Towards Language Modeling Using Tensor Trains

Anonymous Authors[1]

Abstract

Tensor networks have previously shown great potential in language modeling in theory but lack practical implementations in real natural language tasks. To make it practical, we propose a novel tensor network language model based on the simplest tensor network (i.e., tensor trains), called 'Tensor Train Language Model' (TTLM). TTLM represents sentences in an exponential space constructed by the tensor product of words, but in practice alternatively, computes the probabilities of sentences in a low-dimensional fashion. Experimental evaluations on real language modeling tasks show that the proposed variants of TTLM (i.e., TTLM-Large and TTLM-Tiny) outperform the vanilla Recurrent Neural Networks (RNNs) with low-scale of hidden units. Interestingly, we demonstrate that the architectures of Second-order RNNs, Recurrent Arithmetic Circuits, and Multiplicative Integration RNNs are, essentially, special cases of that of TTLM.[1]

1. Introduction

Human languages, like many biological systems, including families of proteins, genomes, and neurons in the brain, have significant long-range correlations that decay with a power law (Tagliazucchi et al., 2012; Mora & Bialek, 2011). Current network models like LSTMs (Hochreiter & Schmidhuber, 1997) are hard to match long-range and higher-order statistics of natural languages (Lin & Tegmark, 2016).

Recently, researchers have turned to tensor network language modeling, which contains models that exhibit correlation functions that decay with the power law (Pestun & Vlassopoulos, 2017; Pestun et al., 2017; Miller et al., 2021). Tensor networks are, roughly, decompositions of large tensors into sets of smaller tensors and have been employed in physics, mathematics, and machine learning (Cohen et al., 2016). However, the so-called 'tensor network language model' is either a concept that needs to be proved practically (Pestun & Vlassopoulos, 2017) or unsuitable in real-world language modeling tasks (Miller et al., 2021) due to their way of modeling probabilities. Towards making tensor network language modeling practical, we make the first step to applying it to real language modeling datasets.

As proof-of-concept work, we derive a Tensor Train Language Model (TTLM) (the simplest tensor network). Technically, we represent a sentence based on the exponential semantic space constructed by the tensor product of word representations. The probability of the sentence is defined by the inner product of two high-dimensional tensors: the input $\Phi(X)$ and the global coefficients $\mathcal{A}$, and decomposed into conditional probabilities.

Under the framework of TTLM, we propose two variants: TTLM-Tiny and TTLM-Large. Also, we clarify the relationship between the proposed TTLM and a series of Recurrent Neural Networks (RNNs) (i.e., Second-order RNNs (Goudreau et al., 1994), Recurrent Arithmetic Circuits (RACs) (Levine et al., 2018), and Multiplicative Integration RNNs (MI-RNNs) (Wu et al., 2016)). These connections open a new eye to understanding RNNs and give some natural implementations for TTLM.

We benchmark these TTLM variants and analyze the difference in their working mechanism and behaviors. Experimental results on the language modeling task show that our TTLM variants could outperform than Vanilla-RNNs under the same training setting. These demonstrate the feasibility of TTLM.

The main contributions of our work can be summarized as follows:

- We propose a novel Tensor Train Language Model, as an illustration of how tensor networks can be applied to real-world language modeling datasets.
- We propose two novel TTLM variants, TTLM-Large and TTLM-Tiny, and theoretically demonstrate the relationship between TTLM and a series of existing

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

[1]The code is available at https://github.com/tensortrainlm/tensortrainlm.

RNNs.

- Compared to Vanilla-RNNs on WikiText-2 and PTB datasets, TTLM-Large reduces perplexity by 14.3 and 16.0, respectively, and TTLM-Tiny reduces perplexity by 1.7 and 8.5, respectively.

2. Related Work

Previous studies on tensor networks in machine learning have mainly been devoted to analyzing the theoretical properties of neural networks. A better understanding of feed-forward, convolutional and recurrent architectures has been gained, including compression parameters (Novikov et al., 2015), expressive power (Cohen et al., 2016; Cohen & Shashua, 2016; Khrulkov et al., 2018), and depth efficiency for long-term memory (Levine et al., 2018). For sequence modeling tasks in NLP, there are two stages of the previous research.

Theoretical Proposals. (Pestun & Vlassopoulos, 2017) propose a tensor network language model aims to construct the long-range correlation in real-world language modeling, (Pestun et al., 2017) propose a quantum statistical language model on a one-dimensional lattice which is called trace-density model. To the best of our knowledge, these tensor network language models have remained a theoretical proposal instead of an empirical one.

Sequence Modeling. (Novikov et al., 2021) propose a new efficient tensor train-based approach to tensor-train density estimation that allows efficient computation of probability density function. (Miller et al., 2021) apply a recurrent tensor network, uniform matrix product state, to the probabilistic sequence modeling while opening significant new research directions in the design of sequential generative models. However, probably due to the efficiency issue, most of the existing models have only been tested on small vocabulary size and simple datasets like Tomita grammars (Tomita, 1982), further evaluation on moderately-scaled natural language datasets is necessary to fully assess their performance. We provide a comparison of the datasets we used with related work in the Appendix: C.

This paper is the first work to derive a tensor network language model in a way that can be applied to real-world language modeling datasets. The efforts to improve efficiency are twofold. First, we do not calculate the probability normalization term used for the total probability law; instead, we turn to calculate conditional probabilities based on context representations, as described in Sec. 4.2.3. Second, we further decompose TT cores and use low-scale hidden units, see in Sec. 5.1.

3. Preliminaries

We briefly recapitulate basic notions and notations[2]; full technical introductions can be found in standard textbooks (Bi et al., 2022; Itskov, 2009).

Notation. For the purposes of this paper, every *tensor* $\mathbf{A}$ is a multidimensional array of elements (called *components*) of $\mathbb{R}$, each denoted by its integer coordinates in the array; e.g., for a two-dimensional array, the component at position $i, j \in \mathbb{N}$ is denoted A_{ij}. The *order* of a tensor is how many indices it has (e.g., a vector $\boldsymbol{v}$ is a first-order tensor, a matrix $\boldsymbol{M}$ is a second-order tensor, etc.). The *dimension* of a tensor refers to the number of values that a particular index (or so-called *mode*) can take, e.g., the dimension of $\mathbf{B} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is $I_1 \times I_2 \times I_3$.

Tensor Product (Cohen et al., 2016). For two tensors $\mathbf{C} \in \mathbb{R}^{I_1 \times \cdots \times I_j}$ (order j) and $\mathbf{D} \in \mathbb{R}^{I_{j+1}, \times \cdots \times I_{j+k}}$ (order k), their *tensor product* is denoted by $\otimes$ and return a tensor $E_{i_1 \cdots i_{j+k}} = C_{i_1 \ldots i_j} \cdot D_{i_{j+1} \cdots i_{j+k}}$ (order $j + k$). Notice that in the case $j = k = 1$, the tensor product reduces to an outer product between vectors.

Generalized Inner Product (Kossaifi et al., 2020). For two tensor $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ of the same size, their *inner product* is defined as $\langle \mathbf{X}, \mathbf{Y} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} X_{i_1,i_2,\ldots,i_N} Y_{i_1,i_2,\ldots,i_N}$. For two tensors $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N \times I_x}$ and $\mathbf{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots I_N \times I_y}$ sharing N modes of the same size, the "generalized inner product" is calculated as

$$\langle \mathbf{X}, \mathbf{Y} \rangle_N = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} X_{i_1,i_2,\ldots,i_N} Y_{i_1,i_2,\ldots,i_N}$$

with $\langle \mathbf{X}, \mathbf{Y} \rangle_N \in \mathbb{R}^{I_x \times I_y}$.

4. Language Modeling Using Tensor Trains

We introduce a language model in tensor space in Sec. 4.1, and define our *Tensor Train Language Model* in Sec. 4.2.

4.1. Language Models in a Tensor Space

Natural language typically has complex dependencies between features (e.g., tokens or words) (Hou et al., 2013)[3] that are not captured well by standard methods such as feature concatenation. One could also see a similar interaction between any arbitrary features in factorization machines (Rendle, 2010). Given text consists of N words

[2] Most of the notations here follow the textbook *Deep Learning* (Goodfellow et al., 2016).

[3] Such dependencies (including collocation) have been viewed as an analogy of entanglement (Hou et al., 2013).

