LANGUAGE MODELING USING TENSOR TRAINS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tensor networks have previously been shown to have potential in language modelling in theory, but lack of practical evidence support. We propose a novel Tensor Train Language Model (TTLM) based on Tensor-Train decomposition. We prove that TTLM generalizes Second-order Recurrent Neural Networks (RNNs), Recurrent Arithmetic Circuits and Multiplicative Integration RNNs in the sense that the architecture of all of these are, essentially, special cases of that of TTLM. To show the usefulness of TTLM, we perform a principled experimental evaluation on language modeling tasks, showing that our proposed variants, TTLM-large and TTLM-Tiny, can be more effective than Vanilla RNN while TTLM-Tiny has the half of the model size.

1 INTRODUCTION

A language model assigns a probability to each N-gram of words from some vocabulary V; the number of such N-grams increases exponentially w.r.t to length n, hence the domain of a language models is by definition the exponential space $\mathbb{V}^N$. However, due to that the vanilla exponential space is intractable, existing work tends to use recurrent or auto-regressive architectures to generate conditional probabilities based on the context (typically encapsulated as a fixed-length dense vector). This indeed simplifies calculation.

Recently, researchers (Pestun & Vlassopoulos, 2017; Miller et al., 2021) have reconsidered the view of language models as joint probabilities of text, as it leads to an exponential tensor space representations. When measuring joint probabilities, long-range correlations between words could be reserved in the exponential tensor space. To deal with the exponential space complexity, a mathematical tool called 'tensor network' [1] has been used to reduce the exponential space of language modeling to a tractable one (Pestun & Vlassopoulos, 2017). However, the so called 'tensor network language model' in Pestun & Vlassopoulos (2017) is currently a concept that needs to be proved practically.

As a proof-of-concept work, we derives 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 representation. The probability of the sentence is obtained by the inner product of two high-dimensional tensors: the input $\Phi(X)$ and the global coefficients $\mathcal{A}$.

Interestingly, we prove that the proposed TTLM could generalizes a series of existing RNNs (i.e., Second-order RNN (Goudreau et al., 1994), Recurrent Arithmetic Circuits (Levine et al., 2018) and Multiplicative Integration RNNs (Wu et al., 2016)). This not only opens a new eye to understand RNNs but also gives some natural implementations for TTLM.

Accompanying with these RNNs implementations of TTLM, we propose two variants TTLM-Large, TTLM-Tiny. We benchmark these TTLM variants and analyze the difference in their working mechanism and behaviors. Experimental results on language modeling tasks show that our TTLM variants could outperform than Vanilla-RNNs under the same training setting. At the same time, TTLM-Tiny uses half parameters of Vanilla-RNNs. These demonstrate the feasibility of TTLM.

The **contributions** of this work are three-fold: (i) We propose a novel Tensor Train Language Model, as a first attempt to apply tensor networks on language modeling tasks; (ii) We theoretically use TTLM to generalize a series of existing RNNs, and proposed two novel variants: TTLM-Large and

[1] Tensor networks are, roughly, decompositions of large tensors into sets of smaller tensors, and have been employed in physics, mathematics and machine learning (Sun et al., 2020; Novikov et al., 2015; Cohen et al., 2016a; Stoudenmire & Schwab, 2016b; Cheng et al., 2019; Novikov et al., 2016; Selvan & Dam, 2020).

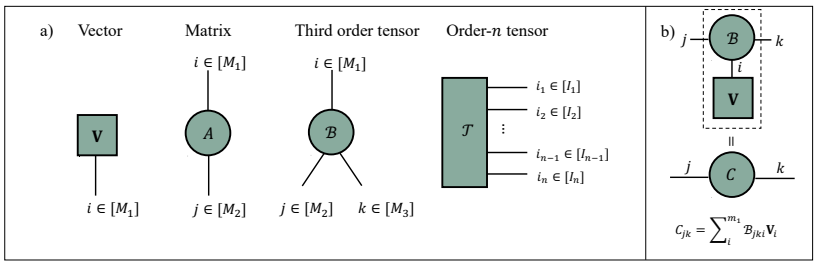


Figure 1: Several examples of tensor diagram notation

TTLM-Tiny, and (iii) Our experiment demonstrates that TTLM-Tiny can be more effective than Vanilla RNNs with half of the model size.

2 PRELIMINARIES

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

For the purposes of this paper, every **tensor** $\mathcal{T}$ 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 $\mathcal{T}_{ij}$. The *order* of a tensor is how many indices it has (e.g., a vector is a first-order tensor, a matrix is a second-order tensor, etc.). The *dimension* of a tensor refers to the number of values that a particular index can take, e.g., the dimension of $\mathcal{B} \in \mathbb{R}^{M_1 \times M_2 \times M_3}$ is $M_1 \times M_2 \times M_3$.

Tensor product denoted by $\otimes$, is an operator that intakes two tensors $\mathcal{E} \in \mathbb{R}^{M_1 \times \cdots \times M_j}$ (order j) and $\mathcal{F} \in \mathbb{R}^{M_{j+1}, \times \cdots \times M_{j+k}}$ (order k) and return a tensor $\mathcal{D}_{i_1 \cdots i_{j+k}} = \mathcal{E}_{i_1 \ldots i_j} \cdot \mathcal{F}_{i_{j+1} \cdots i_{j+k}}$ (order $j+k$) defined by : $\mathcal{D}_{i_1 \cdots i_{j+k}} = \mathcal{E}_{i_1 \ldots i_j} \cdot \mathcal{F}_{i_{j+1} \cdots i_{j+k}}$.

