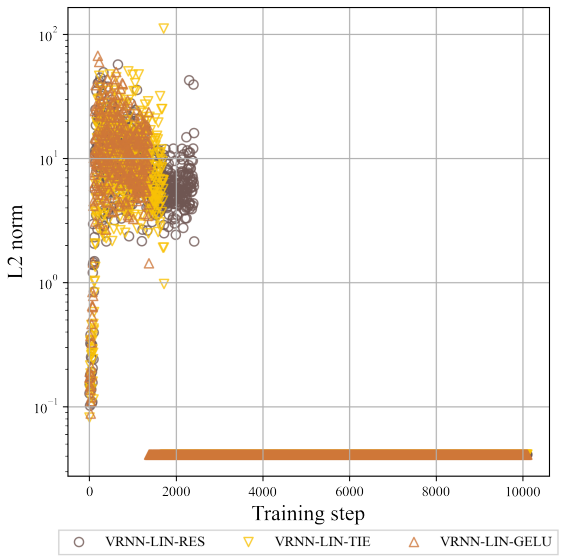

Figure 6.1 L_2 norms of gradients of $\mathbf{A}$ on sCIFAR. We add $\varepsilon = 1 \times 10^{-2}$ to display zero entries on logarithmic-scale axes. vRNN-LIN-RES refers to the model in the ninth row of Table 6.3; vRNN-LIN-TIE refers to the eleventh row; and vRNN-LIN-GELU refers to the final row.

The S4 architecture alleviates the vanishing or exploding gradients inherent in standalone Vanilla-RNNs. As listed in Table 6.3, removing residual connections, GELU activation, or tying dropout results in poor effectiveness on sCIFAR. Empirically, Fig. 6.1 illustrates the L_2 norm of the gradients of the hidden-to-hidden matrix $\mathbf{A}$, showcasing that these models experience initial gradient explosion followed by vanishing within the initial $3k$ training steps. These models struggle to train effectively owing to gradient flow issues.

Regarding gradient flow in recurrent linearities, our results offer three empirical insights for the model components in the S4 architecture: **(1)** they highlight the practical advantages of residual connections and GELU activation for linearities. Residual connections enable the model to learn residual functions, capturing differences between input data and recurrent units. GELU activation promotes consistent gradient flow, contrasting with ReLU (Agarap, 2018), which can render neurons inactive during training owing to zero outputs (Hendrycks and Gimpel, 2016). **(2)** Concerning dropout techniques, stronger regularization could potentially disrupt gradient flow during training. Therefore, we advise not to tie dropout across the sequence length when applying recurrent linearities. **(3)** Removing GLU activation (Dauphin *et al.*, 2017) did not cause gradient issues, but limited model effectiveness. The significance of GLU was discussed in Orvieto *et al.* (2023). They argued that a linear RNN, coupled with proper nonlinear reparameterization of inputs, can represent any regular nonlinear dynamical system. A key finding in their analysis was that position-wise nonlinearities effectively transferred signal information to higher frequencies, enabling the system to go beyond linearity in the

Table 6.4 Test perplexity of Vanilla RNN without residual connections on WikiText-2. D refers to the number of residual blocks. ✗ signifies numerical instability during training, causing loss value to become NaNs.

Model	Segment Length						
	$L=16$	$L=32$	$L=64$	$L=128$	$L=256$	$L=512$	$L=1024$
vRNN-Tanh ($D=1$)	65.84	58.03	60.50	**56.97**	58.40	60.76	60.69
vRNN-Tanh ($D=6$)	62.25	59.85	60.88	63.08	67.43	68.04	68.42
vRNN-Lin ($D=1$)	68.38	59.98	60.28	66.86	68.00	✗	✗
vRNN-Lin ($D=6$)	72.38	✗	✗	✗	✗	✗	✗

spectral domain and increasing the layer capacity. The empirical support from Table 6.3 reinforces their assertion that combining recurrent linearities and nonlinear GLU activations surprisingly outperforms recurrent nonlinearities.

While the S4 architecture can alleviate gradient exploding/vanishing for classification tasks, it fails to handle gradient flow issues in language modeling tasks. Table 6.4 shows that when the segment length in WT2 increases, one-layer vRNN-Lin fails to manage data with $L=512$, and the sixth-layer vRNN-Lin cannot even handle sentences with $L=32$. Note that we do not include residual connections here, as we sought to assess the impact of recurrent units fully; the models are still untrainable even if we use residual connections.

Exponential Growth in Recurrent Steps

Numerical instability in recurrent nonlinearities is typically classified as gradient exploding or vanishing (Pascanu *et al.*, 2013). Regarding recurrent linearities, we propose a new factor contributing to their numerical instability: *hidden states can easily exceed precision bounds as their values exponentially increase with recurrent steps.*

We establish our claim through three steps. Initially, we trace the occurrence of NaN values. Fig. 6.2 illustrates the average of the components of the last sequence hidden state $\mathbf{h}^{(L)}$ throughout the training steps. For clarity, the term *training step* refers to an iteration during training, with a total count equivalent to training examples (i.e., $|T|$) divided by batch size (i.e., B); the size of $\mathbf{h}^{(L)}$ is $B \times L$. Given $B=50$ and $|T|=54k$, each epoch comprises 1400 training steps. At the 808th step, vRNN-Lin-LN surpasses float16 precision bounds.

Subsequently, we delve into the cause of NaN values. Fig. 6.3 illustrates the evolution of the average $\mathbf{h}^{(t)}$ across recurrent steps. For each training step, the model undergoes 784 recurrent steps, corresponding to the sequence length of sMNIST (i.e., $L = 784$). Thus, the initial three peaks in Fig. 6.3 (left) correspond to the first three points in Fig. 6.2.

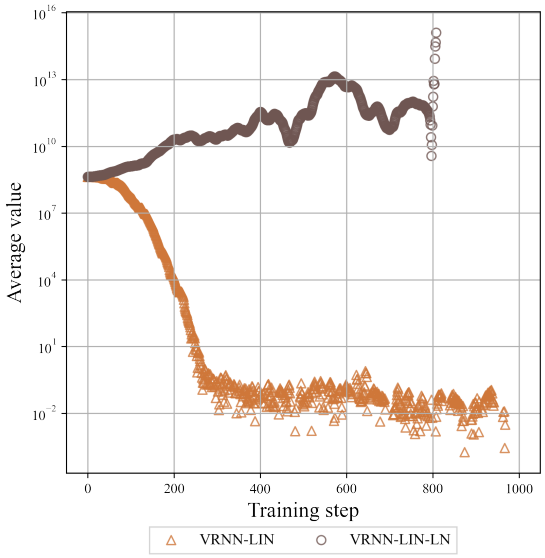

Figure 6.2 Average of $\mathbf{h}^{(L)}$ on sMNIST. VRNN-LIN corresponds to the model in the eighth row of Table 6.3, and VRNN-LIN-LN corresponds to the twelfth row. At the 808th training step, the values in VRNN-LIN-LN surpassed float16 precision bounds and became NaNs.

Third, we further categorize the pattern of this monotonic growth by using the least squares method (Marquardt, 1963) to fit an exponential function to the data:

$$y = ae^{-bx} \tag{6.4}$$

where y is the average of the componentes of $\mathbf{h}^{(t)}$ and $x \in [784]$ is the recurrent steps. The fitted result is $a = 0.051$ and $b = -0.033$ ($R^2 = 0.995$). Thus, to a good approximation, the average of $\mathbf{h}^{(t)}$ exhibits exponential growth within each training step. This growth should arise from the additive operation $\mathbf{A}\mathbf{h}^{(t-1)} + \mathbf{B}\mathbf{f}^{(t)}$, driving $\mathbf{h}^{(t)}$ beyond data precision. Note that the averages of the hidden states are not distributed independently and identically (i.i.d.), which could lead to invalid inferences when using the least squares method for curve fitting. We shall conduct additional statistical tests that do not rely on i.i.d. assumptions to ensure the validity of the inference in future work.

Unlike recurrent linearities, this exponential growth can be effectively curbed by recurrent nonlinearities, as shown in Fig. 6.3 (right). We assess three nonlinear activations: GELU, Tanh, and ReLU. The average of $\mathbf{h}^{(t)}$ remains stable across recurrent steps. We highlight two key observations regarding this constraint: **(1)** This limitation does not appear to stem from an upper bound of nonlinear activations, as ReLU and GELU successfully curtail the growth of $\mathbf{h}^{(t)}$ despite lacking an explicit upper bound. **(2)** This constraint maintains robustness even with increased learning rates (tested up to 3×10^{-2}) and modifications on model components.

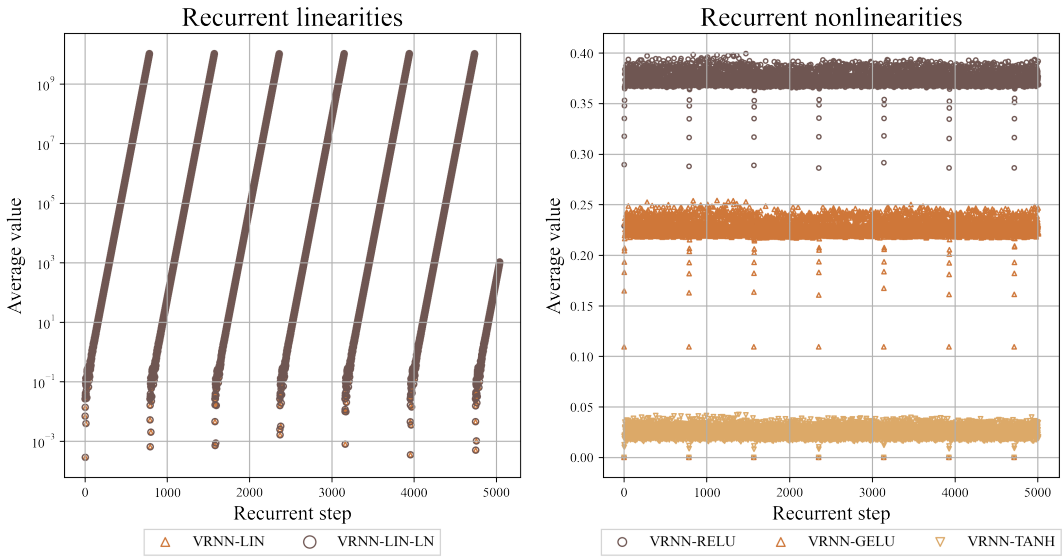

Figure 6.3 Additive hidden states within recurrent steps on sMNIST. (**Left**) Average of $\mathbf{h}^{(t)}$ for recurrent lienarities. vRNN-LIN corresponds to the model in the eighth row of Table 6.3, and vRNN-LIN-LN corresponds to the twelfth row. (**Right**) Average of $\mathbf{h}^{(t)}$ for recurrent nonlienarities. Models include vRNN-TANH (first row of Table 6.3), vRNN-GELU (replacing Tanh with GELU), and vRNN-RELU (replacing Tanh with ReLU).

The S4 architecture partially addresses the problem of exponential growth in hidden states for recurrent linearities. In essence, the S4 architecture preserves the increasing pattern of hidden states, but limits their peak values over training steps, which enhances numerical stability: Upon comparing vRNN-LIN-LN and vRNN-LIN, we observe that they both exhibit exponential growth in the average of $\mathbf{h}^{(t)}$ (see Fig. 6.3). As training progresses, however, the average of $\mathbf{h}^{(L)}$ in vRNN-LIN consistently decreases (see Fig. 6.2). In other words, the "peak" value of $\mathbf{h}^{(t)}$ in vRNN-LIN consistently decreases.

While vRNN-LIN-LN reaches an explosive value, vRNN-LIN does not. Given that we solely modify the normalization layer between vRNN-LIN and vRNN-LIN-LN, we speculate that batch normalization can effectively control the scaling of hidden states in recurrent linearities in long-range scenarios. It is worth noting that for recurrent linearities, even if we retain the intact S4 architecture (i.e., vRNN-LIN), the model encounters NaN values if the learning rate reaches 3×10^{-3}. The difference lies in the ability of recurrent nonlinearities to change the exponential growth, whereas the S4 architecture primarily constrains the peak value of $\mathbf{h}^{(t)}$ during recurrent steps.