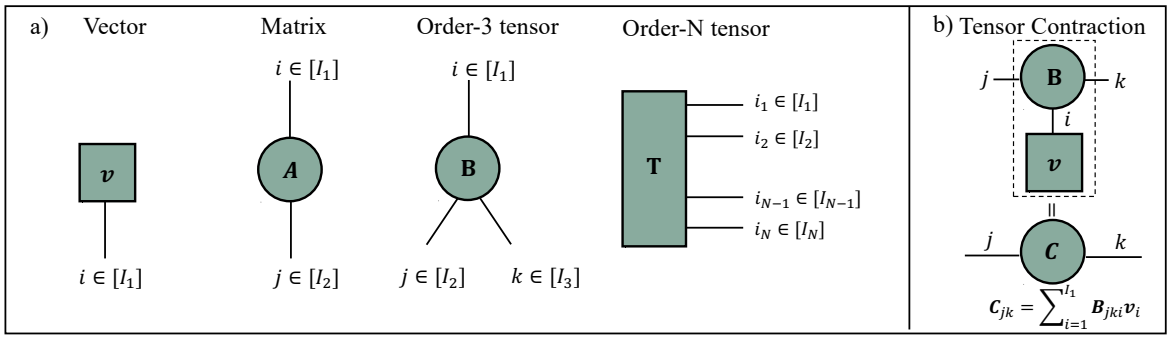

Figure 1. A quick introduction to *tensor diagram notation*. There are two rules of tensor diagrams: (1) tensors are notated by solid shapes with a number of 'legs' corresponding to their indices; (2) connecting two index lines implies a *contraction* or summation over the connected indices. In this paper, we augment our equations with these diagrams to make them easier to understand.

$X = [x^{(1)}, x^{(2)}, \cdots, x^{(N)}]$ and a feature extractor $\boldsymbol{f}_i \in \mathbb{R}^{I_i}$ (it can be one-hot encoding or word embedding), we now define a representation of X designed to capture these dependencies:

$$\begin{aligned}\Phi(X) &= \boldsymbol{f}_1(x^{(1)}) \otimes \boldsymbol{f}_2(x^{(2)}) \cdots \otimes \boldsymbol{f}_N(x^{(N)}) \\ &= \bigotimes_{i=1}^{N} \boldsymbol{f}_i(x^{(i)})\end{aligned} \tag{1}$$

where the tensor space is $\mathbb{R}^{I_1} \otimes \mathbb{R}^{I_2} \otimes \cdots \otimes \mathbb{R}^{I_N}$. Each component of $\boldsymbol{f}_i$ represents independent meaning-bearing units, such as morphemes or latent factors. For simplicity, we assume that a text shares the same one-hot encoding $\boldsymbol{f}(x^{(t)}) \in \mathbb{R}^{|V|}$ in later sections. Consequently, $\Phi(X)$ is a $|V|^N$-dimensional tensor that records all possible combinations of words in X.

Inspired by (Zhang et al., 2019; Kossaifi et al., 2020), we define a tensor regression model to compute the probability for each text X:

$$\begin{aligned}p(X) &= \langle \mathcal{A}, \Phi(X) \rangle \\ &= \sum_{i_1, i_2, \cdots, i_N = 1}^{|V|} \mathcal{A}_{i_1, \cdots, i_N} \cdot \Phi(X)_{i_1, \cdots, i_N}\end{aligned} \tag{2}$$

where $\langle \cdot \rangle$ denotes the inner product of two same-sized tensors, and $\mathcal{A}$ is a regression weight tensor of the same shape as $\Phi(X)$ in the tensor space $\mathbb{V}^{\otimes N} = \underbrace{\mathbb{V} \otimes \cdots \otimes \mathbb{V}}_{N}$ where $\mathbb{V}$ refers to $\mathbb{R}^{|V|}$. Similar functions were considered in (Novikov et al., 2016; Stoudenmire & Schwab, 2016; Khrulkov et al., 2018; Zhang et al., 2019).

4.2. Tensor Train Language Model

4.2.1. Tensor-Train Decomposition

Suppose the sequence of indices of words in the text X is $w_1, w_2, \cdots, w_N$, where $w_i \in \{1, 2, \cdots, |V|\}$ and its corresponding weight in $\mathcal{A}$ is denoted as $\mathcal{A}_{w_1 w_2 \cdots w_N}$. We use TT decomposition to represent $\mathcal{A}_{w_1 w_2 \cdots w_N}$ in the TT format (Oseledets, 2011) as follows:

$$\begin{aligned}\mathcal{A}_{w_1 w_2 \ldots w_N} &= \underbrace{\mathbf{G}^{(1)}_{:,w_1}}_{1 \times R_1} \underbrace{\mathbf{G}^{(2)}_{:,w_2,:}}_{R_1 \times R_2} \cdots \underbrace{\mathbf{G}^{(N)}_{:,w_N}}_{R_{N-1} \times 1} \\ &= \sum_{\alpha_1, \cdots, \alpha_{N-1}} G^{(1)}_{w_1 \alpha_1} G^{(2)}_{\alpha_1 w_2 \alpha_2} \cdots G^{(N)}_{\alpha_{N-1} w_N}\end{aligned} \tag{3}$$

where the tensors $\mathbf{G}^{(t)} \in \mathbb{R}^{R_{t-1} \times |V| \times R_t}$ ($t = 1, ..., d$, $R_0 = R_N = 1$ by definition) are called *TT cores*, and R_k for $k = 1, \cdots, N$ are called *TT ranks*.

Despite its site-dependent TT cores $\mathbf{G}^{(t)}$ potentially giving it more expressiveness for language modeling, this property currently generates unnecessary obstacles to its applicability, like the choice of R_t. Here we follow the convention of considering a special class of TT decompositions (Khrulkov et al., 2018; Miller et al., 2021), i.e. supposing all the intermediate TT cores are equal to each other $\mathbf{G} = \mathbf{G}^{(2)}, \ldots, \mathbf{G}^{(N-1)} \in \mathbb{R}^{R \times |V| \times R}$ and $\mathbf{G}^{(1)} = \mathbf{G}^{(N)} \in \mathbb{R}^{|V| \times R}$ in Eq. 3.

4.2.2. Definition of TTLM

We define *Tensor Train Language Model* (TTLM) as:

$$\begin{aligned}p(X) = \sum_{i_1, \cdots, i_N = 1}^{|V|} \sum_{\alpha_1, \cdots, \alpha_{N-1} = 1}^{R} & f(x^{(1)})_{i_1} G^{(1)}_{i_1 \alpha_1} \cdots \\ & f(x^{(N)})_{i_N} G^{(N)}_{\alpha_{N-1} i_N}\end{aligned} \tag{4}$$

where each $\boldsymbol{f}(x^{(t)})$ is a one-hot vector having $w_t = 1$ for at most one t, and has zeros elsewhere. The tensor diagram notation of TTLM is shown in Fig. 2a. Note that Eq. 4 can compute the elements of $\mathcal{A}$ in the low-dimensional space

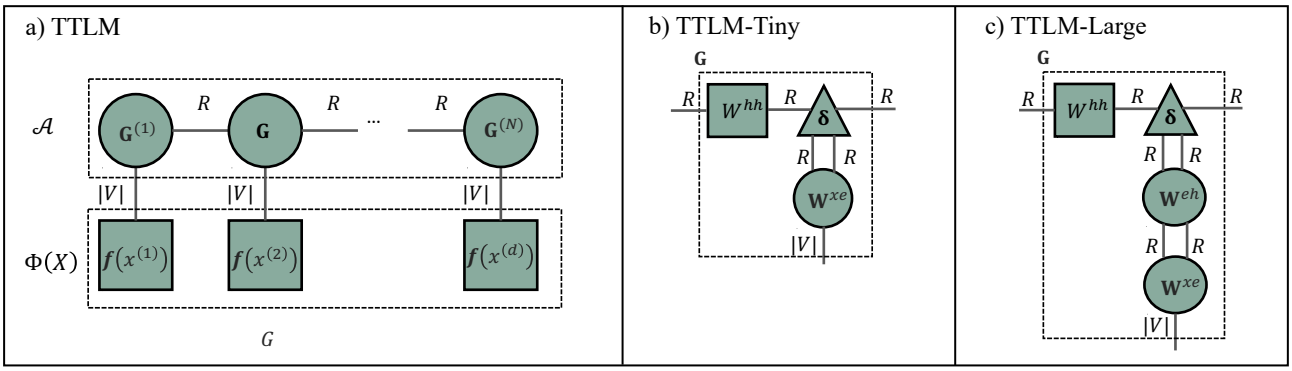

Figure 2. a) Tensor Train Language Model based on Eq. 4. b) TT core of TTLM-Tiny. c) TT core of TTLM-Large. The dashed line in the square represents $\mathcal{A}$, $\Phi(X)$, or $\mathbf{G}$. Note that the only difference between TTLM-Large and TTLM-Tiny is whether to use tensor $\mathbf{W}^{eh}$.

as Eq. 3 does. This can be observed if we represent the elements of $\mathbf{G}^{(t)}_{:,w_t,:}$ in Eq. 3 as:

$$G^{(t)}_{\alpha_{t-1}w_t\alpha_t} = \sum_{i=1}^{|V|} f(x^{(t)})_i G^{(t)}_{\alpha_{t-1}i\alpha_t} \tag{5}$$

Since $\mathcal{A}_{w_1w_2\ldots w_N}$ here equals to $p(X)$, we can derive Eq. 4 by inserting Eq. 5 into Eq. 3.