For two tensor $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \times \cdots I_n}$ of the same size, their **inner product** is defined as $\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{I_2} \sum_{i_1=1}^{I_2} \cdots \sum_{i_n=1}^{I_n} \mathcal{X}_{i_1, i_2, \ldots, i_n} \mathcal{Y}_{i_1, i_2, \ldots, i_n}$.

As is customary, we use a diagrammatic notation for networks of contracted tensors. There are two rules of tensor diagrams: 1) tensors are notated by solid shapes with a number of 'legs' corresponding to its indices (see Fig. 1a) 2) connecting two index lines implies a **contraction**, or summation over the connected indices (see Fig. 1b). In this paper we augment our equations with these diagrams to make them easier to visualize.

3 TENSOR NETWORK LANGUAGE MODEL

Language model A *language model* maps *N*-grams to probabilities:

$$g : \mathbb{V}^N \to \mathbb{R}^+$$

where $\sum_{x \in \mathbb{V}^N} g(x) = 1$. In Sec.3.1, we give a regression-based language model in tensor space.

3.1 LANGUAGE MODEL IN A TENSOR SPACE

Text representation A given text (e.g., a *N*-gram X) consists of many words, $X = \{x^{(1)}, x^{(2)}, \cdots, x^{(d)}\}$. Let $\mathbf{f}$ be a feature extractor, each view of X could be represented as a concatenation of feature maps:

$$\Phi(X)_{\text{concat}} = (\mathbf{f}_1(x^{(1)}); \mathbf{f}_2(x^{(2)}); \cdots ; \mathbf{f}_d(x^{(d)}))$$

where $\mathbf{f}$ could be one-hot encoding or word embedding.

Natural language typically has complex dependencies between features (e.g., words) (Hou et al., 2013) [2], that are not captured by standard methods such as feature concatenation (like $\Phi(X)_{\text{concat}}$). One could also see the similar interaction between any arbitrary features in Factorization machines (Rendle, 2010). We now define a representation designed to capture these dependencies:

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

where the tensor space is $\mathbb{R}^{M^d} := \underbrace{\mathbb{R}^M \otimes \mathbb{R}^M \otimes \cdots \otimes \mathbb{R}^M}_{d}$. Here, each dimension of $\mathbf{f}_i$ represents an independent meaning-bearing unit, which can be a morpheme, a latent factor, or even a word in a rough sense. Consequently, $\Phi(X)$ is a M^d-dimensional tensor which records all possible interaction between meaning-bearing units in X.

Regression for Language Model Inspired by Kossaifi et al. (2020), we define a tensor regression model to obtain a the estimated probability for each text X:

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

where $\mathcal{A} \in \mathbb{R}^{M^d}$ is a regression weight tensor which has the same shape as $\Phi(X)$; $\langle \cdot \rangle$ denotes the inner product of two same-sized tensors. Similar functions were considered in Novikov et al. (2016); Stoudenmire & Schwab (2016a); Khrulkov et al. (2017).

3.2 Tensor Train Decomposing $\mathcal{A}$

We use TT decomposition to represent $\mathcal{A}$ in the TT format (Oseledets, 2011) as follow:

$$\begin{aligned}\mathcal{A}_{w_1 w_2 \ldots w_d} &= \underbrace{\mathbf{G}^{(1)}[w_1]}_{1 \times r_1} \underbrace{\mathbf{G}^{(2)}[w_2]}_{r_1 \times r_2} \cdots \underbrace{\mathbf{G}^{(d)}[w_d]}_{r_{d-1} \times 1} \\ &= \sum_{\alpha_1 = 1}^{r_1} \sum_{\alpha_2 = 2}^{r_2} \cdots \sum_{\alpha_{d-1} = 1}^{r_{d-1}} \mathbf{G}^{(1)}_{w_1 \alpha_1} \mathbf{G}^{(2)}_{\alpha_1 w_2 \alpha_2} \cdots \mathbf{G}^{(d)}_{\alpha_{d-1} w_d}\end{aligned} \tag{3}$$

where the tensors $\mathbf{G}^{(t)} \in \mathbb{R}^{r_{t-1} \times M \times r_t}$ ($t = 1, ..., d$, $r_0 = r_d = 1$ by definition) are the so-called **TT cores**. $w_1, w_2, \cdots, w_d$ together denotes the specific entry $\mathcal{A}_{w_1 w_2 \cdots w_d}$. The diagrammatic notation of the TT format is .

3.3 The Tensor Train Language model

We now combine $\Phi(X)$ and $\mathcal{A}$ in the TT format, to obtain our *Tensor Train Language Model* (TTLM).

Suppose the sequence of indices of words in the N-gram X is $w_1, w_2, \cdots, w_d$, where $w_i \in \{1, 2, \cdots, |V|\}$. By Eq. 3, we have $\mathcal{A}_{w_1, w_2, \cdots, w_d} = \mathbf{G}^{(1)}[w_1]\mathbf{G}^{(2)}[w_2] \cdots \mathbf{G}^{(d)}[w_d]$, and elements of $\mathbf{G}^{(t)}[w_t]$ can thus be represented as:

$$\mathbf{G}^{(t)}_{\alpha_{t-1} w_t \alpha_t} = \sum_{i_t = 1}^{|V|} \mathbf{f}_\theta(x^{(t)})_{i_t} \mathbf{G}^{(t)}_{\alpha_{t-1} i_t \alpha_t} \tag{4}$$

[2] Such dependency (including collocation) was considered as an analogy of entanglement, or said 'statistical correlation' (Hou et al., 2013).