6.3.2 Multiplicative Recurrent Units

In the preceding section, we analyzed the impact of the S4 architecture on additive recurrent linearities. Our observations indicated that the architecture partially mitigates two issues: exponential growth of hidden states and gradient vanishing/exploding. Altering the architecture, however, could easily lead to numerical instability for recurrent linearities, casting uncertainty on its wider applicability beyond additive models. Considering the intrinsic multiplicative characteristics of tensor networks, we proceed to investigate multiplicative recurrent linearities. Our objective is to contrast the impact of the S4 architecture on additive and multiplicative units.

We compare two types of units as specified in Eq. 6.2: MI-RNNs with nonlinear activations and the one with identity activations (i.e., RACs). Their selection is primarily because that they can be viewed as special cases of TTLMs. We refer to Sec. 6.2.2 for a detailed explanation. Before delving into specific experiments, we shall clarify the term *multiplicative integration* in this context. The hidden states of the two models are defined as follows:

$$\mathbf{h}^{(t)} = \phi\left(\mathbf{A}\mathbf{h}^{(t-1)} \odot \mathbf{B}\mathbf{f}^{(t)}\right), \tag{6.5}$$

where $\odot$ denotes the Hadamard product and ϕ is an activation function. This basic form of multiplicative integration does not involve any additive operator. While Wu and King (2016) argued for the advantages of multiplicative integration over additive integration, they introduced several modifications to this basic form. Specifically, they first introduced two additional bias vectors, β_1 and β_2, added to $\mathbf{A}\mathbf{h}^{(t-1)}$ and $\mathbf{B}\mathbf{f}^{(t)}$:

$$\mathbf{h}^{(t)} = \phi\left((\mathbf{A}\mathbf{h}^{(t-1)} + \beta_1) \odot (\mathbf{B}\mathbf{f}^{(t)} + \beta_2)\right). \tag{6.6}$$

Then, they introduced another bias vector α to gate the term $\mathbf{A}\mathbf{h}^{(t-1)} \odot \mathbf{B}\mathbf{f}^{(t)}$, obtaining the following formulation:

$$\mathbf{h}^{(t)} = \phi\left(\alpha \odot \mathbf{A}\mathbf{h}^{(t-1)} \odot \mathbf{B}\mathbf{f}^{(t)} + \beta_1 \odot \mathbf{B}\mathbf{f}^{(t)} + \beta_2 \odot \mathbf{A}\mathbf{h}^{(t-1)}\right). \tag{6.7}$$

Table 6.5 Test accuracy (in %) of one-layer multiplicative recurrent linearities on sMNIST and sCIFAR. ✗ signifies numerical instability during training, causing loss values to become NaNs. BN and LN refer to Batch Normalization and Layer Normalization.

Model	Dataset		Architecture					
	sMNIST	sCIFAR	Residual	Optimizer	Tie_dropout	Norm	GLU	GELU
MI-RNN	93.55	✗	On	AdamW	Off	BN	On	On
	86.55	✗	Off	AdamW	Off	BN	On	On
	19.84	✗	On	SGD	Off	BN	On	On
	78.53	✗	On	AdamW	On	BN	On	On
	56.24	✗	On	AdamW	Off	LN	On	On
	87.64	✗	On	AdamW	Off	BN	Off	On
	96.18	✗	On	AdamW	Off	BN	On	Off
RAC	65.79	✗	On	AdamW	Off	BN	On	On
	48.76	✗	Off	AdamW	Off	BN	On	On
	15.67	✗	On	SGD	Off	BN	On	On
	✗	✗	On	AdamW	On	BN	On	On
	✗	✗	On	AdamW	Off	LN	On	On
	36.68	✗	On	AdamW	Off	BN	Off	On
	79.54	✗	On	AdamW	Off	BN	On	Off

Thus, Wu *et al.* (2016) considered that the general formulation of multiplicative integration in Eq. 6.7 naturally included the additive integration as a special case. Their alterations to multiplicative integration, however, introduced the additive operator, thereby deviating from the inherent multiplicativity of tensor networks. Thus, our analysis solely focuses on the basic form of multiplicative integration, as specified in Eq. 6.5.

We present the test accuracy for one-layer MI-RNN and RAC on sMNIST and sCIFAR in Table 6.5. In contrast to Table 6.3, additive integration outperforms multiplicative integration in terms of robustness and accuracy. Multiplicative integration struggles with images in sCIFAR with a sequence length of 1024, regardless of identity or nonlinear activation usage.

Gradients Exploding/Vanishing

Gradient exploding or vanishing causes graver difficulties when training multiplicative recurrent units compared with additive ones, supported by theoretical and empirical observations. Theoretically, the gradient of hidden states in RAC can be computed as follows:

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-n)}} = \left(\mathbf{A}^\top \operatorname{diag}\left(\mathbf{B}\mathbf{f}^{(t-n+1)}\right)\right)^n, \tag{6.8}$$

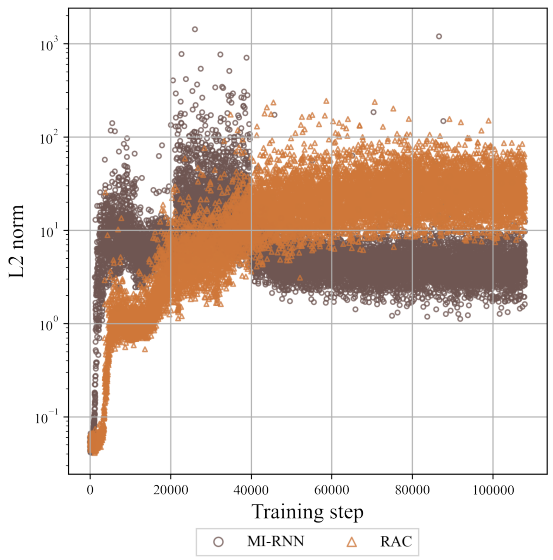

Figure 6.4 L_2 norms of gradients of $\mathbf{A}$ on sCIFAR. MI-RNN refers to the model in the first row of Table 6.5 and RAC to the eighth row.

where $\mathbf{h}^{(t)} = \mathbf{A}\mathbf{h}^{(t-1)} \odot \mathbf{B}\mathbf{f}^{(t)}$ and $n \in [L]$. In comparison to Eq. 6.3, the exponentiation of $\mathbf{B}\mathbf{f}^{(t)}$ introduces more challenges during training within multiplicative recurrent units. The empirical results, as listed in Table 6.5, also demonstrate the difficulties: the models failed to be trained on sCIFAR; even on sMNIST with a sequence length of 784, the effectiveness of the models was notably poorer when contrasted with additive counterparts.

Furthermore, the S4 architecture fails to stabilize the gradient flow in multiplicative recurrent units, as shown in Fig. 6.4. While gradient issues are present, they do not overwhelmingly impede model training. As evidenced by MI-RNN in Fig. 6.4, despite gradient explosion, an accuracy of 93.55% is attained on sMNIST. We posit that the primary obstacle lies in the computation of hidden states during recurrent steps, a matter that we expound upon in the subsequent section.

Exponential Decay in Recurrent Steps

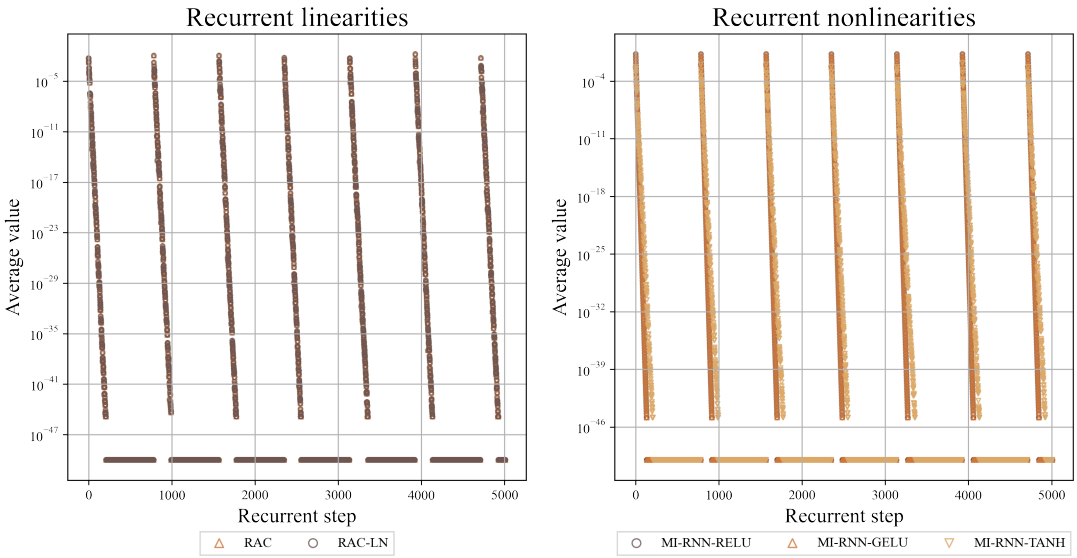

Figure 6.6 Multiplicative hidden states within recurrent steps on sMNIST. We add $\varepsilon = 1 \times 10^{-50}$ to display zero entries on logarithmic-scale axes. (**Left**) Average of $\mathbf{h}^{(t)}$ in linear units. RAC corresponds to the model in the eighth row of Table 6.5 and RAC-LN to the twelfth row. For each sequence ($L = 784$), the hidden states at the first recurrent step have the highest mean and exponentially decrease to 0. (**Right**) Average of $\mathbf{h}^{(t)}$ for nonlinear units. Models include MI-RNN-TANH (first row of Table 6.3), MI-RNN-GELU (replacing Tanh with GELU), and MI-RNN-RELU (replacing Tanh with ReLU).

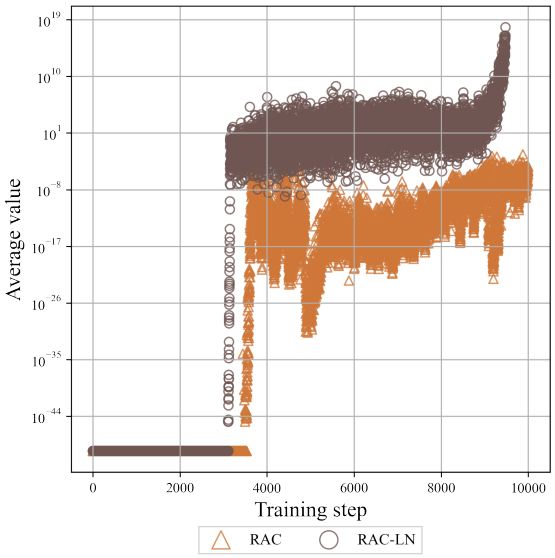

Figure 6.5 Average of $\mathbf{h}^{(L)}$ on sMNIST. We add $\varepsilon = 1 \times 10^{-50}$ to display zero entries on the logarithmic-scale axis. RAC corresponds to the model in the eighth row of Table 6.5 and RAC-LN to the twelfth row. At the 9479th training step, $\mathbf{h}^{(L)}$ in RAC-LN returned NaNs.

We identify that their weakness arises from the computation of hidden states within recurrent steps, analogous to the challenges observed with additive recurrent linearities. Fig. 6.5 displays the average of components of the last sequence hidden state $\mathbf{h}^{(L)}$ during training steps. At the initial $10k$ steps, RAC-LN surpasses float16 precision bounds, leading to NaN values.

While hidden state computation issues exist in both additive integrations and multiplicative integrations, their specifics differ (see Fig. 6.6). For additive recurrent linearities, hidden states exhibit exponential growth (see Fig. 6.3), while a monotonic decay in hidden states is observed for multiplicative recurrent linearities. This decay arises from the recursive computation of $\mathbf{A}\mathbf{h}^{(t-1)} \odot \mathbf{B}\mathbf{f}^{(t)}$, with initializations between these vectors falling within the range of $[0, 1]$.

Table 6.6 Test perplexity of models without residual connections on WikiText-2. D refers to the number of residual blocks. ✗ signifies numerical instability during training, causing loss values to become NaNs.