The critical difference between TTLM defined by Eq. 4 and Eq. 3 is that TTLM has combined the weights $\mathcal{A}$ and the input data $\Phi(X)$ together, indicating its potential to be used for language modeling tasks.

4.2.3. Recursive Probability Computation.

We recursively unfold the calculation of TTLM in Eq. 4 and find that $\mathbf{G}$ has two sources of "input": the information from the previous recursive unfolding, and the input data $\boldsymbol{f}(x^{(t)})$ (see Eq. 13 for a detailed version). From this perspective, $\mathbf{G}$ acts as a bilinear map $\mathbf{G} : \mathbb{R}^{|V|} \times \mathbb{R}^R \to \mathbb{R}^R$, and we can regard the information in the previous step as a hidden state $\boldsymbol{h}^{(t)}_{\text{TTLM}}$, given by:

$$\boldsymbol{h}^{(t)}_{\text{TTLM}} = \boldsymbol{f}(x^{(t)})^T \mathbf{G} \boldsymbol{h}^{(t-1)}_{\text{TTLM}} \tag{6}$$

where $\boldsymbol{f}(x^{(t)})$, $\mathbf{G}$, and $\boldsymbol{h}^{(t-1)}_{\text{TTLM}}$ are contracted together (we permute the indices of $\mathbf{G}$ from $\mathbb{R}^{R\times|V|\times R}$ to $\mathbb{R}^{|V|\times R\times R}$ which does not change the number of indices).

Utilizing this recursive property, we here provide further details about computing $p(X)$ by TTLM in practice. In language modeling, $p(X)$ is often decomposed using the chain rule (Bahl et al., 1983) as follows:

$$p(X) = \prod_{t=1}^{N} p(x^{(t)}|x^{(1:t-1)})$$

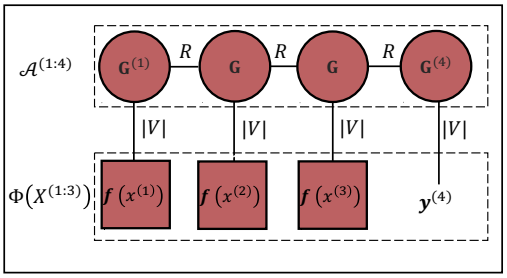

Figure 3. Recursive calculation of conditional probability in TTLM. Here we provide an example that given the text $x^{(1:3)}$, $\boldsymbol{y}^{(4)} = \psi(\mathbf{G}^{(4)}\boldsymbol{h}^{(3)}_{\text{TTLM}})$ where $\boldsymbol{y}^{(4)}$ is the probability distribution of word $x^{(4)}$.

where $x^{(1:t-1)}$ denotes the text $[x^{(1)}, x^{(2)}, \cdots, x^{(t-1)}]$. At time t, the output prediction of a model, $\boldsymbol{y}^{(t)} \in \mathbb{V}$, is a probability distribution of word $x^{(t)}$ given $x^{(1:t-1)}$.

In TTLM, we define $\boldsymbol{y}^{(t)}$ as follows:

$$\boldsymbol{y}^{(t)} = \psi\left(\mathbf{G}^{(t)}\boldsymbol{h}^{(t-1)}_{\text{TTLM}}\right) \tag{7}$$

where $\mathbf{G}^{(t)} \in \mathbb{R}^{|V|\times R}$ is the last TT core in TT format at time t. ψ is any function that ensures that $\boldsymbol{y}^{(t)}$ is non-negative and that the conditional probabilities sum to 1. For the use of TTLM 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 paper, we shall use the $\mathrm{softmax}$ function (Bridle, 1990). Fig. 3 provides an example of a recursive calculation of conditional probability.

We can derive the definition of $\boldsymbol{y}^{(t)}$ in high-dimensional

space, if we substitute $\boldsymbol{h}_{\text{TTLM}}^{(t-1)}$ in Eq. 7 by Eq. 4 and Eq. 6:

$$\boldsymbol{y}^{(t)} = \psi\left(\sum_{i_1,\cdots,i_{t-1}}\sum_{\alpha_1,\cdots,\alpha_{t-1}} f(x^{(1)})_{i_1}\boldsymbol{G}^{(1)}_{i_1\alpha_1}\cdots\mathbf{G}^{(t)}_{\alpha_{t-1}}\right) \tag{8}$$

$$= \psi\left(\langle\mathcal{A}^{(1:t)}), \Phi(X^{(1:t-1)})\rangle_{t-1}\right) \tag{9}$$

where $\mathcal{A}^{(1:t)} \in \mathbb{V}^{\otimes t}$, $\Phi(X^{(1:t-1)}) = \bigotimes_{i=1}^{t-1} \boldsymbol{f}(x^{(i)}) \in \mathbb{V}^{\otimes t-1}$ and $\langle\cdot\rangle_{t-1}$ denotes the "generalized inner product" defined in Sec. 3. Note that Eq. 8 is the low-dimensional form of Eq. 9, similarly to the relationship between Eq. 4 and Eq. 2.

By these definitions, there are some interesting properties of TTLM. (1) We can use *teacher forcing* (Marcus, 1998) to learn parameters of TT cores. (2) The hidden-to-output tensor $\mathbf{G}^{(t)}$ is defined to be the same as the input-to-hidden tensor $\mathbf{G}^{(1)}$. (3) $\mathbf{G}$ and $\mathbf{G}^{(t)}$ have no parameters in common. We provide a detailed explanation of the relationship between different TT cores in Appendix A.

5. TTLM Variants

To show the versatility and practical applicability of the TTLM framework, we now propose two new variants: TTLM-Large and TTLM-Tiny in Sec. 5.1. We briefly summarize the relationship between TTLM and some widely-used RNNs in Sec. 5.2.

5.1. New Variants: TTLM-Large and TTLM-Tiny

The TT core $\mathbf{G}$ in TTLM is an entire third-order tensor. In the two variants, we decompose $\mathbf{G}$ into several separate tensors without violating the TT format, as shown in Fig. 2b and Fig. 2c. We define TTLM-Tiny and TTLM-Large as follows:

$$\begin{aligned}\boldsymbol{h}_{\text{Tiny}}^{(t)} &= \boldsymbol{f}(x^{(t)})^T\mathbf{W}^{xe}\boldsymbol{\delta}\boldsymbol{W}^{hh}\boldsymbol{h}_{\text{Tiny}}^{(t-1)}\\ \boldsymbol{h}_{\text{Large}}^{(t)} &= \boldsymbol{f}(x^{(t)})^T\mathbf{W}^{xe}\mathbf{W}^{eh}\boldsymbol{\delta}\boldsymbol{W}^{hh}\boldsymbol{h}_{\text{Large}}^{(t-1)}\end{aligned} \tag{10}$$

where $\boldsymbol{W}^{hh} \in \mathbb{R}^{R\times R}$ is the hidden-to-hidden matrix; $\mathbf{W}^{xe} \in \mathbb{R}^{|V|\times R\times R}$ is the input-to-hidden tensor; $\mathbf{W}^{eh} \in \mathbb{R}^{R\times R\times R\times R}$; and $\boldsymbol{\delta} \in \mathbb{R}^{R\times R\times R\times R}$ is a fourth-order diagonal tensor such that $\delta_{ijkl} = 1$ iff the $i = j = k = l$, and $\delta_{ijkl} = 0$ otherwise.

The relationship between our proposed models and TTLM is as follows: $\mathbf{W}^{xe}$ in both models take the same role as $\mathbf{G}^{(t)}$ in TTLM (i.e. input-to-hidden and hidden-to-output), while $\mathbf{G} = \mathbf{W}^{xe}\boldsymbol{\delta}\boldsymbol{W}^{hh}$ in TTLM-Tiny and $\mathbf{G} = \mathbf{W}^{xe}\mathbf{W}^{eh}\boldsymbol{\delta}\boldsymbol{W}^{hh}$ in TTLM-Large.