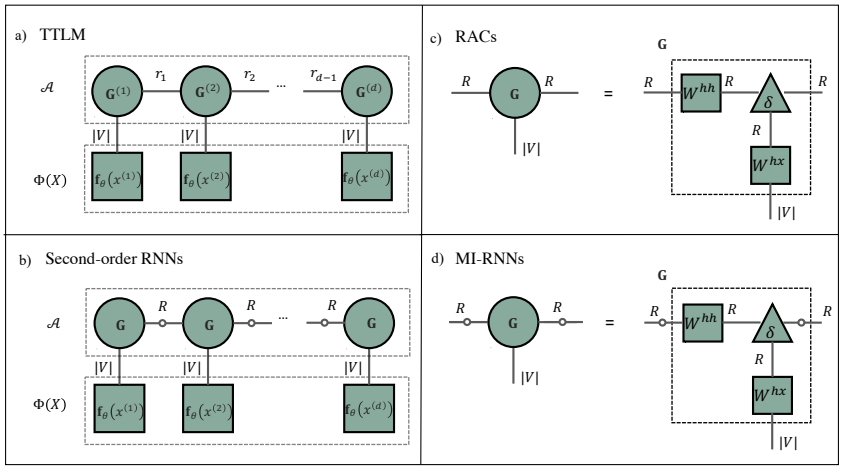
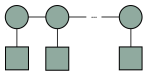


Figure 2: a) Tensor Train Language Model given Eq. 5. b) Second-order RNNs under TTLM framework. c) Hidden state of RACs under TTLM framework. d) hidden state of MI-RNNs under TTLM framework. The dashed line in the square represents $\mathcal{A}$, $\Phi(X)$ or $\mathbf{G}$. The small hollow circles represent the activation functions.

where each $\mathbf{f}_\theta(x^{(t)})$ is a one-hot vector having $w_t = 1$ for at most one t, and has zeros elsewhere. When we use $\mathbf{f}_\theta(x^{(t)})$ to retrieve a slice of $\mathbf{G}^{(t)}$, we can naturally integrate the input data into the TT format of $\mathcal{A}$ by substituting Eq. 4 into Eq. 3:

$$
\begin{aligned}
\mathcal{A}_{w_1,\cdots,w_d} &= \sum_{i_1,\cdots,i_d=1}^{|V|} \sum_{\alpha_1=1}^{r_1} \cdots \sum_{\alpha_{d-1}=1}^{r_{d-1}} \mathbf{f}_\theta(x^{(1)})_{i_1} \mathbf{G}^{(1)}_{i_1\alpha_1} \mathbf{f}_\theta(x^{(2)})_{i_2} \mathbf{G}^{(2)}_{\alpha_1 i_2 \alpha_2} \cdots \mathbf{f}_\theta(x^{(d)})_{i_d} \mathbf{G}^{(d)}_{\alpha_{d-1} i_d} \\
&= \sum_{\alpha_1=1}^{r_1} \cdots \sum_{\alpha_{d-1}=1}^{r_{d-1}} \mathbf{G}^{(1)}_{w_1\alpha_1} \mathbf{G}^{(2)}_{\alpha_1 w_2 \alpha_2} \cdots \mathbf{G}^{(d)}_{\alpha_{d-1} w_d} \qquad (5) \\
&= \mathbf{G}^{(1)}[w_1]\mathbf{G}^{(2)}[w_2]\cdots\mathbf{G}^{(1)}[w_d]
\end{aligned}
$$

By representing $\Phi(X)$ in the TT format of $\mathcal{A}$, we can calculate Eq. 2 much more efficiently due to the low-dimensional form. We refer to the representation of $\mathcal{A}$ by Eq. 5 as TTLM, with diagrammatic notation:  .

4 Bridging TTLM and RNNs

We now prove that TTLM is a generalization of Second-order RNNs in Sec. 4.1, and subsequently generalize the proof to demonstrate Recurrent Arithmetic Circuits (RACs) and Multiplicative Integration RNNs (MI-RNNs) are variants of TTLM in Sec. 4.2.

To avoid symbol clutter when representing hidden states in different RNNs, we use $\mathbf{f}_\theta(x^{(t)}) \in \mathbb{R}^{|V|}$ to represent the input (as it Sec. 3.3); $W^{hx} \in \mathbb{R}^{R\times|V|}$ is the input-to-hidden transition matrix; $W^{hh} \in \mathbb{R}^{R\times R}$ is the hidden-to-hidden transition matrix; and $\phi(\cdot)$ is an activation function. To differentiate hidden states, we use the notation: Second-order RNNs ($\mathbf{h}^{(t)}_{\text{2nd}}$), TNLM ($\mathbf{h}^{(t)}_{\text{TNLM}}$), RACs ($\mathbf{h}^{(t)}_{\text{RAC}}$) and MI-RNNs ($\mathbf{h}^{(t)}_{\text{MI}}$).