Model	Segment Length						
	$L = 16$	$L = 32$	$L = 64$	$L = 128$	$L = 256$	$L = 512$	$L = 1024$
MI-RNN ($D = 1$)	88.89	127.00	168.67	209.62	✗	✗	✗
MI-RNN ($D = 6$)	✗	✗	✗	✗	✗	✗	✗
RAC ($D = 1$)	106.79	106.88	210.99	✗	✗	✗	✗
RAC ($D = 6$)	✗	✗	✗	✗	✗	✗	✗

We further analyze the pattern of this monotonic decay by using the least squares method (Marquardt, 1963) to fit the decay with an exponential function in Eq. 6.4. The result yields $a = 0.1604$ and $b = 0.7731$ ($R^2 = 0.9703$). Therefore, to a good approximation, the average of $\mathbf{h}^{(t-1)}$ at each training step in multiplicative recurrents decays exponentially.

Nonlinear activations provide comparatively less advantage to multiplicative recurrent units than to additive units. While hidden states within nonlinear activations in additive recurrent units demonstrate stable behavior (see Fig. 6.3), those in multiplicative recurrent units continue to exhibit an exponential decay (see Fig. 6.6 (left)). Despite the weakness of nonlinear activations, multiplicative recurrent nonlinearities still remain more robust than linearities. Two advantages are observed: **(1)** In comparison to MI-RNN-TANH and other variants, Tanh activation can attenuate the rate of decay in hidden states. This feature aids in managing longer sentence lengths effectively. **(2)** In the context of drawing the right part of Fig. 6.3, certain entries are negative in recurrent linearities, making them unsuitable for representation in a logarithmic figure; thus, we use their absolute values for plotting purposes. The R-squared value for recurrent linearities is only 0.4038. In contrast, the average of hidden states in multiplicative recurrent nonlinearities is positive, indicating a more stable exponential decay trend.

Multiplicative integration consistently underperforms additive integration in language modeling tasks, as shown in Table 6.6 (cf. Table 6.4). For multiplicative integration, we reduce the learning rate to 3×10^{-5} and set the rate within recurrent units at 7.5×10^{-6} to prevent encountering NaNs during training. Even with such low learning rates, the model struggles to handle sequences of significant length ($L = 256$), underscoring the limitations of multiplicative integration in long-range scenarios. Furthermore, these experiments highlight the limitations of the S4 architecture in effectively

constraining the exponential decay of hidden states and gradient flow in multiplicative recurrent units.

After presenting our analysis of linearity and multiplicativity, we express concerns regarding the applicability of tensor networks in long-range scenarios. The core attributes of tensor networks—linearity and additivity—cause considerable training difficulties. These properties might carry theoretical benefits, but their real-world efficacy lags behind that of other recurrent units. Notably, although the S4 architecture can accommodate linearity to a certain extent, it encounters difficulties when handling multiplicative recurrent units.

6.4 Evaluation on Linear Multiplicative Models

To enhance the effectiveness of recurrent units, prior research has explored gating mechanisms (Hochreiter and Schmidhuber, 1997; Chung *et al.*, 2014; Zhou *et al.*, 2016), connections (Hihi and Bengio, 1995; Zhang *et al.*, 2016; Koutnik *et al.*, 2014; Chung *et al.*, 2016), and initializations (Talathi and Vartak, 2015; Arjovsky *et al.*, 2016; Helfrich *et al.*, 2018), as discussed in Sec. 4.2.1. These approaches, however, seldom focus on models that are both linear and multiplicative. In the subsequent section, we shall evaluate our proposed models (i.e., LMM, LMM-MMn, and LMM-MLP) as outlined in Sec. 5.4 to tackle these intricate challenges.

6.4.1 Exponential decay in Complex Numbers

A key distinction between our proposed model and previously evaluated multiplicative recurrent linearities is the complex-valued representations in our models. This property introduces numerical vulnerabilities in the hidden states. In our earlier discussion in Sec. 6.3.2, we analyzed the exponential decay of hidden states within multiplicative recurrent linearities operating in the real domain. When comparing RAC between the real and complex domains, we observe a faster exponential decay of hidden states in the complex domain (see Fig. 6.7). This section analyzes the factors contributing to this phenomenon.

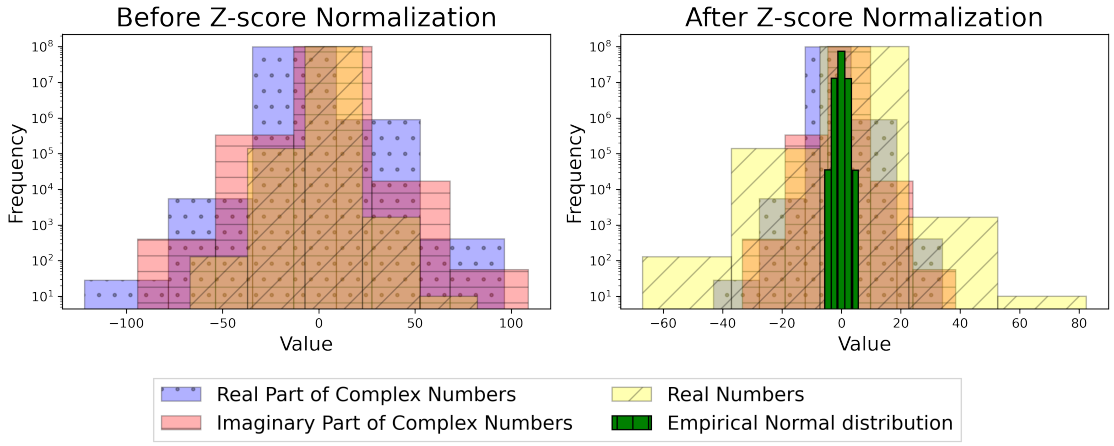

Figure 6.8 Multiplicatitions of four complex or real numbers sampled 100 million times from a normal distribution with mean 0 and variance 1.

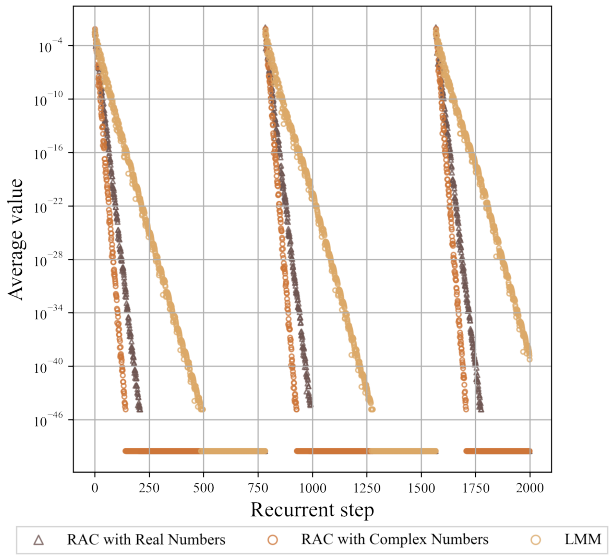

Figure 6.7 Average of $\mathbf{h}^{(t)}$ on sMNIST. We add $\varepsilon = 1 \times 10^{-50}$ to display zero entries on the logarithmic-scale axis. LMM displays the real part of the hidden states.

The use of complex numbers accelerates the decay of hidden states. To ascertain disparities in the multiplication outcomes between complex and real numbers, we emulate the exponential decay of multiplicative recurrent linearities. We generate both real and complex numbers 100 million times from a normal distribution with mean 0 and variance 1. Following this, we execute multiplicative operations on the sets of *four* complex or real numbers. This design aligns with the four multiplication operations in Eq. 5.20 (i.e., the component $j \in [N]$ of $\mathbf{h}^{(t)}$ at timestamp $t \in [L]$ is computed as $h_j^{(t)} = \lambda_j h_j^{(t-1)} \gamma_j x_j^{(t)}$).

As shown in Fig. 6.8 (left), the empirical distributions of complex numbers and real numbers after four multiplication operations are significantly different. We conduct a two-sample Kolmogorov-Smirnov test (Massey Jr, 1951), indicating that the distributions of the real part of complex numbers and real numbers are likely dissimilar ($p-$value < 0.001). Before z-score normalization, the distribution of complex numbers exhibits longer and fatter tails compared with that of real numbers. Note that Fig. 6.8 (left) represents the result of one iteration (i.e., one recurrent step during model training). In practical scenarios, the iteration count could reach 784 (e.g., in sMNIST), leading complex numbers to

more likely exceed normal precision bounds. Theoretical instability may stem from the mechanism of complex number multiplication. Multiplying two real numbers a and b is simply ab. Multiplying two complex numbers $(a+ib)$ by $(c+id)$ results in $(ac-bd)+i(ad+bc)$. As supported by Fig. 6.8 (left), the norm of $(ac-bd)$ or $(ad+bc)$ after four multiplication operations often exceeds that of ab when initialized under a normal distribution.

Although the norm of complex numbers, after multiplications, tends to be larger than that of real numbers, attributing a higher likelihood of generating outliers to complex numbers is not accurate. We compute Fisher's kurtosis (DeCarlo, 1997) before normalizing the two empirical distributions. The kurtosis value for the real part of complex numbers is 21.1, which is surprisingly lower than that of real numbers (i.e., 82.14). A higher kurtosis value indicates either more existing outliers (for the sample kurtosis) or a propensity to produce outliers (for the kurtosis of a probability distribution) (Westfall, 2014). Although this result contradicts the observation from Fig. 6.8 (left), when we normalize the empirical distribution, the distribution of real numbers exhibits longer and fatter tails compared with that of complex numbers, as shown in Fig. 6.8 (right).

Introducing complex-valued representation amplifies training difficulties for the models. This issue can be mitigated by LMMs. Comparing LMM and RAC in Fig. 6.7, it is evident that the average of hidden states decays substantially slower in LMM. This result arises from the setup of learnable parameter matrices in LMM, specifically the replacement of dense matrices (i.e., matrices with almost non-zero elements) with diagonal matrices. Without dense matrices, the component of $j \in [N]$ of $\mathbf{h}^{(t)}$ at timestamp $t \in [L]$ is computed as $\lambda_j h_j^{(t-1)} \gamma_j x_j^{(t)}$ in Eq. 5.20. This computation avoids the summation brought by matrix multiplication, which could make the data decay faster.

However, the exponential pattern persists in LMM, prompting us to employ normalization techniques (see Sec. 5.4.2). As depicted in Fig. 6.9, the values of hidden states stabilize after applying Min-Max normalization or an MLP layer. In the subsequent section, we will evaluate the effectiveness of LMM, LMM-MLP, and LMM-MMn.

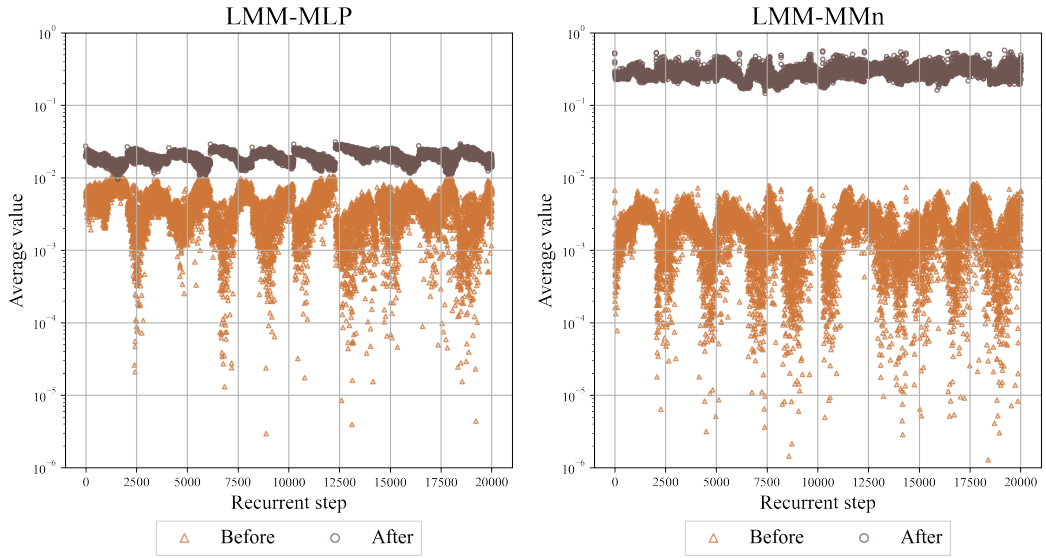

Figure 6.9 Average of the real part of hidden states within recurrent steps on sCIFAR. The legend *before* corresponds to the value of $\mathbf{\Lambda h}^{(t-1)} \odot \mathbf{\Gamma x}^{(t)}$ and *after* to the value after an MLP layer (i.e., LMM-MLP) or Min-Max normalization (i.e., LMM-MMn).