As in RNNs, we compute the conditional probability recursively for TTLM-Large and TTLM-Tiny as:

$$\boldsymbol{y}^{(t)} = \psi(\mathbf{V}\mathbf{P}\boldsymbol{h}^{(t)}) \tag{11}$$

where $\mathbf{V} \in \mathbb{R}^{R\times|V|\times R}$ is an output embedding tensor, $\mathbf{P} \in \mathbb{R}^{R\times R\times R}$ is a projector tensor. Then we tie the input tensor $\mathbf{W}^{xe}$ to the output embedding tensor $\mathbf{V}$ (we provide a detailed explanation in Sec. 6.2).

One obvious advantage of our models is to utilize information from the hidden layer and input data separately. Such interaction, particularly TTLM-Tiny, can potentially avoid overfitting, similarly to (Wu et al., 2016) where *multiplication integration* between two sources of "input" can outperform many other methods. In Sec 6.3, we provide relevant experimental evidence.

5.2. Existing TTLM Variants

Given the fact that TT scores of TTLM can vary, Appendix B provides a detailed illustration that three existing models, namely Second-order RNNs, Recurrent Arithmetic Circuits (RACs), and Multiplicative Integration RNNs (MI-RNNs) can be considered as one of the "special" implementations of TTLM.

We briefly summarize the differences between the three models: 1) Second-order RNNs use the third-order $\mathbf{T}$ as the TT cores with an activation function given Eq. 12; 2) RACs use $\boldsymbol{W}^{hx} \odot \boldsymbol{W}^{hh}$ as the TT cores given Eq. 16; 3) MI-RNNs use $\boldsymbol{W}^{hx} \odot \boldsymbol{W}^{hh}$ as the TT cores with an activation function given Eq. 17.

Along with our two proposed models, we study the experimental performance of second-order RNNs, RACs and MI-RNNs compared to TTLM-Large and TTLM-Tiny in Sec. 6.

6. Experimental Evaluation

To further understand the properties of TTLM variants, we now investigate the effectiveness of TTLM, TTLM-Large and TTLM-Tiny compared to Second-order RNNs, RACs, MI-RNNs, and Vanilla-RNNs.

We specify our experimental setting in Sec. 6.1 and implementation details in Sec. 6.2. We study the influence of ranks on the performance of TTLM variants in Sec 6.3 and examine the impact of nonlinear activation functions on the effectiveness of TTLM variants in Sec.6.4.

6.1. Experimental Setting

Task, Datasets, and Metric. We conduct experiments on two word-level language model datasets: **(1)** English Penn

Model	WikiText-2 Param	WikiText-2 PPL	PTB Param	PTB PPL	Hidden (Rank)	Layer	Embed Size
Transformer (Vaswani et al., 2017)	90.5M	293.0	32.8M	208.7	20	1	400
Vanilla-RNNs (Mikolov & Zweig, 2012)	11.6M	96.6	4.0M	115.3	20	1	400
Second-order RNNs (Hochreiter & Schmidhuber, 1997)	11.8M	96.0	4.2M	108.2	20	1	400
RACs (Levine et al., 2018)	11.6M	97.6	4.0M	116.8	20	1	400
MI-RNNs (Wu et al., 2016)	11.6M	99.6	4.0M	119.1	20	1	400
TTLM	12.2M	546.4	4.2M	559.8	20	1	400
TTLM-Tiny	11.6M	94.9	4.0M	106.8	20	1	400
TTLM-Large	11.8M	**82.3**	4.2M	**99.3**	20	1	400

Table 1. Test set PPL on the WikiText-2 and PTB datasets. The symbol "−" means these data are not available in their original paper. The "Param" column denotes the number of parameters; see Sec. 6.2 for a detailed description. The "Hidden (Rank)" column denotes the number of hidden units or ranks. The "Embed Size" column denotes the size of each embedding vector. We report the lowest test set PPL of the Transformer whose number of heads is selected from [2, 4, 5, 8].

Treebank (PTB) (Marcinkiewicz, 1994), which consists of 929k training tokens, 73k validation tokens, and 82k test tokens. Its vocabulary size is 10k. **(2)** The WikiText-2 dataset (Merity et al., 2016) is derived from Wikipedia articles and consists of 2088k training tokens, 217k validation tokens, 45k test tokens, and a vocabulary of over 30k types. We compare these models on the language modeling task, evaluated by the Perplexity (PPL) (Meister & Cotterell, 2021); the lower the PPL, the better the model.

Baselines. Our models are compared with the following baselines: Transformer (Vaswani et al., 2017) , Vanilla-RNNs (Mikolov & Zweig, 2012), Second-order RNNs (Hochreiter & Schmidhuber, 1997), Recurrent Arithmetic Circuits (RACs) (Levine et al., 2018), and Multiplicative Integration RNNs (MI-RNNs) (Wu et al., 2016). The implementation details are provided in Sec. 6.2.

Hyperparameters. **(1)** To compare the effectiveness of comparable models on the same scale, we set the rank/hidden units of TTLM variants/Vanilla-RNNs as [5, 10, 20, 25, 30, 35, 40, 45, 50]. The embedding size of these models is the squared number of hidden units/ranks. This setup is because of the architectures of TTLM-Large and TTLM-Tiny as introduced in Sec. 5.1. **(2)** To avoid the potential impact of the embedding size on Vanilla-RNNs' performance, we provide several common choices of embedding size in the model by setting its embedding size as [100, 200, 300]. We name them as RNNs-100, RNNs-200, and RNNs-300 correspondingly and display them in Fig. 4. **(3)** We train all models for 50 epochs and choose the best model in the validation set to predict the result in the test set. **(4)** The weights in the models are adjusted to minimize the average cross entropy loss over training sequences via stochastic gradient descent computed using the truncated backpropagation through time algorithm (Werbos, 1990; Williams & Peng, 1990). The random seed is fixed to ensure the experimental results are not influenced by initializing the weights.

6.2. Implementations

Model	Training Parameters
Vanilla-RNN	$\boldsymbol{W}^{xe} \in \mathbb{R}^{E\times\|V\|}$, $\boldsymbol{W}^{eh} \in \mathbb{R}^{E\times H}$, $\boldsymbol{W}^{hh} \in \mathbb{R}^{H\times H}$, $\boldsymbol{P} \in \mathbb{R}^{H\times E}$, $\boldsymbol{V} \in \mathbb{R}^{E\times\|V\|}$
MI-RNNs	$\boldsymbol{W}^{xe} \in \mathbb{R}^{E\times\|V\|}$, $\boldsymbol{W}^{eh} \in \mathbb{R}^{E\times H}$, $\boldsymbol{W}^{hh} \in \mathbb{R}^{H\times H}$, $\boldsymbol{P} \in \mathbb{R}^{H\times E}$, $\boldsymbol{V} \in \mathbb{R}^{E\times\|V\|}$
RACs	$\boldsymbol{W}^{xe} \in \mathbb{R}^{E\times\|V\|}$, $\boldsymbol{W}^{eh} \in \mathbb{R}^{E\times H}$, $\boldsymbol{W}^{hh} \in \mathbb{R}^{H\times H}$, $\boldsymbol{P} \in \mathbb{R}^{H\times E}$, $\boldsymbol{V} \in \mathbb{R}^{E\times\|V\|}$
Second-order RNNs	$\boldsymbol{W}^{xe} \in \mathbb{R}^{E\times\|V\|}$, $\mathbf{T} \in \mathbb{R}^{H\times H\times H}$, $\boldsymbol{W}^{hh} \in \mathbb{R}^{E\times H}$, $\boldsymbol{P} \in \mathbb{R}^{H\times E}$, $\boldsymbol{V} \in \mathbb{R}^{E\times\|V\|}$
TTLM	$\mathbf{G} \in \mathbb{R}^{R\times\|V\|\times R}$, $\boldsymbol{G}^{(t)} \in \mathbb{R}^{R\times\|V\|}$, $\boldsymbol{G}^{(1)} \in \mathbb{R}^{R\times\|V\|}$
TTLM-Tiny	$\mathbf{W}^{xe} \in \mathbb{R}^{R\times\|V\|\times R}$, $\boldsymbol{W}^{hh} \in \mathbb{R}^{R\times R}$, $\mathbf{P} \in \mathbb{R}^{R\times R\times R}$, $\mathbf{V} \in \mathbb{R}^{R\times R\times\|V\|}$
TTLM-Large	$\mathbf{W}^{xe} \in \mathbb{R}^{R\times\|V\|\times R}$, $\mathbf{W}^{eh} \in \mathbb{R}^{R\times R\times R\times R}$, $\boldsymbol{W}^{hh} \in \mathbb{R}^{R\times R}$, $\mathbf{P} \in \mathbb{R}^{R\times R\times R}$, $\mathbf{V} \in \mathbb{R}^{R\times R\times\|V\|}$

Table 2. Training parameters in our implementation. E is the embedding size, H is the hidden units in RNNs, and R is the rank in the TTLM. We set $H = R$ and $E = R^2$ to make the parameters of all models in the same scale. The parameters of $\boldsymbol{W}^{xe}$ and $\mathbf{W}^{xe}$ are uniformly initialized in the interval $[-0.1, 0.1]$, $\boldsymbol{W}^{eh}$, $\mathbf{W}^{eh}$ and $\boldsymbol{W}^{hh}$ are uniformly initialized between $[-\frac{1}{\sqrt{H}}, \frac{1}{\sqrt{H}}]$.