4.1 TTLM generalizes 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 $\mathcal{T}$ with the i-th coordinate of the hidden states $\mathbf{h}_{\text{2nd}}^{(t)}$ defined as (Hochreiter & Schmidhuber, 1997; Maupomé & Meurs, 2020):

$$h_{\text{2nd}_i}^{(t)} = \phi(\mathbf{f}_\theta(x^{(t)})^T \mathcal{T}[i] \mathbf{h}_{\text{2nd}}^{(t-1)} + \mathbf{b}) \tag{6}$$

where $\mathcal{T}[i] \in \mathbb{R}^{M \times R}$ is the ith slice of tensor $\mathcal{T} \in \mathbb{R}^{M \times R \times R}$, 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 $\mathbf{f}_\theta(x^{(t)})$ which equals to 1.

Claim 4.1. *The third-order tensor $\mathcal{T}$ in Second-order RNNs equals the TT cores in TTLM. The hidden states of Second-order RNNs is identical to that of TTLM if they are accompanied by an activation function.*

Proof. The detailed proof can be found in Appendix A.1. The diagrammatic notation of Second-order RNNs is shown in Fig. 2b.

□

4.2 TTLM GENERALIZES 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:

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

$$\mathbf{h}_{\text{RAC}}^{(t)} = W^{hx} \mathbf{f}_\theta(x^{(t)}) \odot W^{hh} \mathbf{h}_{\text{RAC}}^{(t-1)} \tag{7}$$

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 $\mathbf{h}_{\text{MI}}^{(t)}$ defined as (Wu et al., 2016):

$$\mathbf{h}_{\text{MI}}^{(t)} = \phi(W^{hx} \mathbf{f}_\theta(x^{(t)}) \odot W^{hh} \mathbf{h}_{\text{MI}}^{(t-1)}) \tag{8}$$

Claim 4.2. *Given the condition the TT-scores: $\mathbf{G} = W^{hx} \odot W^{hh}$. The hidden states of RACs are identical to that of TTLM. The hidden states of MI-RNNs are identical to that of TTLM if they are accompanied by an activation function.*

Proof. The detailed proof can be found in Appendix A.2. The diagrammatic notations of RACs and MI-RNN are shown in Fig. 2. □

Given Claim 4.1, 4.2 and the fact that TT scores of TTLM can vary, TTLM is a generalization of Second-RNNs, RACs and MI-RNNs. Consequently, these models shall be simulated by TTLM with an activation function and we leave finding a theoretical proof of this conjecture to a future work.

5 VARIANTS OF TTLM

To show the versatility and practical applicability of the TTLM framework, we now introduce two new variants that have not appeared in the literature before: TTLM-Large and TTLM-Tiny in Sec. 5.1. Then we briefly conclude the existing variants of TTLM in Sec. 5.2.

5.1 NEW VARIANTS: TTLM-LARGE AND TTLM-TINY

At time t, the two models reshape the word embedding into matrix $M^{(t)}$ and contracted with the hidden states as shown in Fig. 3. Formally, $M^{(t)}$ is defined as:

$$\begin{aligned} M_{\text{large}}^{(t)} &= \text{reshape}\left(\mathbf{f}_\theta(x^{(t)})^T W^{xe} W^{eh}\right) \\ M_{\text{tiny}}^{(t)} &= \text{reshape}\left(\mathbf{f}_\theta(x^{(t)})^T W^{xe}\right) \end{aligned} \tag{9}$$

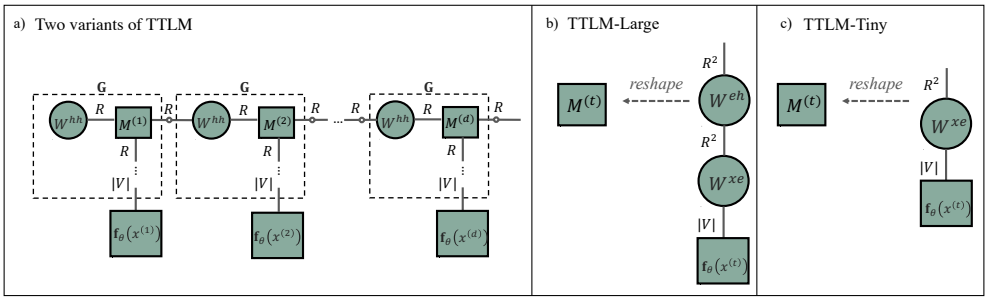


Figure 3: Two variants: TTLM-Large and TTLM-Tiny. The only difference between TTLM-Large and TTLM-Tiny is whether to use the input-to-hidden transition matrix W^{eh}.

where reshape$(\cdot)$ is a function transforms a R^2-dimensional vector to a matrix with the size $R \times R$ without changing its value; $\mathbf{f}_\theta(x^{(t)}) \in \mathbb{R}^{|V|}$; $W^{xe} \in \mathbb{R}^{|V| \times R^2}$ is the embedding matrix; $W^{eh} \in \mathbb{R}^{R^2 \times R^2}$ is the input-to-hidden transition matrix; $M^{(t)}_{\text{large}}, M^{(t)}_{\text{tiny}} \in \mathbb{R}^{R \times R}$. Thus, the only difference between TTLM-Large and TTLM-Tiny is the input-to-hidden transition matrix W^{xe}.

Their hidden states are defined as:

$$\mathbf{h}^{(t)} = M^{(t)} W^{hh} \mathbf{h}^{(t-1)} \tag{10}$$

One obvious advantage for using "tensor contraction" and "reshape" is that the TT cores of our models have a simpler parametrization. Such simplification, particularly TTLM-Tiny, in TT cores expects to reduce model parameters and avoid overfitting. This benefit has been supported by Wu et al. (2016), who shows that the multiplication integration ($\otimes$ or a diagonal tensor δ as proved in Appendix A.2) can outperform third-order tensor $\mathcal{T}$ in Second-order RNNs.