6.4.2 Model Effectiveness

Table 6.7 Test accuracy (in %) of one-layer models on sMNIST and sCIFAR.

	sMNIST	sCIFAR
LMM	✗	✗
LMM-MLP	96.13	44.71
LMM-MMn	79.21	46.15

We evaluate our proposed models in pixel-level classification tasks and language modeling tasks, as presented in Table 6.8 and 6.7. Our initial finding indicates that training multiplicative recurrent linearities using complex numbers is unstable. Even though the hidden states of LMM exhibit slower decay compared with RAC (see Fig. 6.7), it consistently encounters NaNs numbers during training, even when employing a lower learning rate (i.e., 3×10^{-7}).

The exact reasons contributing to this instability remain uncertain. Two factors potentially contributing to this issue are: **(1)** In the previous section, we analyzed that the distribution of complex numbers multiplications has longer and fatter tails compared with that of real numbers (see Fig. 6.8). **(2)** The gradient properties of complex numbers appear to be more sensitive and vulnerable than those of real numbers within the PyTorch framework (Paszke *et al.*, 2019). Given a complex number $x = a + ib$ where $a, b \in \mathbb{R}$, PyTorch treats the real and imaginary parts of numbers (i.e., a and b) as separate learnable parameters instead of a single entity (i.e., x). This distinction can lead to more multiplications

Table 6.8 Test perplexity of one-layer models without residual connections on WikiText-2. LMM-MMn outperforms LMM-MLP and other multiplicative recurrent units on all sequence lengths (cf. Table 6.6)

Model	Segment Length						
	$L = 16$	$L = 32$	$L = 64$	$L = 128$	$L = 256$	$L = 512$	$L = 1024$
LMM (Eq. 5.20)	✗	✗	✗	✗	✗	✗	✗
LMM-MLP (Eq. 5.23)	66.65	65.19	65.63	66.26	66.32	71.50	74.93
LMM-MMn (Eq. 5.24)	65.66	62.01	62.90	65.04	63.57	65.59	66.84

in the gradients when using complex numbers compared with real numbers, potentially resulting in numerical instability. Formally, the gradient of hidden states in Eq. 5.20 is computed as follows:

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-n)}} = \left(\mathbf{\Lambda}^\top \mathrm{diag}\left(\mathbf{\Gamma}\mathbf{x}^{(t)}\right)\right)^n, \tag{6.9}$$

where $n \in [t]$. Since $\mathbf{\Lambda}$ and $\mathbf{\Gamma}$ are diagonal matrices, the component of $j \in [N]$ of $\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-n)}}$ is computed as $(\lambda_j \gamma_j)^n \prod_{i=t-n+1}^{t} x_j^{(i)}$. When treating the real and imaginary parts of numbers as separate parameters, the computation of gradients in the complex domain requires $8n$ times of multiplications than when dealing with real-valued variables (i.e., $2n$). This increased multiplications may lead to numerical instability in the models. To mitigate this challenge, specialized techniques and libraries may be necessary, like JAX (Bradbury *et al.*, 2018), designed for efficiently handling complex-valued gradients. Further exploration of this topic is left for future work.

Our second finding is that the model using complex numbers is effective if the exponential decay in hidden states can be constrained. Table 6.8 lists the test perplexity of LMM-MMn and LMM-MLP on WikiText-2 with varying sentence lengths. LMM-MMn is more effective than multiplicative recurrent units: RACs and MI-RNNs (cf. Table 6.8). Compared with additive recurrent linearities, LMM-MMn is more effective and stable than vRNN-LIN, but slightly worse than vRNN-TANH (see Table 6.4). The reason contributing to this success is that Min-Max normalization or an MLP layer curbs the exponential growth of hidden states, as shown in Fig. 6.9. An advantage of LMM-MMn lies in its recurrent layer having fewer parameters than other recurrent units, reduced from $2H^2$ to $4H$ (H is the dimension of hidden states). Furthermore, considering LMM-MMn's linearity, multiplicativity, and complex-valued representations, we believe that it inherits the fundamental properties of tensor networks. Thus, its success provides empirical evidence for researchers that leverage tensor networks to analyze the expressivity of neural networks (e.g., Khrulkov *et al.*, 2018; Cohen *et al.*, 2016).

7 Discussion and Future Work

We provide a discussion capturing the main findings and implications of the thesis, along with their limitations or weaknesses in Sec. 7.1. In closing, we offer insights and suggestions for future work in Sec. 7.2.

7.1 Main Findings and Implications

Training Recurrent Linearities within the S4 architecture Is Feasible

Linearity is a theoretical property of tensor networks, which contrasts with nonlinearity—a key property for the success of neural network models (e.g., Siegelmann, 2012; Pascanu *et al.*, 2013; Erichson *et al.*, 2020). Our ablation in Sec. 6.3.1 explained the underlying factors in the S4 architecture (Gu, Goel, and Ré, 2022) driving the success of linear recurrent units. When the unit was linear (i.e., with identity activations), it encountered two issues: **(1)** The unstable gradient flow was due to exponentiation forms (see Eq. 6.3). **(2)** Hidden states easily exceeded precision bounds as their values exponentially increased within recurrent steps (see Fig. 6.3 (right)).

The S4 architecture partially addressed the aforementioned issues. Its components, including residual connections, GELU activation, and dropout implementations, stabilized the gradient flow (see Fig. 6.1). To curb the exponential growth, the S4 architecture preserved the increasing pattern of hidden states, but limited their peak values over training steps, which enhanced numerical stability (see Fig. 6.2). However, recurrent linearities within the S4 architecture necessitated a low learning rate (i.e., 3×10^{-4}) to prevent encountering "NaN" values, whether in tasks involving pixel-level classification or language modeling. In contrast, nonlinear recurrent linearities exhibited greater robustness, as nonlinear

activations curbed the growth of hidden states (see Fig. 6.3 (right)). An interesting observation was that ReLU and GELU controlled growth despite lacking an explicit upper bound. Thus, we speculate that this constraint arises not from the upper bound of nonlinear activations, but from their nonlinearity. Taking these findings together, we emphasize that training recurrent linearities appears feasible only when accompanied by the S4 architecture, and the effectiveness of standalone recurrent linearities remains uncertain.

A limitation of the experiments is that we used the least squares method (Marquardt, 1963) to fit an exponential function to the growth of hidden states. These data are however not independently and identically distributed (i.i.d.), potentially resulting in invalid inferences. We shall use statistical techniques such as bootstrap methods (e.g., Genest and Rémillard, 2008) to obtain valid inferences in future work.

Multiplicativity Causes Graver Difficulties than Linearity during Training

Multiplicativity is another theoretical property of tensor networks, yet its impact on model effectiveness has been seldom discussed in prior research. We emphasized that no existing research exclusively used multiplication integration in neural network models. Even studies (e.g., Wu *et al.*, 2016) advocating for the advantages of multiplication integration incorporated additive integration into their models (see Sec. 6.3.2). Our ablation in Sec. 6.3.2 showed that multiplicative recurrent linearities failed to be trained on sCIFAR, and even on sMNIST with a sequence length of 784 (see Table 6.5). We identified two factors that contribute to their poor effectiveness: **(1)** multiplicative recurrent linearities introduced extra exponential forms into their gradient flow compared with the additive units (see Eq. 6.8), and **(2)** their hidden states decayed exponentially, irrespective of the use of nonlinear activations (see Fig. 6.6).

The poor effectiveness of the models on language modeling tasks (see Table 6.6) further substantiates the difficulty of training a linear and multiplicative recurrent unit (i.e., RACs). It is worth noting that we theoretically demonstrate that RACs are a special implementation of TTLMs in Sec. 5.3.2. Thus, our experiment at this stage raises questions about the empirical reliability of using tensor

networks to analyze the expressivity of neural network models, since linearity and multiplicativity, as the fundamental properties of tensor networks, pose massive difficulties for training models.

In future work, this conclusion could be theoretically and experimentally substantiated. On the experimental side, we shall implement previous tensor network language models on language modeling tasks (e.g., WT2; Merity *et al.*, 2016). Although the research objectives and architecture of previous models, particularly focusing on the drawbacks of their training paradigm, are discussed in Sec. 4.3, we have not conducted empirical comparisons because reimplementation is time-consuming. On the theoretical side, we could analyze the upper bound and convergence of the hidden states in multiplicative recurrent units models, in order to validate the difficulties posed by the multiplicativity of tensor networks in model training.

Complex Numbers Introduces Numerical Vulnerabilities

As demonstrated by the previous two findings, linearity and multiplicative pose difficulties in training tensor networks, and our proposed tensor-train language models (TTLMs) are no exception. To devise an effective class of TTLMs, we proposed the Linear Multiplicative Models (LMMs) in Sec. 5.4. LMMs inherit fundamental properties from tensor networks, which are linear, multiplicative, and complex-valued. Though hidden states of LMMs exhibited slower decay compared with that of RACs (see Fig. 6.7 and Table 6.7), LMMs without normalization techniques still failed to be trained on pixel-level classification tasks and language modeling tasks.

We identified two contributing factors related to the introduction of complex numbers. **(1)** We observed a faster exponential decay of hidden states in the complex domain (see Fig. 6.7). We emulated this exponential decay of multiplicative recurrent linearities in Sec. 6.4.1. Our result showed that the distribution of complex numbers exhibited longer and fatter tails compared with that of real numbers (see Fig. 6.8 (left)). This observation was supported by a two-sample Kolmogorov-Smirnov test, indicating likely dissimilarity between the distributions of the real part of complex numbers and real numbers. Interestingly, after Z-score normalization, the distribution of real numbers had longer and fatter tails than that of complex numbers (see Fig. 6.8 (right)). **(2)** We speculate that the second reason

is that our implementation framework (i.e., PyTorch) treats the real and imaginary parts of numbers as separate parameters instead of a single entity, resulting in more multiplications when computing the gradient of complex numbers (see Sec. 6.4.2). This property potentially leads to numerical instability in the models.

A limitation of the experiments is that we did not adapt the model architecture to accommodate complex-valued representations. A study by Sarroff *et al.* (2015) shows that the effectiveness of complex-valued RNNs does not outperform its real-valued counterpart. Several factors could impact the effectiveness of complex-valued models. For example, an alternative deep learning framework, like JAX (Bradbury *et al.*, 2018), which is more suitable for handling complex-valued gradients, is preferred by some researchers (e.g., Orvieto *et al.*, 2023). We will discuss this direction in detail in Sec. 7.2.

Linear, Multiplicative, and Complex-valued Model is Plausible in Practice

Since introducing complex-valued representation amplifies training difficulties for the model, we further develop normalization techniques for LMMs to stabilize the computation of hidden states in the latter part of the sentence (see Sec.5.4.2). When LMMs use a normalization technique such as Min-Max normalization (i.e., LMM-MMn), the model is more effective in language modeling tasks than all multiplicative recurrent units and vanilla RNNs with identity activations (see Table 6.6).

Since LMM-MMn inherits fundamental properties from tensor networks, which are linear, multiplicative, and complex-valued. We believe that its success provides concrete empirical evidence for researchers (e.g., Cohen *et al.*, 2016; Khrulkov *et al.*, 2018) that leverage tensor networks to analyze the expressivity of neural networks.

A limitation of the experiments is that we did not comprehensively compare our models and SOTA models. Although we mentioned SOTA models such as LRUs (Orvieto *et al.*, 2023) and the commonly used LRA tasks (Tay *et al.*, 2020) in our experimental setup (see Sec. 6.2), we did not directly compare our models with them. This decision was primarily because that LMMs exhibit a large effectiveness

gap compared to SOTA models, making the comparison less informative at the current stage. In other words, LRA tasks are hard to be solved by the current version of LMMs. In future research, we need to refine the architecture of LMMs and then compare it with SOTA models on LRA tasks to provide a more comprehensive assessment of its capabilities. We outline a potential direction for improvement in Sec. 7.2.