We implement all models using PyTorch on GPU A100 with one single card.

We use the PyTorch version of the standard Transformer (Vaswani et al., 2017).[4] For RNNs, there are five matrix parameters: $\boldsymbol{W}^{xe} \in \mathbb{R}^{E\times|V|}$ is the input embedding matrix,

[4]The code is available at https://pytorch.org/tutorials/beginner/transformer_tutorial.html.

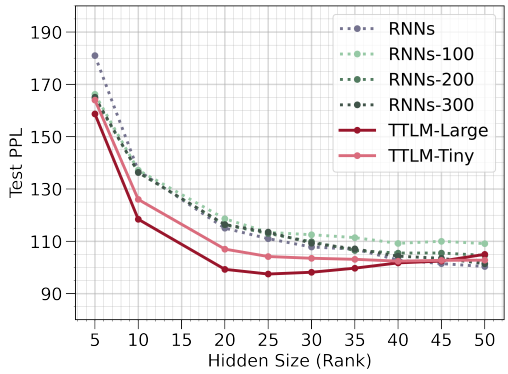

Figure 4. Test set PPL on the PTB dataset w.r.t. ranks/hidden units. RNNs here denotes Vanilla-RNNs, which has the same embedding size as TTLM-Large and TTLM-Tiny. RNNs-100, RNNs-200, and RNNs-300 are the Vanilla-RNNs with fixed embedding sizes of 100, 200, and 300, respectively.

$\boldsymbol{W}^{eh} \in \mathbb{R}^{E \times H}$ is the embedding-to-hidden matrix, $\boldsymbol{W}^{hh} \in \mathbb{R}^{H \times H}$ is the hidden-to-hidden matrix. We tie (share the same training parameters) the input embedding $\boldsymbol{W}^{xe}$ and output embedding $\boldsymbol{V}$, which has been proved to lead to a significant reduction in perplexity (Press & Wolf, 2016). So there is a projection matrix $\boldsymbol{P} \in \mathbb{R}^{H \times E}$ before the output embedding. All this process is introduced in (Press & Wolf, 2016).

For TTLM models, we tie the input tensor $\mathbf{W}^{xe}$ and $\mathbf{V}$. The implementation of δ is functioned by a reshape function, so the interaction between hidden and input can be computed by matrix product. We also let $\mathbf{G}^{(1)}$ have the same parameters along the dimension $|V|$ (i.e., $\mathbf{G}^{(1)}$ is simplified into a $\boldsymbol{G}^{(1)} \in \mathbb{R}^{1 \times R}$ and thus it can be viewed as the initial hidden state).

6.3. Rank and Effectiveness Analysis

The rank of the TT format has been used to explain the expressive power or long-term memory capacity of RNNs (Khrulkov et al., 2018; Levine et al., 2018). The *rank* of TT decomposition has been proved to be the dimension of the hidden states of RNNs (Khrulkov et al., 2018), which reflect on the capacity of the RNNs. The higher rank can have more capacity and vice versa. However, the relationship between rank and effectiveness in language modeling has yet to be shown practically. We will evaluate the effectiveness of TTLM-Large and TTLM-Tiny w.r.t ranks.

6.3.1. Effectiveness

Table 1 presents the results of the test set PPL for our models and the baselines on the WikiText-2 and PTB datasets. As shown, compared to Vanilla-RNNs, TTLM-Large reduces PPL by 14.3 and 16.0, respectively, and TTLM-Tiny reduces PPL by 1.7 and 8.5, respectively. Thus, when the number of hidden units or ranks is set to 20 and the embedding size is 400, both TTLM-Large and TTLM-Tiny perform better than all the baselines.

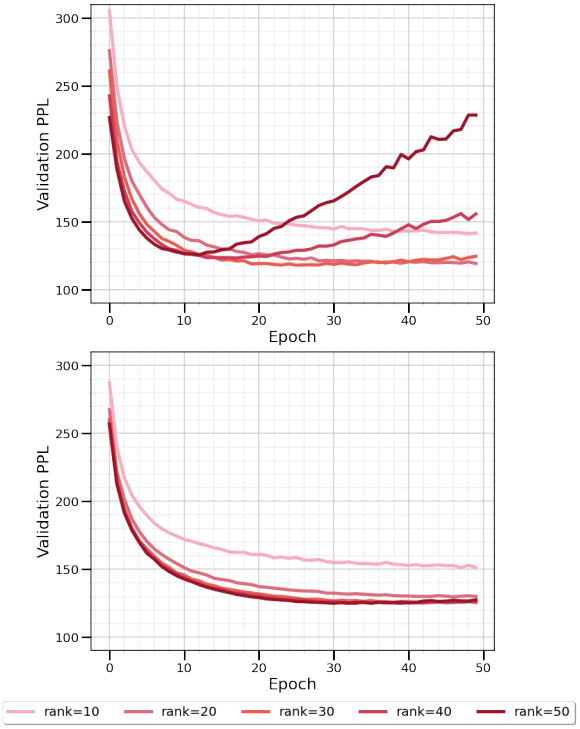

Figure 5. Validation set PPL of TTLM-Large and TTLM-Tiny with increasing ranks on the PTB dataset. Top: TTLM-Large, Bottom: TTLM-Tiny.

To further evaluate the effectiveness of our models, we conduct a comparison between TTLM-Large, TTLM-Tiny, and Vanilla-RNNs using increasing ranks, as depicted in Fig. 4. It's worth noting that we also include RNNs-100, RNNs-200, and RNNs-300 to control for the potential impact of large embedding size. As shown, even when the number of hidden units reaches 40, the test set PPL of RNNs decreases steadily, while TTLM-Large and TTLM-Tiny do not. Thus, we expect our models to outperform Vanilla-RNNs with a *low-scale* of hidden units (i.e., the number ranges from 5 to 40), but not larger scales.

6.3.2. Overfitting

Fig. 5 illustrates the performance of TTLM-Large and TTLM-Tiny on the validation set as the number of ranks increases. As shown, the validation set PPL of TTLM-Large starts to rise in earlier training epochs when we gradually enlarge its ranks. In contrast, the validation PPL of TTLM-Tiny stably decreases as the number of ranks increases. The comparison indicates that TTLM-Large is more prone to overfitting than TTLM-Tiny. This finding is further sup-

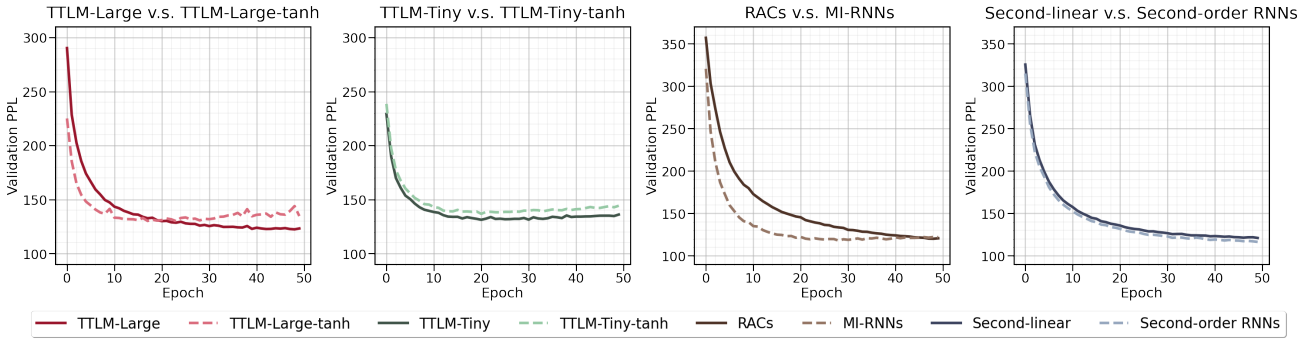

Figure 6. The influence of nonlinearity on TTLM variants on the PTB dataset. The suffix `-tanh` refers to a model using the $tanh$ activation function, indicated by dashed lines. Second-linear refers to Second-order RNNs without activation functions. Setting: all models have 17 hidden units/ranks.

ported by the results in Fig. 4. The test set PPL of TTLM-Tiny consistently improves as the rank increases, while TTLM-Large's performance declines when the number of ranks reaches 25.