5.2 Existing TTLM variants

Along with our two proposed models, Second-RNNs, RACs and MI-RNN are RNNs implementation of TTLM as proved in Sec. 4. Here, we shortly conclude the difference between the three models: 1) Second-order RNNs uses the third-order $\mathcal{T}$ as the TT cores with an activation function given Eq. 6; 2) RACs uses $W^{hx} \odot W^{hh}$ as the TT cores given Eq. 7; 3) MI-RNNs uses $W^{hx} \odot W^{hh}$ as the TT cores with an activation function given Eq. 8.

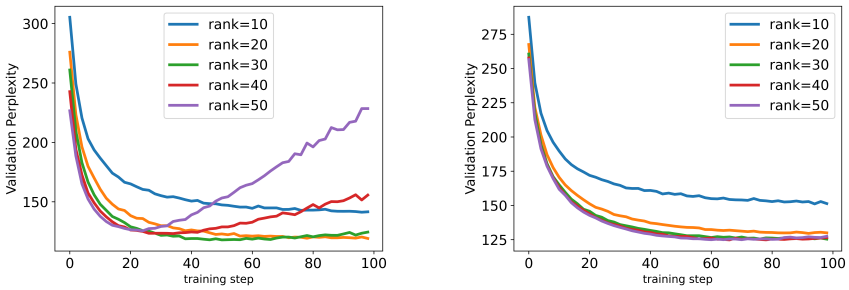
These five models are chosen to be studied in the experiment, because they can all be considered as one of "special" implementations of TTLM, and they can help us differentiate properties of TTLM variants.

6 Experiments

We conduct experiments from three distinct perspectives: (1) The **rank** of TT decomposition has been proved to be the dimension of the hidden states of RNNs (Khrulkov et al., 2017). Here, we study the influence of the rank on the effectiveness of our TTLM variants in Sec 6.3.1. (2) Different TT cores settings will influence the number of trainable parameters. Sec. 6.3.2 compares the **model size** of TT variants (3) The non-linearity of **activation function** is not consistent with the linearity of the TT format. Thus, we evaluate if TTLM variants are flexible enough to be used in practice without activation function in Sec. 6.3.3.

6.1 Baselines and Experimental Setting

Baselines We provide some original results of RNN-based models as references: RNN-based language model (Mikolov & Zweig, 2012). Long Short-Term Memory network-based language model (LSTM) (Zaremba et al., 2014; Grave et al., 2016; Merity et al., 2017). Note that in order to follow



(a) Validation perplexity on TTLM-Large (b) Validation perplexity on TTLM-Tiny

Figure 4: Rank analysis for the TTLM-Large and TTLM-Tiny. RNNs in this figure is Vanilla-RNNs. The embedding size of RNNs is the same as TTLM-Large and TTLM-Tiny. RNNs-100, RNNs-200 and RNNs-300 means the RNNs with embedding size 100, 200, 300 respectively.

a realistic case, the baseline models implemented by us have an embedding matrix W^{xe} between one-hot vectors $\mathbf{f}_\theta(x^{(t)})$ and the input-to-hidden transition matrix W^{hx} [3].

Task, **Evaluation metric** and **Datasets** These models are compared on the language modeling task and evaluated by the Perplexity (PPL) (Meister & Cotterell, 2021): the lower the perplexity is, the better the model is. Two datasets are used : English Penn Treebank (PTB) (Marcinkiewicz, 1994) and WikiText-2 language modeling dataset (Merity et al., 2016).

Hyperparameters To compare the effectiveness of comparable models in the same scale: 1) We set the rank/hidden size of TTLM variants/Vanilla-RNNs is [5, 10, 20, 25, 30, 35, 40, 45, 50] (the reasons for this choice is explained in 6.3.1). Due to the "reshape" function introduced in Sec. 5.1, the embedding size of these models is the squared number of hidden size/ranks. 3) In addition, we also provide several common choices of embedding size in Vanilla-RNNs ([100, 200, 300]), while keeping the hidden size equals to the rank of our models (we name them as RNNs-100, RNNs-200, RNNs-300 correspondinly). This is to reduce the impact of the large embedding size on the model performance. 4) We use the SGD optimization algorithm with learning rate 5e-1. The random seed is fixed to make sure the experimental results are not influenced by initializing the weights. 4) We train all the models for 50 epochs and choose the best model in the validation set to predict the result in the test dataset. 5) We share the weights of input embedding and output embedding as shown in (Press & Wolf, 2016).

6.2 Experimental Results

Rank 1) The influence of the rank on our models based on the validation PPL is shown in Fig. 4. The validation PPL of TTLM-Large drops down at early training step but easily increase when the rank increasing. In contrast, the validation PPL of TTLM-Tiny stably decreases as the rank increases. 2) Compared to Vanilla-RNNs, the influence of the rank on our models based on the test PPL is shown in Fig. 5. Our models outperform Vanilla-RNNs on all used parameter settings. 3) Table. 1 provides a supplementary example to show the comparison of our models (when rank/hidden states = 20, embedding size = 400) with other baselines and some RNN-based models. TTLM-Large obtains the best PPL among these models, while TTLM-Tiny is more effective than Vanilla-RNN while uses half of parameters.