7.2 Future Work

From this discussion, we discuss the four major findings of the thesis and outline the potential directions to substantiate these findings. We shall summarize them here and then provide detailed insights into the last two directions (which may require more innovative work). In essence, the limitations and possible directions are as follows: **(1)** To enhance the validity of our findings regarding the growth pattern of hidden states, we need to employ statistical techniques, such as bootstrap methods, that do not rely on the i.i.d assumption. **(2)** To verify the training difficulties caused by multiplicativity, we could conduct both experiments and theoretical analysis. For example, we can analyze the convergence of hidden states when using multiplicativity and perform an experimental evaluation of previous tensor network language models. **(3)** The introduction of complex-valued representations requires further modifications to the model architecture. Otherwise, the effectiveness of complex-valued models may not surpass that of their real-valued counterparts. **(4)** There exists a significant gap between the effectiveness of LMMs and SOTA models. Before conducting a comprehensive evaluation of LMMs, we must refine their architecture. We believe the primary focus should be on improving normalization techniques.

Complex-valued Techniques

Tensor networks have emerged as an important theoretical tool to investigate quantum many-body systems (Evenbly and Vidal, 2011; Han and Hung, 2017). From the perspective of quantum mechanics, defining a TN-like model in the complex domain seems to be more natural than in the real domain. From the perspective of the models themselves (i.e., LMMs), we also argue that unlike the symmetric

setting where eigenvalues and eigenvectors are real-valued, one has to allow for complex entries to achieve full equivalence in the non-symmetric case (see Sec. 5.4.1).

In practical scenarios, LMMs however exhibit subpar effectiveness, and we attribute this primarily to the lack of architectural refinement when the model is defined in the complex domain. The implementation of LLMs can be categorized into the split Complex-Valued Neural Networks (split-CVNNs, Lee *et al.*, 2022), where complex-valued input (real and imaginary parts) splits into a pair of real-valued inputs and feeds into the real-valued neural network (RVNNs). Also, the activation function, Min-Max normalization (MMn), operates on real-valued weights (i.e., the real part of the hidden states) instead of complex-valued weights. Though this setup makes the model easier for us to implement, it could lead to *phase distortion* (i.e., the phase information of complex-valued output is not captured accurately) and inaccuracy in approximating the output of the neural network as it is updated by using real-valued gradients, which does not represent the true complex gradient (Savitha *et al.*, 2009).

Thus, to deal with complex-valued representations, we may need to consider both the weights and activation functions in the complex domain (e.g., Hirose, 1992). To achieve this goal, we could consider the following two options in the future: **(1)** The design of activation functions remains a challenging and long-standing task in the field of CVNN (Lee *et al.*, 2022). One direction is to design a suitable complex-valued activation function (i.e., normalization function or factor in LMMs). For instance, Tachibana and Otsuka (2018) discussed four types of ReLU in complex domains as potential options. **(2)** A broader and more ambitious vision entails developing gradient-based algorithms in the complex domain. In our previous discussion in Sec. 2.3, we introduced the learning algorithm for RNNs, *Backpropagation Through Time* (BPTT) (Werbos, 1974; Rumelhart *et al.*, 1985; Werbos, 1990); BPTT is a particular type of *Backpropagation* algorithm (BP) (Rumelhart *et al.*, 1986). The BP algorithm has been extended into complex domains by various researchers, including (Benvenuto and Piazza, 1992; Leung and Haykin, 1991; Nitta, 1997). Thus, devising a new variant of BPTT in the complex domain is a possible direction to enhance model effectiveness.

Normalization Functions

Although normalization functions (e.g., MMn) stabilize gradients and mitigate exponential issues in hidden states, they introduce unexpected difficulties. For instance, while LMMs-MMn is effective language modeling, it struggles in pixel-level classification tasks. We hypothesize that this divergence stems from the need for more coherent information retention in pixel-level classification tasks. Specifically, an activation function alters the value of hidden states at each time, which constantly morphs past information; In the tasks requiring access to long-range past information (e.g., `PathX` in LRA tasks), this constant information morphing impedes the model to retrieve past information. An example that may relate to this observation is the challenges faced by "powerful" neural network models that struggle to do a simple copying task (Smolensky *et al.*, 2022). For example, the model takes an ordered list of five digits (e.g., $[3,9,7,4,7]$), encoding the entire sequence internally, and then reproducing that sequence as the output.

We believe that a pivotal innovation for improving model effectiveness involves devising an elegant normalization factor tied to the original model parameters (i.e., $\mathbf{\Lambda},\mathbf{\Gamma}$ in Eq. 5.20), without introducing additional parameters (e.g., LMM-MLP) or functions (e.g., LMM-MMn). An illustrative example of this can be found in Orvieto *et al.* (2023), where they introduce a normalization factor tied to λ_i in Eq. 5.22 (i.e., $\log(\sqrt{1-|\lambda_i|^2})$). This normalization factor enabled their model to successfully tackle the `PathX` problem with a sequence length of $16k$ tokens. As we introduced in Sec. 4.2.2, the derivation of their normalization factor demands considerable effort, which involves theoretical analysis such as the convergence of hidden states and significant engineering efforts.

In future work, we could theoretically and experimentally validate the information loss w.r.t. time steps caused by normalization functions and devise a novel normalization factor tailored to multiplicative models, with the aim of further enhancing their effectiveness and efficiency.

8

Conclusion

Tensor networks have been used to interpret the theoretical properties of neural networks, albeit without concrete empirical evidence. Focusing on this issue, we introduce a class of tensor-train language models (TTLMs). TTLMs encode the joint probability distribution of sequences into a wave function and learn the conditional probability distributions during training. Our experimental evaluation underscores the difficulties caused by the theoretical properties of tensor networks, including linearity, multiplicativity, and complex-valued representations.

To enhance the effectiveness of TTLMs, we introduce a class of complex-valued variants: the Linear Multiplicative Models (LMMs), which reduce the number of parameters in the recurrent layer and maintain effectiveness comparable to vanilla RNNs. LMMs inherit the fundamental properties of tensor networks; thus, their success in language modeling tasks provides empirical support for prior research that establishes the connections between neural networks and tensor networks. The linearity and multiplicativity of LMMs also offer a novel perspective in a field dominated by nonlinear and additive models.

9 Bibliography

Agarap, Abien Fred (2018). „Deep learning using rectified linear units (relu)“. In: *arXiv preprint arXiv:1803.08375*.

Ahmed, Farag, Ernesto William De Luca, and Andreas Nürnberger (2009). „Revised n-gram based automatic spelling correction tool to improve retrieval effectiveness“. In: *Polibits* 40, pp. 39–48.

Alexander, Rafael N, Glen Evenbly, and Israel Klich (2021). „Exact holographic tensor networks for the Motzkin spin chain“. In: *Quantum* 5, p. 546.

Algoet, Paul H and Thomas M Cover (1988). „A sandwich proof of the Shannon-McMillan-Breiman theorem“. In: *The annals of probability*, pp. 899–909.

Alpay, Tayfun (2021). „Periodicity, Surprisal, Attention: Skip Conditions for Recurrent Neural Networks“. PhD thesis. Staats-und Universitätsbibliothek Hamburg Carl von Ossietzky.

Altmann, Eduardo G, Giampaolo Cristadoro, and Mirko Degli Esposti (2012). „On the origin of long-range correlations in texts“. In: *Proceedings of the National Academy of Sciences* 109.29, pp. 11582–11587.

Andreas, Jacob, Andreas Vlachos, and Stephen Clark (2013). „Semantic parsing as machine translation“. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 47–52.

Arjovsky, Martin, Amar Shah, and Yoshua Bengio (2016). „Unitary evolution recurrent neural networks“. In: *International Conference on Machine Learning*. PMLR, pp. 1120–1128.

Axler, Sheldon (1997). *Linear algebra done right*. Springer Science & Business Media.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton (2016). „Layer normalization“. In: *arXiv preprint arXiv:1607.06450*.

Bahl, Lalit R, Frederick Jelinek, and Robert L Mercer (1983). „A maximum likelihood approach to continuous speech recognition“. In: *IEEE transactions on pattern analysis and machine intelligence* 2, pp. 179–190.

Bak, Per, Chao Tang, and Kurt Wiesenfeld (1987). „Self-organized criticality: An explanation of the 1/f noise“. In: *Physical review letters* 59.4, p. 381.

Bak, Per, Chao Tang, and Kurt Wiesenfeld (1988). „Self-organized criticality“. In: *Physical review A* 38.1, p. 364.

Baldi, Pierre, Kyle Cranmer, Taylor Faucett, Peter Sadowski, and Daniel Whiteson (2016). „Parameterized machine learning for high-energy physics“. In: *arXiv preprint arXiv:1601.07913*.

Belinkov, Yonatan and James Glass (2019). „Analysis methods in neural language processing: A survey“. In: *Transactions of the Association for Computational Linguistics* 7, pp. 49–72.

Bellman, Richard (1966). „Dynamic programming“. In: *Science* 153.3731, pp. 34–37.

Bengio, Yoshua, Réjean Ducharme, and Pascal Vincent (2000). „A neural probabilistic language model“. In: *Advances in neural information processing systems* 13.

Bengio, Yoshua and Paolo Frasconi (1994). „An EM approach to learning sequential behavior“. In: *Advances in Neural Information Processing Systems* 7.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). „Learning long-term dependencies with gradient descent is difficult“. In: *IEEE transactions on neural networks* 5.2, pp. 157–166.

Benvenuto, Nevio and Francesco Piazza (1992). „On the complex backpropagation algorithm“. In: *IEEE Transactions on Signal Processing* 40.4, pp. 967–969.

Bi, Yingyue, Yingcong Lu, Zhen Long, Ce Zhu, and Yipeng Liu (2022). „Chapter 1 - Tensor decompositions: computations, applications, and challenges“. In: *Tensors for Data Processing*. Ed. by Yipeng Liu. Academic Press, pp. 1–30.

Bommasani, Rishi, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, *et al.* (2021). „On the opportunities and risks of foundation models“. In: *arXiv preprint arXiv:2108.07258*.

Born, Max (1926). „Quantenmechanik der stoßvorgänge“. In: *Zeitschrift für physik* 38.11-12, pp. 803–827.

Bradbury, James, Roy Frostig, Peter Hawkins, *et al.* (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13.

Bradbury, James, Stephen Merity, Caiming Xiong, and Richard Socher (2017). „Quasi-Recurrent Neural Networks“. In: *International Conference on Learning Representations*.

Brants, Thorsten, Ashok C Popat, Peng Xu, Franz J Och, and Jeffrey Dean (2007). „Large language models in machine translation“. In.

Bridgeman, Jacob C and Christopher T Chubb (2017). „Hand-waving and interpretive dance: an introductory course on tensor networks“. In: *Journal of physics A: Mathematical and theoretical* 50.22, p. 223001.

Bridle, John S (1990). „Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition“. In: *Neurocomputing*. Springer, pp. 227–236.

Brown, Peter F, Stephen A Della Pietra, Vincent J Della Pietra, Robert L Mercer, *et al.* (1993). „The mathematics of statistical machine translation: Parameter estimation“. In.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, *et al.* (2020). „Language models are few-shot learners“. In: *Advances in neural information processing systems* 33, pp. 1877–1901.

Bubeck, Sébastien, Varun Chandrasekaran, Ronen Eldan, *et al.* (2023). *Sparks of Artificial General Intelligence: Early experiments with GPT-4*. arXiv: 2303.12712 [cs.CL].

Carlini, Nicholas, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom B Brown, Dawn Song, Ulfar Erlingsson, *et al.* (2021). „Extracting Training Data from Large Language Models.“ In: *USENIX Security Symposium*. Vol. 6.

Cartuyvels, Ruben, Graham Spinks, and Marie-Francine Moens (2021). „Discrete and continuous representations and processing in deep learning: Looking forward“. In: *AI Open* 2, pp. 143–159.

Chang, Shiyu, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang (2017). „Dilated recurrent neural networks“. In: *Advances in neural information processing systems* 30.

Chen, Stanley F and Joshua Goodman (1999). „An empirical study of smoothing techniques for language modeling“. In: *Computer Speech & Language* 13.4, pp. 359–394.

Chen, Yanqing, Bryan Perozzi, Rami Al-Rfou, and Steven Skiena (2013). „The expressive power of word embeddings“. In: *arXiv preprint arXiv:1301.3226*.

Cheng, Song, Lei Wang, Tao Xiang, and Pan Zhang (2019). „Tree tensor networks for generative modeling“. In: *Physical Review B* 99.15, p. 155131.

