A SIMPLE RECURRENT UNIT WITH REDUCED TENSOR PRODUCT REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Widely used recurrent units, including Long-short Term Memory (LSTM) and Gated Recurrent Unit (GRU), perform well on natural language tasks, but their ability to learn structured representations is still questionable. Exploiting reduced Tensor Product Representations (TPRs) — distributed representations of symbolic structure in which vector-embedded symbols are bound to vector-embedded structural positions — we propose the TPRU, a simple recurrent unit that, at each time step, explicitly executes structural-role binding and unbinding operations to incorporate structural information into learning. The gradient analysis of our proposed TPRU is conducted to support our model design, and its performance on multiple datasets shows the effectiveness of our design choices. Furthermore, observations on linguistically grounded study demonstrate the interpretability of our TPRU.

1 INTRODUCTION

Recurrent units are widely used in sequence modelling tasks, including text, speech, DNA/RNA sequencing data, etc. Due to their ability to produce distributed vector representations for sequences with various lengths, and their property of being a universal function approximator (Schäfer & Zimmermann, 2006), they quickly succeeded in numerous research areas. The training algorithm, backpropagation through time (BPTT, Werbos et al. (1990)), has remained the same across the years, with known gradient vanishing and exploding issues. Many theoretically justifiable or empirically useful regularisations (Pascanu et al., 2013; Kanai et al., 2017) have been proposed to ease the learning process and to smooth the gradient flow for long sequences, and informative analysis methods have been conducted to visualise the information encoded in the learnt vector representations (Zeiler & Fergus, 2013; Karpathy et al., 2015). The main difficulty in understanding the learning process of a particular RNN unit comes from the extreme nonlinear activation functions, which cause the gradient flow to be chaotic. Here, we aim to propose a recurrent unit that has fewer nonlinear operations by incorporating Tensor Product Representations (Smolensky, 1990) .

Tensor product representation (TPR) is an instantiation of general neural-symbolic computing in which symbol structures are embedded in a vector space via a *filler-role decomposition*: the structure is captured by a set of N **roles** r_i (e.g., left-child-of-root), each of which is bound to a **filler** f_i (a symbol or substructure). A TPR embedding of a symbol structure $\boldsymbol{S}$ derives from vector embeddings of the roles $\{\boldsymbol{r}_i\}$ and their fillers $\{\boldsymbol{f}_i\}$ via the outer or tensor product: $\boldsymbol{S} = \sum_{i=1}^{N} \boldsymbol{r}_i \otimes \boldsymbol{f}_i = \sum_{i=1}^{N} \boldsymbol{r}_i \boldsymbol{f}_i^\top = \boldsymbol{R}\boldsymbol{F}^\top$, where $\boldsymbol{R}$ and $\boldsymbol{F}$ respectively denote matrices having the role or filler vectors as columns. Each $\boldsymbol{r}_i \otimes \boldsymbol{f}_i$ is the embedding of a role-filler binding: a constituent; $\otimes$ is the **binding** operation. The **unbinding** operation returns the filler of a particular role in $\boldsymbol{S}$; it is performed by the inner product: $\boldsymbol{f}_i = \boldsymbol{u}_i^\top \boldsymbol{S}$, where $\boldsymbol{u}_i$ is the dual of $\boldsymbol{r}_i$, satisfying $\boldsymbol{u}_i^\top \boldsymbol{r}_j = \delta_{ij} \equiv 1$ if $i = j$ else 0. Letting $\boldsymbol{U}$ be the matrix with columns $\{\boldsymbol{u}_i\}$, we have $\boldsymbol{U}^\top \boldsymbol{R} = \boldsymbol{I}$.

As our goal is to propose a recurrent unit that takes the simple form $\boldsymbol{b}_t = f(\boldsymbol{b}_{t-1}, \boldsymbol{x}_t)$, where $\boldsymbol{b}_t, \boldsymbol{b}_{t-1} \in \mathbb{R}^d, \boldsymbol{x}_t \in \mathbb{R}^{d'}$ and $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$. Thus, we consider a reduced version of TPRs, in which $\{\boldsymbol{r}_i\}, \{\boldsymbol{u}_i\} \subset \mathbb{R}^{d \times 1}$ then fillers become a set of scalars instead of vectors, and the binding and unbinding operations are implemented by simple matrix-vector multiplications as follows:

$$\boldsymbol{b}_t = \boldsymbol{R}\boldsymbol{f}_t \text{ ,where } \boldsymbol{f}_t = h(\boldsymbol{U}^\top \boldsymbol{b}_{t-1}, \boldsymbol{U}^\top \boldsymbol{x}_t) \tag{1}$$

The implementation of the function $h : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$ is crucial as its dynamics determines the gradient flow and the learning process of the proposed TPRU, therefore a careful design is essential. Sections are organised following the workflow below:

• We present a gated recurrent unit, the **TPRU**, along with a set of analysis on the gradient flow to justify our decisions in architecture design. Even though the number of nonlinear operations is reduced, the derivative of the proposed unit demonstrates the flexibility of our TPRU in learning.