Model size Table. 1 lists the model size of baselines and our proposed models. TTLM-Tiny has the smallest model size. Second-order RNNs is 19 times larger than the TTLM-Tiny; Vnailla-RNNs, MI-RNNs, and RACs has the same model size that among TTLM variants.

Activation function Effects of $tanh$ activation function on TT variants based on PPL is shown Fig.6. For the convergence of validation perplexity, the activation function speeds up its convergence

[3]This change obeys the multiplicative form and follows the TT-format of TTLM (W^{xe} is part of TT cores)

Model	WikiText-2	PTB	Model Size	Cell Size
RNN (Mikolov & Zweig, 2012)		124.7	-	
LSTM (Zaremba et al., 2014)	-	114.5	-	
LSTM (Grave et al., 2016)	99.3	82.3	-	
LSTM (Merity et al., 2017)	100.9	80.6	-	
Vanilla-RNNs*	96.6	115.3	16.8K ($\times$2)	8.4K ($\times$20)
Second-order RNNs*	96.0	108.2	168.4K ($\times$19)	160K ($\times$390)
RACs*	97.6	116.8	16.8K ($\times$2)	8.4K ($\times$20)
MI-RNNs*	99.6	119.1	16.8K ($\times$2)	8.4K ($\times$20)
TTLM-Large	**82.3**	**99.3**	168.4K ($\times$19)	160K ($\times$390)
TTLM-Tiny	94.9	106.8	8.8K ($\times$1)	**420B** ($\times$1)

Table 1: PPL Evaluation on Test set in WikiText-2 and PTB. Models tagged with $*$ indicate that they are re-implemented by ourselves.

for TTLM-Large-tanh, TTLM-Tiny-tanh, MI-RNNs while barely influences that of Second-order RNNs. For the magnitude of the lowest validation perplexity, the activation function drags down TTLM-Large-tanh and TTLM-Tiny-tanh but have little influence on multiplicative integration and the third-tensor $\mathcal{T}$ in Second-order RNNs.

6.3 Discussion

6.3.1 Rank and Effectiveness Analysis

The rank of TT-format has been used to explain the expressive power or long-term memory capacity of RNNs (Khrulkov et al., 2017; Levine et al., 2018). However, the relationship between the rank and effectiveness has rarely shown practically.

Based Fig. 4a and Fig. 4b, we can find that TTLM-Large are more easy to be overfitting than TTLM-Tiny. Since the difference between the models is that TTLM has more trainable parameters (the number of parameters in W^{eh}) than TTLM-Tiny, this suggests that the simpler parameterization of the TT cores, the easier the model is to avoid overfitting. This finding follows the comparison between MI-RNNs and Second-order RNNs by Wu et al. (2016).

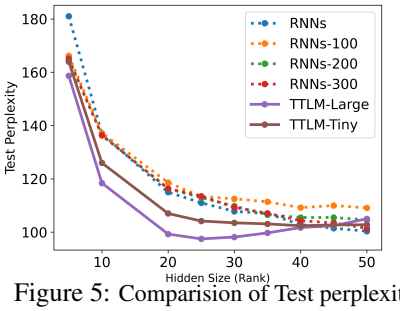


Figure 5: Comparision of Test perplexity

Despite the effectiveness of our models under the current hyperparameter settings, Fig. 5 reveals their limited potential when the rank is larger than 40 compared to the fact that the test perplexity of Vanilla-RNNs still stably decrease. Therefore, we expect that our model can outperform vanilla RNNs if using a low scale of hidden size (i.e. range from 5 to 50), and we treat this property as the "tradeoff" to use the simple parameterization of TT cores.

6.3.2 On the Model Size

The *model size* is defined to be the total number of parameters excluding word embedding layers (to be independent of the dataset size). Based on this definition, it equals to the size of *cell* and the size of *transfer layer*: 1) *Cell* means matrices in hidden states of each model (i.e.W^{hx} and W^{hh} in Vanilla-RNNs, RACs and MI-RNNs; $\mathcal{T}$ in Second-order RNNs; W^{eh} and W^{hh} in TTLM-Large; W^{hh} in TTLM-Tiny) 2) *transfer layer* refers the layer projects from the hidden states to the embedding layer. Note that this size of transfer are controlled to be same for baselines and models [4]. Thus, in our experiments, the only variable influence the *model size* is the size of cell, i.e. trainable parameters in hidden states.

[4]Because the dimension of hidden states/ranks and embeddings between different models are the same in experiments

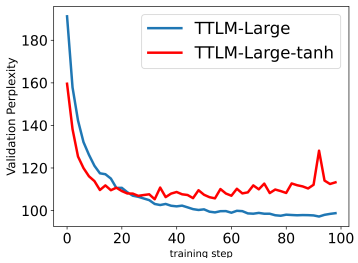
Therefore, and shown by Table. 1, RACs, MI-RNN and Vallina-RNNs have the same model size. TTLM-Large has the same model size of Second-order RNNs. For TTLM-Tiny, its model size is only half of Vallina-RNNs.

The more space-efficient version of TTLM (i.e. TTLM-small) could save some space, this is particularly beneficial in edge devices where memory and space is expensive. Also, smaller models could facilitate deployment in multi-GPU scenarios, in a sense the communication cost between GPU might be smaller.