Chung, Junyoung, Sungjin Ahn, and Yoshua Bengio (2016). „Hierarchical multiscale recurrent neural networks“. In: *arXiv preprint arXiv:1609.01704*.

Chung, Junyoung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio (2014). „Empirical evaluation of gated recurrent neural networks on sequence modeling“. In: *NIPS 2014 Workshop on Deep Learning, December 2014*.

Chung, Stephen and Hava Siegelmann (2021). „Turing completeness of bounded-precision recurrent neural networks“. In: *Advances in Neural Information Processing Systems* 34, pp. 28431–28441.

Church, Kenneth Ward (1989). „A stochastic parts program and noun phrase parser for unrestricted text“. In: *International Conference on Acoustics, Speech, and Signal Processing,* IEEE, pp. 695–698.

Cichocki, Andrzej (2014). „Era of big data processing: A new approach via tensor networks and tensor decompositions“. In: *arXiv preprint arXiv:1403.2048*.

Cichocki, Andrzej, Anh-Huy Phan, Qibin Zhao, Namgil Lee, Ivan Oseledets, Masashi Sugiyama, Danilo P Mandic, *et al.* (2017). „Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives“. In: *Foundations and Trends® in Machine Learning* 9.6, pp. 431–673.

Clarkson, Philip and Tony Robinson (2001). „Improved language modelling through better language model evaluation measures“. In: *Computer Speech & Language* 15.1, pp. 39–53.

Clauset, Aaron, Cosma Rohilla Shalizi, and Mark EJ Newman (2009). „Power-law distributions in empirical data“. In: *SIAM review* 51.4, pp. 661–703.

Clerckx, Bruno and Claude Oestges (2013). *MIMO wireless networks: channels, techniques and standards for multi-antenna, multi-user and multi-cell systems*. Academic Press.

Cohen, Nadav, Or Sharir, and Amnon Shashua (2016). „On the expressive power of deep learning: A tensor analysis“. In: *Conference on learning theory*. PMLR, pp. 698–728.

Cohen, Nadav and Amnon Shashua (2016). „Convolutional rectifier networks as generalized tensor decompositions“. In: *International conference on machine learning*. PMLR, pp. 955–963.

Cover, Thomas M (1999). *Elements of information theory*. John Wiley & Sons.

Creswell, Antonia, Murray Shanahan, and Irina Higgins (2022). „Selection-Inference: Exploiting Large Language Models for Interpretable Logical Reasoning“. In: *The Eleventh International Conference on Learning Representations*.

Cui, Jiaxi, Zongjian Li, Yang Yan, Bohua Chen, and Li Yuan (2023). „Chatlaw: Open-source legal large language model with integrated external knowledge bases“. In: *arXiv preprint arXiv:2306.16092*.

Dao, Tri, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré (2022). „Flashattention: Fast and memory-efficient exact attention with io-awareness“. In: *Advances in Neural Information Processing Systems* 35, pp. 16344–16359.

Dauphin, Yann N, Angela Fan, Michael Auli, and David Grangier (2017). „Language modeling with gated convolutional networks“. In: *International Conference on Machine Learning*. PMLR, pp. 933–941.

De Lathauwer, Lieven, Bart De Moor, and Joos Vandewalle (2000). „A multilinear singular value decomposition“. In: *SIAM journal on Matrix Analysis and Applications* 21.4, pp. 1253–1278.

De Mulder, Wim, Steven Bethard, and Marie-Francine Moens (2015). „A survey on the application of recurrent neural networks to statistical language modeling“. In: *Computer Speech & Language* 30.1, pp. 61–98.

De Novais, Eder Miranda, Thiago Dias Tadeu, and Ivandré Paraboni (2010). „Improved text generation using n-gram statistics“. In: *Advances in Artificial Intelligence–IBERAMIA 2010: 12th Ibero-American Conference on AI, Bahía Blanca, Argentina, November 1-5, 2010. Proceedings 12*. Springer, pp. 316–325.

DeCarlo, Lawrence T (1997). „On the meaning and use of kurtosis.“ In: *Psychological methods* 2.3, p. 292.

Deschacht, Koen, Jan De Belder, and Marie-Francine Moens (2012). „The latent words language model“. In: *Computer Speech & Language* 26.5, pp. 384–409.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2018). „Bert: Pre-training of deep bidirectional transformers for language understanding“. In: *arXiv preprint arXiv:1810.04805*.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2016). „Density estimation using real nvp“. In: *arXiv preprint arXiv:1605.08803*.

Driess, Danny, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, *et al.* (2023). „Palm-e: An embodied multimodal language model“. In: *arXiv preprint arXiv:2303.03378*.

Dua, Dheeru and Casey Graff (2017). *UCI Machine Learning Repository*.

Ebeling, Werner and Alexander Neiman (1995). „Long-range correlations between letters and sentences in texts“. In: *Physica A: Statistical Mechanics and its Applications* 215.3, pp. 233–241.

Ebeling, Werner and Thorsten Pöschel (1994). „Entropy and long-range correlations in literary English“. In: *EPL (Europhysics Letters)* 26.4, p. 241.

Erichson, N Benjamin, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W Mahoney (2020). „Lipschitz recurrent neural networks“. In: *arXiv preprint arXiv:2006.12070*.

Evenbly, Glen (n.d.). *Tensors.net: Intro*. `https://www.tensors.net/intro`. Accessed: 2023-02-01.

Evenbly, Glen and Guifré Vidal (2011). „Tensor network states and geometry“. In: *Journal of Statistical Physics* 145, pp. 891–918.

Fannes, Mark, Bruno Nachtergaele, and Reinhard F Werner (1992). „Finitely correlated states on quantum spin chains“. In: *Communications in mathematical physics* 144, pp. 443–490.

Ferris, Andrew J and Guifre Vidal (2012). „Perfect sampling with unitary tensor networks“. In: *Physical Review B* 85.16, p. 165146.

Fonollosa, Jordi, Sadique Sheik, Ramón Huerta, and Santiago Marco (2015). „Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring“. In: *Sensors and Actuators B: Chemical* 215, pp. 618–629.

Gan, Guobing, Peng Zhang, Sunzhu Li, Xiuqing Lu, and Benyou Wang (2022). „MorphTE: Injecting Morphology in Tensorized Embeddings“. In: *Advances in Neural Information Processing Systems*.

Genest, Christian and Bruno Rémillard (2008). „Validity of the parametric bootstrap for goodness-of-fit testing in semiparametric models“. In: *Annales de l'IHP Probabilités et statistiques*. Vol. 44. 6, pp. 1096–1127.

Giles, C, Guo-Zheng Sun, Hsing-Hen Chen, Yee-Chun Lee, and Dong Chen (1989). „Higher order recurrent networks and grammatical inference“. In: *Advances in neural information processing systems* 2.

Ginibre, Jean (1965). „Statistical ensembles of complex, quaternion, and real matrices“. In: *Journal of Mathematical Physics* 6.3, pp. 440–449.

Giovannetti, Vittorio, Simone Montangero, and Rosario Fazio (2008). „Quantum multiscale entanglement renormalization ansatz channels“. In: *Physical review letters* 101.18, p. 180503.

Glorot, Xavier and Yoshua Bengio (2010). „Understanding the difficulty of training deep feedforward neural networks“. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, pp. 249–256.

Goldreich, Oded (2008). „Computational complexity: a conceptual perspective“. In: *ACM Sigact News* 39.3, pp. 35–39.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2020). „Generative adversarial networks“. In: *Communications of the ACM* 63.11, pp. 139–144.

Goudreau, Mark W, C Lee Giles, Srimat T Chakradhar, and Dong Chen (1994). „First-order versus second-order single-layer recurrent neural networks“. In: *IEEE Transactions on Neural Networks* 5.3, pp. 511–513.

Gu, Albert, Karan Goel, Ankit Gupta, and Christopher Ré (2022). „On the parameterization and initialization of diagonal state space models“. In: *Advances in Neural Information Processing Systems* 35, pp. 35971–35983.

Gu, Albert, Karan Goel, and Christopher Ré (2022). „Efficiently modeling long sequences with structured state spaces“. In: *International Conference on Learning Representations*.

Gu, Albert, Caglar Gulcehre, Thomas Paine, Matt Hoffman, and Razvan Pascanu (2020). „Improving the gating mechanism of recurrent neural networks“. In: *International Conference on Machine Learning*. PMLR, pp. 3800–3809.

Gupta, Ankit, Albert Gu, and Jonathan Berant (2022). „Diagonal state spaces are as effective as structured state spaces“. In: *Advances in Neural Information Processing Systems* 35, pp. 22982–22994.

Han, Muxin and Ling-Yan Hung (2017). „Loop quantum gravity, exact holographic mapping, and holographic entanglement entropy“. In: *Physical Review D* 95.2, p. 024011.

Han, Xu, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, *et al.* (2021). „Pre-trained models: Past, present and future“. In: *AI Open* 2, pp. 225–250.

Han, Zhao-Yu, Jun Wang, Heng Fan, Lei Wang, and Pan Zhang (2018). „Unsupervised Generative Modeling Using Matrix Product States“. In: *Physical Review X* 8.3.

Heeman, Peter A (1999). „POS tags and decision trees for language modeling“. In: *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Helfrich, Kyle, Devin Willmott, and Qiang Ye (2018). „Orthogonal recurrent neural networks with scaled Cayley transform“. In: *International Conference on Machine Learning*. PMLR, pp. 1969–1978.

Hendrycks, Dan and Kevin Gimpel (2016). „Gaussian error linear units (gelus)“. In: *arXiv preprint arXiv:1606.08415*.

Hermann, Karl Moritz, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom (2015). „Teaching machines to read and comprehend“. In: *Advances in neural information processing systems* 28.

Hihi, Salah and Yoshua Bengio (1995). „Hierarchical recurrent neural networks for long-term dependencies“. In: *Advances in neural information processing systems* 8.

Hinton, Geoffrey (2022). „The forward-forward algorithm: Some preliminary investigations“. In: *arXiv preprint arXiv:2212.13345*.

Hinton, Geoffrey E (1984). „Distributed representations“. In.

Hirose, Akira (1992). „Proposal of fully complex-valued neural networks“. In: *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. Vol. 4. IEEE, pp. 152–157.

Hochreiter, Sepp, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, *et al.* (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). „Long short-term memory“. In: *Neural computation* 9.8, pp. 1735–1780.

Hopfield, John J (1982). „Neural networks and physical systems with emergent collective computational abilities.“ In: *Proceedings of the national academy of sciences* 79.8, pp. 2554–2558.

Hu, Yuhuang, Adrian Huber, Jithendar Anumula, and Shih-Chii Liu (2018). „Overcoming the vanishing gradient problem in plain recurrent networks“. In: *arXiv preprint arXiv:1801.06105*.

Hur, Yoonhaeng, Jeremy G Hoskins, Michael Lindsey, E Miles Stoudenmire, and Yuehaw Khoo (2022). „Generative modeling via tensor train sketching“. In: *arXiv preprint arXiv:2202.11788*.

Hyland, Stephanie and Gunnar Rätsch (2017). „Learning unitary operators with help from u (n)“. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1.

Ioffe, Sergey and Christian Szegedy (2015). „Batch normalization: Accelerating deep network training by reducing internal covariate shift“. In: *International Conference on Machine Learning*. pmlr, pp. 448–456.

Itskov, Mikhail (2009). *Tensor Algebra and Tensor Analysis for Engineers: With Applications to Continuum Mechanics*. 2nd. Springer Publishing Company, Incorporated.

Iyer, Rukmini, Mari Ostendorf, and Marie Meteer (1997). „Analyzing and predicting language model improvements“. In: *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*. IEEE, pp. 254–261.

Jahn, Alexander and Jens Eisert (2021). „Holographic tensor network models and quantum error correction: a topical review“. In: *Quantum Science and Technology* 6.3, p. 033002.

Jelinek, Fred (1990). „Self-organized language modeling for speech recognition“. In: *Readings in speech recognition*, pp. 450–506.

Jelinek, Fred, Robert L Mercer, Lalit R Bahl, and James K Baker (1977). „Perplexity—a measure of the difficulty of speech recognition tasks“. In: *The Journal of the Acoustical Society of America* 62.S1, S63–S63.