• The instantiation of reduced TPRs allows our model to learn decomposable representations, and the filler $\boldsymbol{f}_t$ provides an interface for us to inspect the information encoded in $\boldsymbol{b}_t$. We conducted a linguistically grounded study, and it shows that our TPRU is capable of distinguishing multiple senses of a polysemy and discovering structured information in sentences.

• Our proposed TPRU demonstrates strong performance on the Logical Entailment task (Evans et al., 2018), Natural Language Inference datasets (Williams et al., 2017), and language modelling, and it has comparable parameters to GRU, and significantly fewer than the widely-used LSTM.

2 Related Work

Recent efforts on understanding RNNs can be categorised into three categories: 1) theoretical analysis through advancements in dynamic systems on the functions learnt by RNN units (Chang et al., 2019), 2) linguistically grounded analysis by developing sets of probing tasks to illustrate the properties of learnt vector representations of RNNs (Conneau et al., 2018; Chrupaa & Alishahi, 2019), and 3) introducing inductive biases into the architecture of networks and comparing the behaviour of learnt models against those of widely-used counterparts. Our work falls in this last category: learning structured representations by incorporating the inductive biases inherent in TPRs.

Some prior work has incorporated TPRs into RNN representations. Question-answering on SQuAD (Rajpurkar et al., 2016) was addressed (Palangi et al., 2017) with an LSTM with hidden state given by a single TPR binding; the present work deploys complexes containing multiple bindings. TPR-style unbinding was applied in caption generation (Huang et al., 2018), but the representations were deep-learnt and not explicitly designed to be TPRs as in the present work. A contracted version of TPRs, Holographic Reduced Representations, was utilised to decompose input space and output space with filler-role decomposition (Luo et al., 2019). Our work differs from these prior papers in that TPR filler-role binding and unbinding operations are explicitly carried out in our proposed recurrent unit, and the two operations directly determine the update of the hidden states. Closest to the present work is Schlag & Schmidhuber (2018), which performed question-answering by using TPRs to embed simple knowledge graphs as tensors; our model does not deploy graph-based inference, which is less natural for the inference tasks we address than it is for the question-answering task addressed in Schlag & Schmidhuber (2018).

Our proposed TPRU inherits the explicit binding and unbinding operations in TPRs in a sense that a set of role vectors are shared across time steps and sequences to serve as basis for composing distributed vector representations. We provide a set of analyses from both theoretical and empirical perspectives, in which gradient analysis of our model helps us make sense of the architecture design and empirical results from three different categories of tasks show the effectiveness of the proposed model, and additional study on the linguistic and learning aspects provide evidence that our model indeed inherits the merits of TPRs, which enhance the interpretability and the capability of learning structured representations.

3 Proposed Recurrent Unit: The TPRU

As shown in both LSTM (Hochreiter & Schmidhuber, 1997) and GRU (Chung et al., 2014) design, a gating mechanism helps the hidden state at the current time step to directly copy information from the previous time step, and alleviate vanishing and exploding gradient issues. Our proposed recurrent unit deploys a single gate, adopting the input gate of the GRU, omitting the reset gate. We expect that further augmenting our TPRU to 2 gates as in the GRU, or 3 gates as in the LSTM, will lead to yet better performance.

Following Eq. 1, at each time step, the TPRU receives two input vectors, $\boldsymbol{b}_{t-1} \in \mathbb{R}^{d\times 1}$ and $\boldsymbol{x}_t \in \mathbb{R}^{d'\times 1}$, and produces $\boldsymbol{b}_t \in \mathbb{R}^{d\times 1}$. An input gate $\boldsymbol{g}_t$ is computed to take a weighted sum of the information produced at current time step $\widetilde{\boldsymbol{b}}_t$ and the previous binding complex $\boldsymbol{b}_{t-1}$,

$$\boldsymbol{b}_t = \boldsymbol{g}_t \circ \widetilde{\boldsymbol{b}}_t + (1 - \boldsymbol{g}_t) \circ \boldsymbol{b}_{t-1}, \text{ where } \boldsymbol{g}_t = \sigma(\boldsymbol{W}_b \boldsymbol{b}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t), \tag{2}$$

$\sigma(\cdot)$ is the logistic sigmoid function, $\circ$ is the Hadamard (element-wise) product, and $\boldsymbol{W}_b \in \mathbb{R}^{d\times d}$ and $\boldsymbol{W}_x \in \mathbb{R}^{d\times d'}$ are matrices of learnable parameters. As we now explain, the calculation of $\widetilde{\boldsymbol{b}}_t$ is carried out by the unbinding and binding operations of TPRs (Eq. 1).