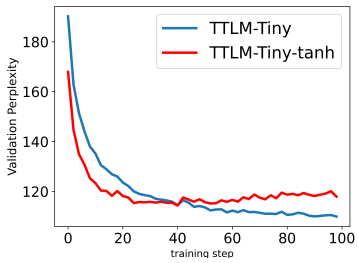
6.3.3 Activation Function Analysis.

Our finding is that the influence of activation functions on TTLM variants depends on TT cores settings, whether from the perspective of the convergence of validation PPL or the magnitude of the lowest validation PPL.

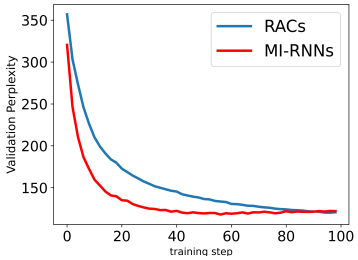
Thus, from an experimental point of view, the effect of activation functions on one TT variant shall not be simply analogized to another TT variant. This reminds us that we shall be careful for the analogy between tensor decomposition and existing neural network models at implementation level declared by previous researches (Khrulkov et al., 2017; Levine et al., 2018). The activation function could be an factor to influence such analogy.



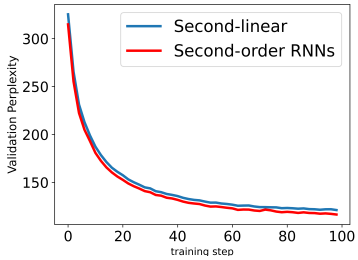
(a) TTLM-Large v.s. TTLM-Large-tanh



(b) TTLM-Tiny v.s. TTLM-Tiny-tanh



(c) RACs v.s. MI-RNN



(d) Second-linear v.s. Second-order RNNs

Figure 6: Comparison of influence of activation functions in TTLM variants. The suffix `-tanh` refers to a model using $tanh$ activation function. Second-linear refers to Second-order RNNs without activation function.

7 Conclusion

We propose a Tensor Train Language Model (TTLM) to apply tensor networks for language modelling. As a theoretical contribution, we prove that TTLM generalizes Second-order RNNs, RACs and MI-RNNs. As an experimental contribution, we propose two variants: TTLM-Large and TTLM-Tiny, and show that (i) they are more effective compared to existing TTLM variants and Vanilla-RNNs in the same training parameters (ii) TTLM-Tiny use the simplest TT core setting and use half parameters of Vanilla-RNNs. The rank and activation functions analysis fill the blank of the experimental studies of the tensor decomposition on language modeling tasks.

REFERENCES

Yingyue Bi, Yingcong Lu, Zhen Long, Ce Zhu, and Yipeng Liu. Chapter 1 - tensor decompositions: computations, applications, and challenges. In Yipeng Liu (ed.), *Tensors for Data Processing*, pp. 1–30. Academic Press, 2022. ISBN 978-0-12-824447-0. doi: https://doi.org/10.1016/B978-0-12-824447-0.00007-8. URL https://www.sciencedirect.com/science/article/pii/B9780128244470000078.

Song Cheng, Lei Wang, Tao Xiang, and Pan Zhang. Tree tensor networks for generative modeling. *Physical Review B*, 99(15):155131, 2019.

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

Nadav Cohen, Or Sharir, and Amnon Shashua. On the Expressive Power of Deep Learning: A Tensor Analysis, May 2016b. URL http://arxiv.org/abs/1509.05009. arXiv:1509.05009 [cs, stat].

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

Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016.

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

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

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

Valentin Khrulkov, Alexander Novikov, and Ivan Oseledets. Expressive power of recurrent neural networks. *arXiv preprint arXiv:1711.00811*, 2017.

Jean Kossaifi, Zachary C Lipton, Arinbjörn Kolbeinsson, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. Tensor regression networks. *The Journal of Machine Learning Research*, 21(1):4862–4882, 2020.

Yoav Levine, Or Sharir, and Amnon Shashua. Benefits of depth for long-term memory of recurrent networks. 2018.

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

Diego Maupomé and Marie-Jean Meurs. Language modeling with a general second-order RNN. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pp. 4749–4753, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.584.

Clara Meister and Ryan Cotterell. 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.

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

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

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

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

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. *Advances in neural information processing systems*, 28, 2015.

Alexander Novikov, Mikhail Trofimov, and Ivan Oseledets. Exponential machines. *arXiv preprint arXiv:1605.03795*, 2016.

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

Vasily Pestun and Yiannis Vlassopoulos. Tensor network language model. *arXiv preprint arXiv:1710.10248*, 2017.

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

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

Raghavendra Selvan and Erik B Dam. Tensor networks for medical image classification. In *Medical Imaging with Deep Learning*, pp. 721–732. PMLR, 2020.

Edwin Stoudenmire and David J Schwab. Supervised learning with tensor networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4799–4807. Curran Associates, Inc., 2016a. URL http://papers.nips.cc/paper/6211-supervised-learning-with-tensor-networks.pdf.

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

Zheng-Zhi Sun, Cheng Peng, Ding Liu, Shi-Ju Ran, and Gang Su. Generative tensor network classification model for supervised machine learning. *Physical Review B*, 101(7):075135, 2020.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan Salakhutdinov. On Multiplicative Integration with Recurrent Neural Networks, November 2016. URL http://arxiv.org/abs/1606.06630. arXiv:1606.06630 [cs].