Jelinek, Frederick (1998). *Statistical methods for speech recognition*. MIT press.

Jiang, Zhengbao, Frank F Xu, Jun Araki, and Graham Neubig (2020). „How can we know what language models know?“ In: *Transactions of the Association for Computational Linguistics* 8, pp. 423–438.

Jing, Li, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić (2017). „Tunable efficient unitary neural networks (eunn) and their application to rnns“. In: *International Conference on Machine Learning*. PMLR, pp. 1733–1741.

Jurafsky, Dan (2000). *Speech & language processing*. Pearson Education India.

Kanai, Sekitoshi, Yasuhiro Fujiwara, and Sotetsu Iwamura (2017). „Preventing gradient explosions in gated recurrent units“. In: *Advances in neural information processing systems* 30.

Khrulkov, Valentin, Alexander Novikov, and Ivan Oseledets (2018). „Expressive power of recurrent neural networks“. In: *International Conference on Learning Representations*.

Kingma, Diederik P and Jimmy Ba (2014). „Adam: A method for stochastic optimization“. In: *arXiv preprint arXiv:1412.6980*.

Kingma, Diederik P and Max Welling (2013). „Auto-encoding variational bayes“. In: *arXiv preprint arXiv:1312.6114*.

Klema, Virginia and Alan Laub (1980). „The singular value decomposition: Its computation and some applications“. In: *IEEE Transactions on automatic control* 25.2, pp. 164–176.

Kneser, Reinhard and Hermann Ney (1995). „Improved backing-off for m-gram language modeling“. In: *1995 international conference on acoustics, speech, and signal processing*. Vol. 1. IEEE, pp. 181–184.

Kolda, Tamara G and Brett W Bader (2009). „Tensor decompositions and applications“. In: *SIAM review* 51.3, pp. 455–500.

Koopman, Bernard O and J v Neumann (1932). „Dynamical systems of continuous spectra“. In: *Proceedings of the National Academy of Sciences* 18.3, pp. 255–263.

Kossaifi, Jean, Zachary C Lipton, Arinbjörn Kolbeinsson, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar (2020). „Tensor regression networks“. In: *The Journal of Machine Learning Research* 21.1, pp. 4862–4882.

Koutnik, Jan, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber (2014). „A clockwork rnn“. In: *International Conference on Machine Learning*. PMLR, pp. 1863–1871.

Krizhevsky, Alex, Geoffrey Hinton, *et al.* (2009). „Learning multiple layers of features from tiny images“. In.

Kuhn, Roland and Renato De Mori (1990). „A cache-based natural language model for speech recognition“. In: *IEEE transactions on pattern analysis and machine intelligence* 12.6, pp. 570–583.

Kullback, Solomon and Richard A Leibler (1951). „On information and sufficiency“. In: *The annals of mathematical statistics* 22.1, pp. 79–86.

Le, Quoc V, Navdeep Jaitly, and Geoffrey E Hinton (2015). „A simple way to initialize recurrent networks of rectified linear units“. In: *arXiv preprint arXiv:1504.00941*.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Yann, Sumit Chopra, Raia Hadsell, M Ranzato, and Fujie Huang (2006). „A tutorial on energy-based learning“. In: *Predicting structured data* 1.0.

Lee, ChiYan, Hideyuki Hasegawa, and Shangce Gao (2022). „Complex-valued neural networks: A comprehensive survey“. In: *IEEE/CAA Journal of Automatica Sinica* 9.8, pp. 1406–1426.

Lee, Kenton, Omer Levy, and Luke Zettlemoyer (2017). „Recurrent additive networks“. In: *arXiv preprint arXiv:1705.07393*.

Lee, YC, Gary Doolen, HH Chen, GZ Sun, Tom Maxwell, and HY Lee (1986). *Machine learning using a higher order correlation network*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States); Univ. of ...

Lei, Tao, Yu Zhang, Sida I Wang, Hui Dai, and Yoav Artzi (2018). „Simple Recurrent Units for Highly Parallelizable Recurrence“. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 4470–4481.

Leung, Henry and Simon Haykin (1991). „The complex backpropagation algorithm“. In: *IEEE Transactions on signal processing* 39.9, pp. 2101–2104.

Levine, Yoav, Or Sharir, and Amnon Shashua (2018). „Benefits of depth for long-term memory of recurrent networks“. In: *6th International Conference on Learning Representations (ICLR) workshop*.

Levitin, Daniel J, Parag Chordia, and Vinod Menon (2012). „Musical rhythm spectra from Bach to Joplin obey a 1/f power law“. In: *Proceedings of the National Academy of Sciences* 109.10, pp. 3716–3720.

Lewis, Mike, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer (2019). „Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension“. In: *arXiv preprint arXiv:1910.13461*.

Lezcano-Casado, Mario and David Martınez-Rubio (2019). „Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group“. In: *International Conference on Machine Learning*. PMLR, pp. 3794–3803.

Li, Shuai, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao (2018). „Independently recurrent neural network (indrnn): Building a longer and deeper rnn“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5457–5466.

Lin, Henry W and Max Tegmark (2017). „Criticality in Formal Languages and Statistical Physics“. In: *Entropy* 19, p. 299.

Lin, Tsungnan, Bill G Horne, Peter Tino, and C Lee Giles (1996). „Learning long-term dependencies in NARX recurrent neural networks“. In: *IEEE Transactions on Neural Networks* 7.6, pp. 1329–1338.

Linkenkaer-Hansen, Klaus, Vadim V Nikouline, J Matias Palva, and Risto J Ilmoniemi (2001). „Long-range temporal correlations and scaling behavior in human brain oscillations“. In: *Journal of Neuroscience* 21.4, pp. 1370–1377.

Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov (2019). „Roberta: A robustly optimized bert pretraining approach“. In: *arXiv preprint arXiv:1907.11692*.

Loshchilov, Ilya and Frank Hutter (2017). „Decoupled weight decay regularization“. In: *arXiv preprint arXiv:1711.05101*.

MacKay, David JC and David JC Mac Kay (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Manaris, Bill, Penousal Machado, Clayton McCauley, Juan Romero, and Dwight Krehbiel (2005). „Developing fitness functions for pleasant music: Zipf's law and interactive evolution systems“. In: *Applications of Evolutionary Computing: EvoWorkkshops 2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC Lausanne, Switzerland, March 30-April 1, 2005 Proceedings*. Springer, pp. 498–507.

Marcinkiewicz, Mary Ann (1994). „Building a large annotated corpus of English: The Penn Treebank“. In: *Using Large Corpora* 273.

Marcus, Gary F (1998). „Rethinking eliminative connectionism“. In: *Cognitive psychology* 37.3, pp. 243–282.

Markov, Andreĭ Andreevich (2006). „An example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains“. In: *Science in Context* 19.4, pp. 591–600.

Marquardt, Donald W (1963). „An algorithm for least-squares estimation of nonlinear parameters“. In: *Journal of the society for Industrial and Applied Mathematics* 11.2, pp. 431–441.

Martin, David, Charless Fowlkes, Doron Tal, and Jitendra Malik (2001). „A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics“. In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Vol. 2. IEEE, pp. 416–423.

Martin, Eric and Chris Cundy (2017). „Parallelizing linear recurrent neural nets over sequence length“. In: *arXiv preprint arXiv:1709.04057*.

Martin, Sven C, Jörg Liermann, and Hermann Ney (1997). „Adaptive topic-dependent language modelling using word-based varigrams“. In: *Fifth European Conference on Speech Communication and Technology*.

Massey Jr, Frank J (1951). „The Kolmogorov-Smirnov test for goodness of fit“. In: *Journal of the American statistical Association* 46.253, pp. 68–78.

Maupomé, Diego and Marie-Jean Meurs (May 2020). „Language Modeling with a General Second-Order RNN“. English. In: *Proceedings of the 12th Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, pp. 4749–4753.

Meister, Clara and Ryan Cotterell (Aug. 2021). „Language Model Evaluation Beyond Perplexity“. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, pp. 5328–5339.

Melis, Gábor, Tomáš Kočiskỳ, and Phil Blunsom (2019). „Mogrifier LSTM“. In: *International Conference on Learning Representations*.

Merity, Stephen, Caiming Xiong, James Bradbury, and Richard Socher (2016). „Pointer sentinel mixture models“. In: *arXiv preprint arXiv:1609.07843*.

Mhammedi, Zakaria, Andrew Hellicar, Ashfaqur Rahman, and James Bailey (2017). „Efficient orthogonal parametrisation of recurrent neural networks using householder reflections“. In: *International Conference on Machine Learning*. PMLR, pp. 2401–2409.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). „Efficient estimation of word representations in vector space“. In: *arXiv preprint arXiv:1301.3781*.

Mikolov, Tomas, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur (2010). „Recurrent neural network based language model.“ In: *Interspeech*. Vol. 2. 3. Makuhari, pp. 1045–1048.

Mikolov, Tomáš, Wen-tau Yih, and Geoffrey Zweig (2013). „Linguistic regularities in continuous space word representations“. In: *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pp. 746–751.

Miller, George A and Jennifer A Selfridge (1950). „Verbal context and the recall of meaningful material“. In: *The American journal of psychology* 63.2, pp. 176–185.

Miller, Jacob, Guillaume Rabusseau, and John Terilla (2021). „Tensor networks for probabilistic sequence modeling“. In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 3079–3087.

Min, Bonan, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth (2021). „Recent advances in natural language processing via large pre-trained language models: A survey“. In: *ACM Computing Surveys*.

Montemurro, Marcelo A and Pedro A Pury (2002). „Long-range fractal correlations in literary corpora“. In: *Fractals* 10.04, pp. 451–461.

Montúfar, Guido F, Johannes Rauh, and Nihat Ay (2011). „Expressive power and approximation errors of restricted Boltzmann machines“. In: *Advances in neural information processing systems* 24.

Moore, Robert C and Chris Quirk (2009). „Improved smoothing for N-gram language models based on ordinary counts“. In: *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pp. 349–352.

Mora, Thierry and William Bialek (2011). „Are biological systems poised at criticality?“ In: *Journal of Statistical Physics* 144, pp. 268–302.

Nitta, Tohru (1997). „An extension of the back-propagation algorithm to complex numbers“. In: *Neural Networks* 10.8, pp. 1391–1415.

Novikov, Alexander, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov (2015). „Tensorizing neural networks“. In: *Advances in neural information processing systems* 28.

Novikov, Georgii S, Maxim E Panov, and Ivan V Oseledets (2021). „Tensor-train density estimation“. In: *Uncertainty in Artificial Intelligence*. PMLR, pp. 1321–1331.

Ontanon, Santiago, Joshua Ainslie, Zachary Fisher, and Vaclav Cvicek (2022). „Making Transformers Solve Compositional Tasks“. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3591–3607.

OpenAI (2023). *GPT-4 Technical Report*. arXiv: 2303.08774 [cs.CL].

Orús, Román (2014). „A practical introduction to tensor networks: Matrix product states and projected entangled pair states“. In: *Annals of physics* 349, pp. 117–158.

Orús, Román (2019). „Tensor networks for complex quantum systems“. In: *Nature Reviews Physics* 1.9, pp. 538–550.

Orvieto, Antonio, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De (2023). „Resurrecting Recurrent Neural Networks for Long Sequences“. In: *arXiv preprint arXiv:2303.06349*.

Oseledets, Ivan V (2011). „Tensor-train decomposition“. In: *SIAM Journal on Scientific Computing* 33.5, pp. 2295–2317.

Östlund, Stellan and Stefan Rommer (1995). „Thermodynamic limit of density matrix renormalization“. In: *Physical review letters* 75.19, p. 3537.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). „On the difficulty of training recurrent neural networks“. In: *International Conference on Machine Learning*. Pmlr, pp. 1310–1318.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, *et al.* (2019). „Pytorch: An imperative style, high-performance deep learning library“. In: *Advances in neural information processing systems* 32.

Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). „Glove: Global vectors for word representation“. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Penrose, Roger (1971). „Applications of negative dimensional tensors“. In: *Combinatorial mathematics and its applications* 1, pp. 221–244.

Perez-Garcia, David, Frank Verstraete, Michael M Wolf, and J Ignacio Cirac (2006). „Matrix product state representations“. In: *arXiv preprint quant-ph/0608197*.