When we focus on the difference between the two models, TTLM-Large has an additional parameter tensor $\mathbf{W}^{eh}$. Thus, we believe that the simpler parameterization of the TT cores, the more easily the model avoids overfitting. This finding is consistent with the comparison between MI-RNNs and Second-order RNNs by (Wu et al., 2016). When it comes to practical situations, we need to be aware that TTLM-Tiny has a lower capacity to fit the training data and, as a result, poses a lower risk of overfitting when compared to TTLM-Large.

6.4. Nonlinearity Analysis

Previous studies have attempted to use TT decomposition as a theoretical platform to investigate RNNs (Khrulkov et al., 2018; Levine et al., 2018). However, one key difference between TT decomposition and the existing neural networks (like Vanilla-RNNs) is the nonlinearity activation functions inside the network models. The lack of nonlinearity in the tensor decomposition calls into question whether its theoretical analysis is transferable to models based on RNNs. To understand whether the effect of the $tanh$ activation function on the TT variants varies with the TT cores, we provide an empirical result as displayed in Fig. 6.

Regarding convergence speed, $\tanh$ speeds up TTLM-Large-tanh, TTLM-Tiny-tanh, and MI-RNNs while barely influencing second-order RNNs. Regarding the magnitude of the lowest validation perplexity, $\tanh$ impairs the performance TTLM-Large and TTLM-Tiny but has little influence on multiplicative integration and the third-order tensor $\mathbf{T}$ in Second-order RNNs.

Thus, the influence of nonlinear activation functions on TTLM variants depends on TT cores settings, both for the convergence of validation PPL and the magnitude of the lowest validation PPL. From an experimental point of view, we believe that the effect of nonlinearity functions on one TT variant cannot simply be transferred or analogized to another TT variant. This also suggests that one should be wary of the analogy between tensor decomposition and existing neural network models at the implementation level declared by previous research (Khrulkov et al., 2018; Levine et al., 2018). The nonlinear activation functions could be a factor influencing such an analogy.

7. Conclusion

Tensor networks having been proposed as promising language models, we first apply TT decomposition to real-world language modeling datasets and name the framework TTLM. We propose two variants: TTLM-Large and TTLM-Tiny, and show that they outperform Vanilla-RNNs with low-scale hidden units. The presentation of the experimental results is an advancement for exploring tensor networks in machine learning. Meanwhile, we demonstrate that Second-order RNNs, RACs, and MI-RNNs are special implementations of TTLM.

A limitation of this study is that it shall examine the influence of different normalization functions. In future research, if appropriate mathematical tools and benchmarks are available, we plan to investigate the long-range correlation modeling capability of TTLM in natural language, which is believed to be one of the core features in TTLM.

References

Alexander, R. N., Evenbly, G., and Klich, I. Exact holographic tensor networks for the motzkin spin chain. *Quantum*, 5:546, 2021.

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

Baldi, P., Cranmer, K., Faucett, T., Sadowski, P., and Whiteson, D. Parameterized machine learning for high-energy physics. *arXiv preprint arXiv:1601.07913*, 2016.

Bi, Y., Lu, Y., Long, Z., Zhu, C., and Liu, Y. Chapter 1 - tensor decompositions: computations, applications, and challenges. In Liu, Y. (ed.), *Tensors for Data Processing*, pp. 1–30. Academic Press, 2022.

Bridle, J. S. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pp. 227–236. Springer, 1990.

Cohen, N. and Shashua, A. Convolutional rectifier networks as generalized tensor decompositions. In *International Conference on Machine Learning*, pp. 955–963. PMLR, 2016.

Cohen, N., Sharir, O., and Shashua, A. On the expressive power of deep learning: A tensor analysis. In *Conference on learning theory*, pp. 698–728. PMLR, 2016.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption.

Fonollosa, J., Sheik, S., Huerta, R., and Marco, S. Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. *Sensors and Actuators B: Chemical*, 215:618–629, 2015.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT Press, 2016.

Goudreau, M. W., Giles, C. L., Chakradhar, S. T., and Chen, D. First-order versus second-order single-layer recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(3):511–513, 1994.

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

Hou, Y., Zhao, X., Song, D., and Li, W. Mining pure high-order word associations via information geometry for information retrieval. *ACM Transactions on Information Systems (TOIS)*, 31(3):1–32, 2013.

Itskov, M. *Tensor Algebra and Tensor Analysis for Engineers: With Applications to Continuum Mechanics*. Springer Publishing Company, Incorporated, 2nd edition, 2009. ISBN 3540939067.

Khrulkov, V., Novikov, A., and Oseledets, I. Expressive power of recurrent neural networks. In *International Conference on Learning Representations*, 2018.

Kossaifi, J., Lipton, Z. C., Kolbeinsson, A., Khanna, A., Furlanello, T., and Anandkumar, A. Tensor regression networks. *The Journal of Machine Learning Research*, 21(1):4862–4882, 2020.

Levine, Y., Sharir, O., and Shashua, A. Benefits of depth for long-term memory of recurrent networks. 2018.

Lin, H. W. and Tegmark, M. Critical behavior from deep dynamics: a hidden dimension in natural language. *arXiv preprint arXiv:1606.06737*, 2016.

Marcinkiewicz, M. A. Building a large annotated corpus of english: The penn treebank. *Using Large Corpora*, 273, 1994.

Marcus, G. F. Rethinking eliminative connectionism. *Cognitive psychology*, 37(3):243–282, 1998.

Martin, D., Fowlkes, C., Tal, D., and Malik, J. 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*, volume 2, pp. 416–423. IEEE, 2001.

Maupomé, D. and Meurs, M.-J. Language modeling with a general second-order rnn. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pp. 4749–4753, 2020.

Meister, C. and Cotterell, R. 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)*, pp. 5328–5339, Online, August 2021. Association for Computational Linguistics.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Mikolov, T. and Zweig, G. Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pp. 234–239. IEEE, 2012.

Miller, J., Rabusseau, G., and Terilla, J. Tensor networks for probabilistic sequence modeling. In *International Conference on Artificial Intelligence and Statistics*, pp. 3079–3087. PMLR, 2021.

Mora, T. and Bialek, W. Are biological systems poised at criticality? *Journal of Statistical Physics*, 144(2):268–302, 2011.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. Tensorizing neural networks. *Advances in neural information processing systems*, 28, 2015.

Novikov, A., Trofimov, M., and Oseledets, I. Exponential machines. *arXiv preprint arXiv:1605.03795*, 2016.

Novikov, G. S., Panov, M. E., and Oseledets, I. V. Tensor-train density estimation. In *Uncertainty in Artificial Intelligence*, pp. 1321–1331. PMLR, 2021.

Oseledets, I. V. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Pestun, V. and Vlassopoulos, Y. Tensor network language model. *arXiv preprint arXiv:1710.10248*, 2017.

Pestun, V., Terilla, J., and Vlassopoulos, Y. Language as a matrix product state. *arXiv preprint arXiv:1711.01416*, 2017.

Press, O. and Wolf, L. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

Rabusseau, G., Li, T., and Precup, D. Connecting weighted automata and recurrent neural networks through spectral learning. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1630–1639. PMLR, 2019.

Radev, D. Clair collection of fraud email, acl data and code repository. *ADCR2008T001*, 2008.

Rendle, S. Factorization machines. In *2010 IEEE International conference on data mining*, pp. 995–1000. IEEE, 2010.

Roe, B. P., Yang, H.-J., Zhu, J., Liu, Y., Stancu, I., and McGregor, G. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 543(2-3):577–584, 2005.

Stoudenmire, E. and Schwab, D. J. Supervised learning with tensor networks. *Advances in Neural Information Processing Systems*, 29, 2016.

Sutskever, I., Martens, J., and Hinton, G. E. Generating text with recurrent neural networks. In *ICML*, 2011.

Tagliazucchi, E., Balenzuela, P., Fraiman, D., and Chialvo, D. R. Criticality in large-scale brain fmri dynamics unveiled by a novel point process analysis. *Frontiers in physiology*, 3:15, 2012.

Tomita, M. 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, 1982.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Williams, R. J. and Peng, J. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

Wu, Y., Zhang, S., Zhang, Y., Bengio, Y., and Salakhutdinov, R. R. On multiplicative integration with recurrent neural networks. *Advances in neural information processing systems*, 29, 2016.