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

APPENDIX

A TTLM GENERALIZES TO SECOND-ORDER RNNS, RACS AND MI-RNNS

The case of equal TT cores In analogy to RNNs we consider a special class of Tensor Trains with the property that all the intermediate TT cores are equal to each other: $\mathbf{G} = \mathbf{G}^{(2)}, ..., \mathbf{G}^{(d-1)}$, as usual in the literature (Khrulkov et al., 2017; Cohen et al., 2016b).

A.1 RELATION TO SECOND-ORDER RNNS

Claim 4.1. The third-order tensor $\mathcal{T}$ in Second-order RNNs equals the TT cores in TTLM. The hidden states of Second-order RNNs are identical to that of TTLM if they are accompanied by an activation function

Proof. The proof is based on the following observation: We recursively unfold the calculation of TTLM in Eq. 5:

$$\begin{aligned}
\mathcal{A}_{w_1,\cdots,w_d} &= \sum_{i_1=1}^{|V|} \mathbf{f}_\theta(x^{(1)})_{i_1} \mathbf{G}^{(1)}_{i_1\alpha_1} \cdots \\
&= \sum_{i_1,i_2=1}^{|V|} \sum_{\alpha_1=1}^{R} \mathbf{f}_\theta(x^{(1)})_{i_1} \mathbf{G}^{(1)}_{i_1\alpha_1} \mathbf{f}_\theta(x^{(2)})_{i_2} \mathbf{G}_{\alpha_1 i_2 \alpha_2} \cdots \\
&\quad \vdots \\
&= \sum_{i_1,\cdots,i_d=1}^{|V|} \sum_{\alpha_1,\cdots,\alpha_{d-1}=1}^{R} \mathbf{f}_\theta(x^{(1)})_{i_1} \mathbf{G}^{(1)}_{i_1\alpha_1} \mathbf{f}_\theta(x^{(2)})_{i_2} \mathbf{G}_{\alpha_1 i_2 \alpha_2} \cdots \mathbf{f}_\theta(x^{(d)})_{i_d} \mathbf{G}^{(d)}_{\alpha_{d-1} i_d}
\end{aligned} \tag{11}$$

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 $\mathbf{f}_\theta(x^{(t)})$. 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 $\mathbf{h}^{(t)}_{\text{TTLM}}$, given by:

$$h^{(t)}_{\text{TTLM}_{\alpha_t}} = \sum_{i_t=1}^{|V|} \sum_{\alpha_t=1}^{R} \mathbf{f}_\theta(x^{(t)})_{i_t} \mathbf{G}_{i_t \alpha_t \alpha_{t-1}} h^{(t-1)}_{\text{TTLM}_{\alpha_{t-1}}} \tag{12}$$

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

We can also represent the hidden states in Second-order RNNs shown by Eq. 6 in element-wise fashion:

$$\begin{aligned}
h^{(t)}_{\text{2nd}_i} &= \phi(\mathbf{f}_\theta(x^{(t)})^T \mathcal{T}[i] \mathbf{h}^{(t-1)}_{\text{2nd}}) \\
&= \phi\left(\sum_{j=1}^{|V|} \sum_{k=1}^{R} \mathbf{f}_\theta(x^{(t)})_j \mathcal{T}_{jik} h^{(t-1)}_{\text{2nd}_k} \right)
\end{aligned} \tag{13}$$

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

After demonstrating that the third-order tensor $\mathcal{T}$ in Second-order RNNs equals the TT cores $\mathbf{G}$, the only difference between the hidden states in Eq. 13 and in Eq. 12 is an activation function. If we add an activation function for $\mathbf{h}^{(t)}_{\text{TTLM}}$ (which TT cores are all equal), the hidden states of Second-order RNNs and TNLM are identical.

□

A.2 RELATION TO RACS AND MI-RNNS

Claim 4.2. Given the condition the TT-scores: $\mathbf{G} = W^{hx} \odot W^{hh}$. The hidden state of RACs is identical to that of TTLM. The hidden state of MI-RNNs is identical to that of TTLM if they are accompanied by an activation function.

Proof. The proof is based on the following observation: In the language of tensor contractions, Eq. 7 involves contracting the input weights matrix W^{hx} with the input vector $\mathbf{f}_\theta(x^{(t)}$, and contracting the hidden weights matrix W^{hh} with $\mathbf{h}_{RAC}^{(t-1)}$. 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. 7 in element-wise fashion:

$$\begin{aligned} h_{\text{RAC}_{\alpha_t}}^{(t)} &= \sum_{i_t=1}^{|V|} \sum_{\alpha_t=1}^{R} \mathbf{f}_\theta(x^{(t)})_{i_t} W_{i_t j}^{hx} \delta_{j\alpha_t k} W_{k\alpha_{t-1}}^{hh} h_{\text{RAC}_{\alpha_{t-1}}}^{(t-1)} \\ &= \sum_{i_t=1}^{|V|} \sum_{\alpha_t=1}^{R} \mathbf{f}_\theta(x^{(t)})_{i_t} \mathbf{G}_{i_t \alpha_t \alpha_{t-1}} h_{\text{RAC}_{\alpha_{t-1}}}^{(t-1)} \end{aligned} \tag{14}$$

where $\mathbf{G} = W^{hx} \odot W^{hh}$. In this case, the hidden state of TNLM in Eq. 12 is equal to the hidden state of RACs in Eq. 14, as shown in Fig. 2c. Similarly, if Eq. 12 is accompanied with an activation function ϕ, Eq. 12 is equal to the hidden state of MI-RNN in Eq. 8 as shown in Fig. 2d.

□