Pestun, Vasily, John Terilla, and Yiannis Vlassopoulos (2017). „Language as a matrix product state“. In: *arXiv preprint arXiv:1711.01416*.

Pestun, Vasily and Yiannis Vlassopoulos (2017). „Tensor network language model“. In: *arXiv preprint arXiv:1710.10248*.

Peters, Jochen and Dietrich Klakow (1999). „Compact maximum entropy language models“. In: *Proceedings of the IEEE workshop on automatic speech recognition and understanding*.

Peters, Matthew E, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer (2018). „Deep contextualized word representations“. In: *arXiv preprint arXiv:1802.05365*.

Petroni, Fabio, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel (2019). „Language models as knowledge bases?“ In: *arXiv preprint arXiv:1909.01066*.

Pfeifer, Robert NC, Glen Evenbly, and Guifré Vidal (2009). „Entanglement renormalization, scale invariance, and quantum criticality“. In: *Physical Review A* 79.4, p. 040301.

Potamianos, Gerasimos and Frederick Jelinek (1998). „A study of n-gram and decision tree letter language modeling methods“. In: *Speech Communication* 24.3, pp. 171–192.

Rabusseau, Guillaume, Tianyu Li, and Doina Precup (2019). „Connecting weighted automata and recurrent neural networks through spectral learning“. In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 1630–1639.

Radev, D (2008). „Clair collection of fraud email, acl data and code repository“. In: *ADCR2008T001*.

Radford, Alec, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, *et al.* (2018). „Improving language understanding by generative pre-training“. In.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, *et al.* (2019). „Language models are unsupervised multitask learners“. In: *OpenAI blog* 1.8, p. 9.

Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu (2020). „Exploring the limits of transfer learning with a unified text-to-text transformer“. In: *The Journal of Machine Learning Research* 21.1, pp. 5485–5551.

Roe, Byron P, Hai-Jun Yang, Ji Zhu, Yong Liu, Ion Stancu, and Gordon McGregor (2005). „Boosted decision trees as an alternative to artificial neural networks for particle identification“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 543.2-3, pp. 577–584.

Rommer, Stefan and Stellan Östlund (1997). „Class of ansatz wave functions for one-dimensional spin systems and their relation to the density matrix renormalization group“. In: *Physical review b* 55.4, p. 2164.

Rosenfeld, Ronald (1994). *Adaptive statistical language modeling; a maximum entropy approach*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE.

Rosenfeld, Ronald (2000). „Two decades of statistical language modeling: Where do we go from here?“ In: *Proceedings of the IEEE* 88.8, pp. 1270–1278.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). „Learning representations by back-propagating errors“. In: *nature* 323.6088, pp. 533–536.

Sahu, Kishore Kumar (2015). „Normalization: A preprocessing stage“. In: *arXiv preprint arXiv:1503.06462*.

Salehinejad, Hojjat, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee (2018). „Recent advances in recurrent neural networks“. In: *arXiv preprint arXiv:1801.01078*.

Sap, Maarten, Ronan Le Bras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A Smith, and Yejin Choi (2019). „Atomic: An atlas of machine commonsense for if-then reasoning“. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01, pp. 3027–3035.

Sarroff, Andy M, Victor Shepardson, and Michael A Casey (2015). „Learning representations using complex-valued nets“. In: *arXiv preprint arXiv:1511.06351*.

Savitha, Ramaswamy, Sundaram Suresh, N Sundararajan, and P Saratchandran (2009). „A new learning algorithm with logarithmic performance index for complex-valued neural networks“. In: *Neurocomputing* 72.16-18, pp. 3771–3781.

Schollwöck, Ulrich (2011). „The density-matrix renormalization group in the age of matrix product states“. In: *Annals of physics* 326.1, pp. 96–192.

Schwenk, Holger, Daniel Déchelotte, and Jean-Luc Gauvain (2006). „Continuous space language models for statistical machine translation“. In: *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pp. 723–730.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (2016). „Neural Machine Translation of Rare Words with Subword Units“. In: *54th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics (ACL), pp. 1715–1725.

Shannon, Claude E (1948). „A mathematical theory of communication“. In: *The Bell system technical journal* 27.3, pp. 379–423.

Shannon, Claude E (1951). „Prediction and entropy of printed English“. In: *Bell system technical journal* 30.1, pp. 50–64.

Shen, Yikang, Shawn Tan, Alessandro Sordoni, and Aaron Courville (2019). „Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks“. In: *International Conference on Learning Representations*.

Shi, Y-Y, L-M Duan, and Guifre Vidal (2006). „Classical simulation of quantum many-body systems with a tree tensor network“. In: *Physical review a* 74.2, p. 022320.

Siegelmann, Hava T (2012). *Neural networks and analog computation: beyond the Turing limit*. Springer Science & Business Media.

Silverman, Bernard W (1986). *Density estimation for statistics and data analysis*. Vol. 26. CRC press.

Sipser, Michael (1996). „Introduction to the Theory of Computation“. In: *ACM Sigact News* 27.1, pp. 27–29.

Smith, Jimmy TH, Andrew Warrington, and Scott W Linderman (2023). „Simplified state space layers for sequence modeling“. In: *International Conference on Learning Representations*.

Smith, Shaden, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, *et al.* (2022). „Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model“. In: *arXiv preprint arXiv:2201.11990*.

Smolensky, Paul, Richard McCoy, Roland Fernandez, Matthew Goldrick, and Jianfeng Gao (2022). „Neurocompositional computing: From the Central Paradox of Cognition to a new generation of AI systems“. In: *AI Magazine* 43.3, pp. 308–322.

Soares, Livio Baldini, Nicholas Fitzgerald, Jeffrey Ling, and Tom Kwiatkowski (2019). „Matching the Blanks: Distributional Similarity for Relation Learning“. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2895–2905.

Stokes, James and John Terilla (2019). „Probabilistic Modeling with Matrix Product States“. In: *Entropy* 21.12, p. 1236.

Stoudenmire, E. Miles and David J. Schwab (2016). „Supervised Learning with Quantum-Inspired Tensor Networks“. In: *arXiv preprint arXiv:1605.05775*.

Sutskever, Ilya and Geoffrey Hinton (2010). „Temporal-kernel recurrent neural networks“. In: *Neural Networks* 23.2, pp. 239–243.

Sutskever, Ilya, James Martens, and Geoffrey E Hinton (2011). „Generating text with recurrent neural networks“. In: *ICML*.

Tachibana, Kanta and Kentaro Otsuka (2018). „Wind prediction performance of complex neural network with ReLU activation function“. In: *2018 57th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*. IEEE, pp. 1029–1034.

Tagliazucchi, Enzo, Pablo Balenzuela, Daniel Fraiman, and Dante R Chialvo (2012). „Criticality in large-scale brain fMRI dynamics unveiled by a novel point process analysis“. In: *Frontiers in physiology* 3, p. 15.

Talathi, Sachin S and Aniket Vartak (2015). „Improving performance of recurrent neural network with relu nonlinearity“. In: *arXiv preprint arXiv:1511.03771*.

Tanaka-Ishii, Kumiko (2021). *Statistical Universals of Language: Mathematical Chance vs. Human Choice*. Springer Nature.

Tang, Xun, Yoonhaeng Hur, Yuehaw Khoo, and Lexing Ying (2022). „Generative modeling via tree tensor network states“. In: *arXiv preprint arXiv:2209.01341*.

Tay, Yi, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler (2020). „Long range arena: A benchmark for efficient transformers“. In: *arXiv preprint arXiv:2011.04006*.

Thoppilan, Romal, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, *et al.* (2022). „Lamda: Language models for dialog applications“. In: *arXiv preprint arXiv:2201.08239*.

Tomita, Masaru (1982). „Dynamic construction of finite-state automata from examples using hill-climbing.“ In: *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, pp. 105–108.

Touvron, Hugo, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, *et al.* (2023). „Llama: Open and efficient foundation language models“. In: *arXiv preprint arXiv:2302.13971*.

Trinh, Trieu H and Quoc V Le (2018). „A simple method for commonsense reasoning“. In: *arXiv preprint arXiv:1806.02847*.

Van den Oord, Aaron, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, *et al.* (2016). „Conditional image generation with pixelcnn decoders“. In: *Advances in neural information processing systems* 29.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). „Attention is all you need“. In: *Advances in neural information processing systems* 30.

Verstraete, Frank and J Ignacio Cirac (2004). „Renormalization algorithms for quantum-many body systems in two and higher dimensions“. In: *arXiv preprint cond-mat/0407066*.

Verstraete, Frank, Valentin Murg, and J Ignacio Cirac (2008). „Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems“. In: *Advances in physics* 57.2, pp. 143–224.

Vidal, Guifré (2008). „Class of quantum many-body states that can be efficiently simulated“. In: *Physical review letters* 101.11, p. 110501.

Vidal, Guifre (2007). „Entanglement renormalization“. In: *Physical review letters* 99.22, p. 220405.

Vorontsov, Eugene, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal (2017). „On orthogonality and learning recurrent networks with long term dependencies“. In: *International Conference on Machine Learning*. PMLR, pp. 3570–3578.

Wang, Dilin, Chengyue Gong, and Qiang Liu (2019). „Improving neural language modeling via adversarial training“. In: *International Conference on Machine Learning*. PMLR, pp. 6555–6565.

Wang, Haochun, Chi Liu, Nuwa Xi, Zewen Qiang, Sendong Zhao, Bing Qin, and Ting Liu (2023). „Huatuo: Tuning llama model with chinese medical knowledge“. In: *arXiv preprint arXiv:2304.06975*.

Wang, Qi, Yue Ma, Kun Zhao, and Yingjie Tian (2020). „A comprehensive survey of loss functions in machine learning“. In: *Annals of Data Science*, pp. 1–26.

Wang, Shaojun, Dale Schuurmans, Fuchun Peng, and Yunxin Zhao (2005). „Combining statistical language models via the latent maximum entropy principle“. In: *Machine Learning* 60, pp. 229–250.

Wei, Chengwei, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo (2023). „An Overview on Language Models: Recent Developments and Outlook“. In: *arXiv preprint arXiv:2303.05759*.

Werbos, Paul (1974). „Beyond regression: New tools for prediction and analysis in the behavioral sciences“. In: *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*.

Werbos, Paul J (1990). „Backpropagation through time: what it does and how to do it“. In: *Proceedings of the IEEE* 78.10, pp. 1550–1560.

Westfall, Peter H (2014). „Kurtosis as peakedness, 1905–2014. RIP“. In: *The American Statistician* 68.3, pp. 191–195.

White, Steven R (1992). „Density matrix formulation for quantum renormalization groups“. In: *Physical review letters* 69.19, p. 2863.

Wu, Shijie, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann (2023). „Bloomberggpt: A large language model for finance“. In: *arXiv preprint arXiv:2303.17564*.

Wu, Yuhuai, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Russ R Salakhutdinov (2016). „On multiplicative integration with recurrent neural networks“. In: *Advances in neural information processing systems* 29.

Wu, Zhizheng and Simon King (2016). „Investigating gated recurrent networks for speech synthesis“. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 5140–5144.

Yang, Hongyang, Xiao-Yang Liu, and Christina Dan Wang (2023). „FinGPT: Open-Source Financial Large Language Models“. In: *arXiv preprint arXiv:2306.06031*.

Yang, Songhua, Hanjia Zhao, Senbin Zhu, Guangyu Zhou, Hongfei Xu, Yuxiang Jia, and Hongying Zan (2023). „Zhongjing: Enhancing the Chinese Medical Capabilities of Large Language Model through Expert Feedback and Real-world Multi-turn Dialogue“. In: *arXiv preprint arXiv:2308.03549*.

Yin, Kayo and Graham Neubig (2022). „Interpreting Language Models with Contrastive Explanations“. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 184–198.

Zhang, Saizheng, Yuhuai Wu, Tong Che, Zhouhan Lin, Roland Memisevic, Russ R Salakhutdinov, and Yoshua Bengio (2016). „Architectural complexity measures of recurrent neural networks“. In: *Advances in neural information processing systems* 29.

Zhou, Guo-Bing, Jianxin Wu, Chen-Lin Zhang, and Zhi-Hua Zhou (2016). „Minimal gated unit for recurrent neural networks“. In: *International Journal of Automation and Computing* 13.3, pp. 226–234.