Zhang, L., Zhang, P., Ma, X., Gu, S., Su, Z., and Song, D. A generalized language model in tensor space. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 7450–7458, 2019.

A. Relationship between TT Cores in TTLM

To help readers understand the roles of TT cores in TTLM, we here provide a detailed calculation of the probability of a text $X = [x^{(1)}, x^{(2)}, \cdots, x^{(N)}]$ by TTLM. Note that all the intermediate TT cores are equal to each other: $\mathbf{G} = \mathbf{G}^{(2)}, ..., \mathbf{G}^{(N-1)}$ and $\mathbf{G}^{(1)} = \mathbf{G}^{(N)}$.

The calculation of $\boldsymbol{y}^{(t)}$ (i.e. the conditional probability of $x^{(t)}$ given $x^{(1:t-1)}$) at time t) can be described as three steps. As step I, suppose $\boldsymbol{f}(x^{(1)})$ is a one-hot vector having $f(x^{(1)})_1 = 1$. The calculation of $\mathbf{G}^{(1)}\boldsymbol{f}(x^{(1)}$ in TTLM is as follows:

$$\begin{aligned}\mathbf{G}^{(1)}\boldsymbol{f}(x^{(1)} &= \begin{bmatrix} f\left(x^{(1)}\right)_1 \\ f\left(x^{(1)}\right)_2 \\ \cdots \\ f\left(x^{(1)}\right)_{|V|} \end{bmatrix} \begin{bmatrix} \mathbf{G}^{(1)}_{11} & \mathbf{G}^{(1)}_{12} & \cdots & \mathbf{G}^{(1)}_{1R} \\ \mathbf{G}^{(1)}_{21} & \mathbf{G}^{(1)}_{22} & \cdots & \mathbf{G}^{(1)}_{2R} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{G}^{(1)}_{|V|1} & \mathbf{G}^{(1)}_{|V|2} & \cdots & \mathbf{G}^{(1)}_{|V|R} \end{bmatrix} \\ &= \left[\mathbf{G}^{(1)}_{11}, \mathbf{G}^{(1)}_{12}, \cdots, \mathbf{G}^{(1)}_{1R}\right]^T \\ &= \boldsymbol{h}^{(1)}_{\text{TTLM}}\end{aligned}$$

As step II, TTLM will calculate $\boldsymbol{f}(x^{(i)})\mathbf{G}\boldsymbol{h}^{(i-1)}_{\text{TTLM}}$ where $i \in \{2, 3, \cdots, t-1\}$. For example, $\boldsymbol{h}^{(2)}_{\text{TTLM}}$ is calculated in Eq. 6 at time $t = 2$ as follows:

$$\boldsymbol{h}^{(2)}_{\text{TTLM}} = \boldsymbol{f}(x^{(2)})^T\mathbf{G}\boldsymbol{h}^{(1)}_{\text{TTLM}}$$

As step III, TTLM will output $\boldsymbol{y}^{(t)}$ as follows:

$$\begin{aligned}\mathbf{G}^{(t)}\boldsymbol{h}^{(t-1)}_{\text{TTLM}} &= \begin{bmatrix} \mathbf{G}^{(t)}_{11} & \mathbf{G}^{(t)}_{12} & \cdots & \mathbf{G}^{(t)}_{1R} \\ \mathbf{G}^{(t)}_{21} & \mathbf{G}^{(t)}_{22} & \cdots & \mathbf{G}^{(t)}_{2R} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{G}^{(t)}_{|V|1} & \mathbf{G}^{(t)}_{|V|2} & \cdots & \mathbf{G}^{(t)}_{|V|R} \end{bmatrix} \begin{bmatrix} h^{(t-1)}_{\text{TTLM}_1} \\ h^{(t-1)}_{\text{TTLM}_2} \\ \cdots \\ h^{(t-1)}_{\text{TTLM}_R} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{i=1}^{R} \mathbf{G}^{(t)}_{1i} h^{(t-1)}_{\text{TTLM}_1} \\ \sum_{i=1}^{R} \mathbf{G}^{(t)}_{2i} h^{(t-1)}_{\text{TTLM}_2} \\ \cdots \\ \sum_{i=1}^{R} \mathbf{G}^{(t)}_{Ri} h^{(t-1)}_{\text{TTLM}_R} \end{bmatrix}\end{aligned}$$

Observing the calculation, $\mathbf{G}^{(1)}$, $\mathbf{G}$ and $\mathbf{G}^{(t)}$ theoretically have no parameters in common (though we set $\mathbf{G}^{(1)} = \mathbf{G}^{(t)}$ for simplicity). Further, their roles in TTLM are different: $\mathbf{G}^{(1)}$ can be viewed as a word embedding matrix; $\mathbf{G}$ deals with two sources of information, i.e. hidden state and input word; $\mathbf{G}^{(t)}$ extracts the evidence provided in $\boldsymbol{h}^{(t-1)}_{\text{TTLM}}$ and generates a set of scores over vocabulary.

B. Relationship between TTLM and some RNNs

We now demonstrate the relationship between TTLM and Second-order RNNs, Recurrent Arithmetic Circuits (RACs) and Multiplicative Integration RNNs (MI-RNNs).

To avoid symbol clutter when representing different RNNs, the notation is: $\boldsymbol{W}^{hx} \in \mathbb{R}^{R\times|V|}$ denotes the input-to-hidden matrix, $\boldsymbol{W}^{hh} \in \mathbb{R}^{R\times R}$ denotes hidden-to-hidden matrix, $\phi(\cdot)$ is an element-wise nonlinear activation function. Also, different hidden states are denoted as: Second-order RNNs ($\boldsymbol{h}^{(t)}_{\text{2nd}}$), RACs ($\boldsymbol{h}^{(t)}_{\text{RAC}}$) and MI-RNNs ($\boldsymbol{h}^{(t)}_{\text{MI}}$).

B.1. Relation to Second-order RNNs

Unlike Vanilla-RNNs (Mikolov & Zweig, 2012) that have *additive* blocks, Second-order RNNs have interaction between hidden states and input data in *multiplicative* form. This is achieved by a third-order tensor $\mathbf{T}$ with the i-th coordinate of the

hidden states $\boldsymbol{h}_{\text{2nd}}^{(t)}$ defined as (Hochreiter & Schmidhuber, 1997; Maupomé & Meurs, 2020):

$$h_{\text{2nd}_i}^{(t)} = \phi(\boldsymbol{f}(x^{(t)})^T \mathbf{T}_{i,:,:} \boldsymbol{h}_{\text{2nd}}^{(t-1)} + \boldsymbol{b}) \tag{12}$$

where $\mathbf{T}_{i,:,:} \in \mathbb{R}^{M \times R}$ is the ith slice of tensor $\mathbf{T} \in \mathbb{R}^{M \times R \times R}$, and $\boldsymbol{b}$ is a bias vector. For simplicity, we will ignore $\boldsymbol{b}$ for other variants of RNNs since $\boldsymbol{b}$ can be seen as 0th component of $\boldsymbol{f}(x^{(t)})$ which equals to 1. (Rabusseau et al., 2019) has provided that Tensor Trains can generalize linear Second-order RNNs. We here provide a basic proof from the perspective of recursive property in TTLM.

Claim B.1. The third-order tensor $\mathbf{T}$ in Second-order RNNs equals the TT cores in TTLM. There is a nonlinear activation ϕ such that the hidden states of Second-order RNNs is identical to that of TTLM when they are accompanied by ϕ.

Proof. The proof is based on the following observation: We recursively unfold the calculation of TTLM in Eq. 4:

$$\begin{aligned} p(X) &= \sum_{i=1}^{|V|} f(x^{(1)})_{i_1} \mathcal{G}_{i_1 \alpha_1}^{(1)} \cdots \\ &= \sum_{i_1, i_2 = 1}^{|V|} \sum_{\alpha_1 = 1}^{R} f(x^{(1)})_{i_1} \mathcal{G}_{i_1 \alpha_1}^{(1)} f(x^{(2)})_{i_2} \mathcal{G}_{\alpha_1 i_2 \alpha_2} \cdots \\ &\vdots \\ &= \sum_{i_1, \cdots, i_N = 1}^{|V|} \sum_{\alpha_1, \cdots, \alpha_{N-1} = 1}^{R} f(x^{(1)})_{i_1} \mathcal{G}_{i_1 \alpha_1}^{(1)} f(x^{(2)})_{i_2} \mathcal{G}_{\alpha_1 i_2 \alpha_2} \cdots f(x^{(N)})_{i_N} \mathcal{G}_{\alpha_{N-1} i_N}^{(N)} \end{aligned} \tag{13}$$

Observe in the above, that at each time step, $\mathbf{G}$ has two sources of "input": the information from the previous recursive unfolding (*e.g.*, in the second line, the first line is the previous information), and the input data $\boldsymbol{f}(x^{(t)})$. From this perspective, $\mathbf{G}$ acts as a bilinear map $\mathbf{G} : \mathbb{R}^{|V|} \times \mathbb{R}^R \to \mathbb{R}^R$, and we can regard the information in the previous line as a hidden state $\boldsymbol{h}_{\text{TTLM}}^{(t)}$, given by:

$$h_{\text{TTLM}_{\alpha_t}}^{(t)} = \sum_{i_t=1}^{|V|} \sum_{\alpha_t=1}^{R} f(x^{(t)})_{i_t} \mathcal{G}_{i_t \alpha_t \alpha_{t-1}} h_{\text{TTLM}_{\alpha_{t-1}}}^{(t-1)} \tag{14}$$

where we permute the indices of $\mathcal{G}_{\alpha_{t-1} i_t \alpha_t}$ as $\mathcal{G}_{i_t \alpha_t \alpha_{t-1}}$ (note that this does not change the number of indices).

We can also represent the hidden states in Second-order RNNs shown by Eq. 12 in element-wise fashion:

$$\begin{aligned} h_{\text{2nd}_i}^{(t)} &= \phi(\boldsymbol{f}(x^{(t)})^T \mathbf{T}_{i,:,:} \boldsymbol{h}_{\text{2nd}}^{(t-1)}) \\ &= \phi\left(\sum_{j=1}^{|V|} \sum_{k=1}^{R} f(x^{(t)})_j \mathcal{T}_{jik} h_{\text{2nd}_k}^{(t-1)} \right) \end{aligned} \tag{15}$$

where j, k are the dummy indices as i_t, α_t; i specifies the coordinate of $\boldsymbol{h}_{2nd}^{(t)}$ just like α_t for $\boldsymbol{h}_{\text{TTLM}}^{(t)}$. Thus, $\mathbf{T}$ and $\mathbf{G}$ are the same-sized trainable bi-linear map.

After demonstrating that the third-order tensor $\mathbf{T}$ in Second-order RNNs equals the TT cores $\mathbf{G}$, the only difference between the hidden states in Eq. 15 and in Eq. 14 is ϕ. If we add ϕ for $\boldsymbol{h}_{\text{TTLM}}^{(t)}$, the hidden states of Second-order RNNs and TTLM are identical, as shown in Fig. 7a.

□

B.2. Relation to RACs and MI-RNNs

We here focus on Multiplicative Integration (MI), a way to connect two sources of inputs by the Hadamard product '$\odot$'. MI has been used in RACs, Multiplicative RNNs (M-RNNs) (Sutskever et al., 2011) and MI-RNNs:

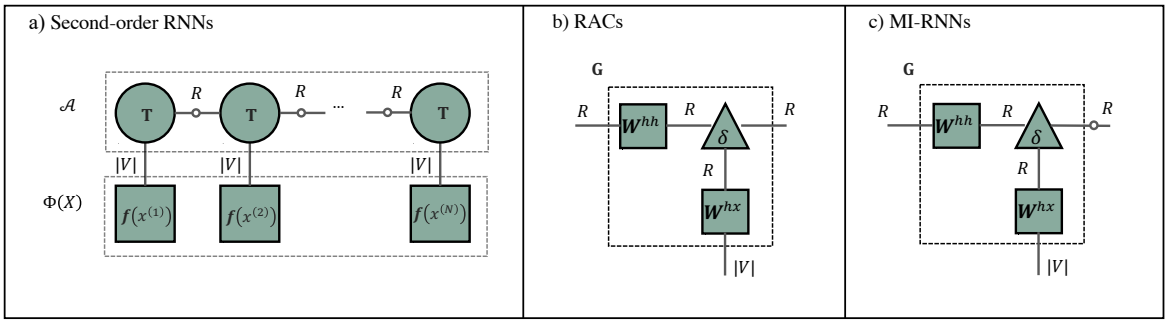

Figure 7. a) Second-order RNNs under TTLM framework. b) Hidden state of RACs under TTLM framework. c) hidden state of MI-RNNs under TTLM framework. The dashed line in the square denotes $\mathcal{A}$, $\Phi(X)$ or $\mathbf{G}$. The small hollow circles denote the activation functions.

Recurrent Arithmetic Circuits (RACs) are recurrent networks with hidden states $\boldsymbol{h}_{\text{RAC}}^{(t)}$ defined as (Levine et al., 2018):

$$\boldsymbol{h}_{\text{RAC}}^{(t)} = \boldsymbol{W}^{hx} \boldsymbol{f}(x^{(t)}) \odot \boldsymbol{W}^{hh} \boldsymbol{h}_{\text{RAC}}^{(t-1)} \tag{16}$$

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

Multiplicative Integration RNNs (MI-RNNs) are RACs with an activation function and hidden states $\boldsymbol{h}_{\text{MI}}^{(t)}$ defined as (Wu et al., 2016):

$$\boldsymbol{h}_{\text{MI}}^{(t)} = \phi(\boldsymbol{W}^{hx} \boldsymbol{f}(x^{(t)}) \odot \boldsymbol{W}^{hh} \boldsymbol{h}_{\text{MI}}^{(t-1)}) \tag{17}$$

Claim B.2. Given the condition the TT-scores: $\mathbf{G} = \boldsymbol{W}^{hx} \odot \boldsymbol{W}^{hh}$. The hidden states of RACs are identical to that of TTLM. There is a nonlinear function ϕ such that the hidden states of MI-RNNs are identical to that of TTLM if they are accompanied by ϕ.

Proof. The proof is based on the following observation: In the language of tensor contractions, Eq. 16 involves contracting the input weights matrix $\boldsymbol{W}^{hx}$ with the input vector $\boldsymbol{f}(x^{(t)})$, and contracting the hidden weights matrix $\boldsymbol{W}^{hh}$ with $\boldsymbol{h}_{\text{RAC}}^{(t-1)}$. 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. 16 in element-wise fashion:

$$\begin{aligned} h_{\text{RAC}_{\alpha_t}}^{(t)} &= \sum_{i_t=1}^{|V|} \sum_{\alpha_t=1}^{R} f(x^{(t)})_{i_t} W_{i_t j}^{hx} \delta_{j \alpha_t k} W_{k \alpha_{t-1}}^{hh} h_{\text{RAC}_{\alpha_{t-1}}}^{(t-1)} \\ &= \sum_{i_t=1}^{|V|} \sum_{\alpha_t=1}^{R} f(x^{(t)})_{i_t} \mathbf{G}_{i_t \alpha_t \alpha_{t-1}} h_{\text{RAC}_{\alpha_{t-1}}}^{(t-1)} \end{aligned} \tag{18}$$

where $\mathbf{G} = \boldsymbol{W}^{hx} \odot \boldsymbol{W}^{hh}$. In this case, the hidden state of TTLM in Eq. 14 is equal to the hidden state of RACs in Eq. 18, as shown in Fig. 7b. Similarly, if Eq. 14 is accompanied with an activation function ϕ, Eq. 14 is equal to the hidden state of MI-RNNs in Eq. 17 as shown in Fig. 7c.

□

Given Claim B.1 and B.2, the three models shall be simulated by TTLM with a nonlinear activation function and we leave finding a theoretical proof of this conjecture to a future work.

C. Comparison with related work on experiment datasets

Dataset	Data type	Vocab	N	Real-world language
Tomita grammars (Tomita, 1982)	Disc.	2	10k	$\times$
Motzkin grammar (Alexander et al., 2021)	Disc.	3	10k	$\times$
Email addresses (Radev, 2008)	Disc.	≤ 256	4k	$\times$
POWER (Dua & Graff, 2017)	Cont.	-	1659k	$\times$
GAS (Fonollosa et al., 2015)	Cont.	-	852k	$\times$
HEPMASS (Baldi et al., 2016)	Cont.	-	315k	$\times$
MINIBOONE (Roe et al., 2005)	Cont.	-	29k	$\times$
BSDS300 (Martin et al., 2001)	Cont.	-	1000k	$\times$
PTB (Marcinkiewicz, 1994)	Disc.	10k	31k	$\surd$
WikiText-2 (Merity et al., 2016)	Disc.	30k	73k	$\surd$

Table 3. The column "datasets" means the training datasets. The first three are used in (Miller et al., 2021), the following five are used in (Novikov et al., 2021), and the last two are used in our paper. The column "N" denotes the number of training examples. The column "Vocab" denotes the number of types. The value "Cont." denotes continuous variable, while the "Disc." denotes discrete variables.