3.1 UNBINDING OPERATION

Consider a set of hypothesised unbinding vectors $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_N] \in \mathbb{R}^{d\times N}$. At time step t, these can be used to unbind fillers $\boldsymbol{f}_{b,t}$ from the previous binding complex $\boldsymbol{b}_{t-1}$ using Eq. 1. We posit two matrices $\boldsymbol{V}_x \in \mathbb{R}^{d\times d'}$ and $\boldsymbol{V}_b \in \mathbb{R}^{d\times d}$ that transform the current input $\boldsymbol{x}_t \in \mathbb{R}^{d'\times 1}$ and the binding complex $\boldsymbol{b}_{t-1}$ into the binding space $\mathbb{R}^{d\times 1}$ yielding fillers $\boldsymbol{f}_{x,t}$ and $\boldsymbol{f}_{b,t}$:

$$\boldsymbol{f}_{b,t} = \boldsymbol{U}^\top \boldsymbol{V}_b \boldsymbol{b}_{t-1} \in \mathbb{R}^{N\times 1}, \qquad \boldsymbol{f}_{x,t} = \boldsymbol{U}^\top \boldsymbol{V}_x \boldsymbol{x}_t \in \mathbb{R}^{N\times 1}. \tag{3}$$

Normalisation is applied for stable learning by gradient descent, and the specific technique (Yang et al., 2018) we adopted here also enforces sparsity on both $\boldsymbol{f}_{b,t}$ and $\boldsymbol{f}_{x,t}$:

$$(\boldsymbol{f}_t)_n = (\widetilde{\boldsymbol{f}}_t)_n^2 / \textstyle\sum_{m=1}^{N} (\widetilde{\boldsymbol{f}}_t)_m^2 \tag{4}$$

$$\text{where } (\widetilde{\boldsymbol{f}}_t)_n = \max(((\boldsymbol{f}_{b,t})_n + b_b), 0) + \max(((\boldsymbol{f}_{x,t})_n + b_x), 0) \tag{5}$$

in which b_b and b_x are two scalar parameters for stable learning, and $(\cdot)_n$ refers to the n-th entry of the vector in the parentheses.

3.2 BINDING OPERATION

Given a hypothesised set of binding role vectors $\boldsymbol{R} = [\boldsymbol{r}_1, \boldsymbol{r}_2, ..., \boldsymbol{r}_N] \in \mathbb{R}^{d\times N}$, applying the binding operation in Eq. 1 to the fillers $\boldsymbol{f}_t$ at time t gives the candidate update $\widetilde{\boldsymbol{b}}_t$ for the binding complex,

$$\widetilde{\boldsymbol{b}}_t = \boldsymbol{R}\boldsymbol{f}_t. \tag{6}$$

The gating mechanism controls the weighted sum of the candidate vector $\widetilde{\boldsymbol{b}}_t$ and the previous binding complex $\boldsymbol{b}_{t-1}$ to produce a binding complex $\boldsymbol{b}_t$ at current time step, as given by Eq. 2.

3.3 UNBINDING AND BINDING ROLE VECTORS

In the TPRU, there is a matrix of role vectors $\boldsymbol{R}$ used for the binding operation and a matrix of unbinding vectors $\boldsymbol{U}$ used for the unbinding operation. To control the number of parameters in our TPRU, instead of directly learning two sets of vectors, a set of random vectors $\boldsymbol{E}$ is sampled from a normal distribution, and two linear projections $\boldsymbol{W}_r, \boldsymbol{W}_u \in \mathbb{R}^{d\times d}$ are learnt to transform $\boldsymbol{E}$ to $\boldsymbol{U} = \boldsymbol{W}_u\boldsymbol{E}$ and $\boldsymbol{R} = \boldsymbol{W}_r\boldsymbol{E}$.

Therefore, in total, our proposed TPRU has six learnable matrices: $\boldsymbol{W}_u$, $\boldsymbol{W}_r$, $\boldsymbol{V}_b$, $\boldsymbol{W}_b \in \mathbb{R}^{d\times d}$ and $\boldsymbol{V}_x$, $\boldsymbol{W}_x \in \mathbb{R}^{d\times d'}$. The number of parameter matrices the same as that of GRU and less than that of LSTM, and, when $d' < d$, the number of learnable parameters is less than that of GRU.

4 GRADIENT ANALYSIS AND COMPLEXITY

This section provides an analysis on the derivatives in our proposed TPRU, and aims to justify the architecture design. For simplicity, we omit the derivatives w.r.t. bias terms in our derivations as they can be absorbed into the input vector by adding an extra dimension. The derivative of the output at current time step $\boldsymbol{b}_t$ (Eq. 2) w.r.t. the output of the previous time step $\boldsymbol{b}_{t-1}$ is

$$\frac{\partial \boldsymbol{b}_t}{\partial \boldsymbol{b}_{t-1}} = \frac{\partial \boldsymbol{b}_t}{\partial \boldsymbol{g}_t}\frac{\partial \boldsymbol{g}_t}{\partial \boldsymbol{b}_{t-1}} + \frac{\partial \boldsymbol{b}_t}{\partial \widetilde{\boldsymbol{b}}_t}\frac{\partial \widetilde{\boldsymbol{b}}_t}{\partial \boldsymbol{b}_{t-1}} + \frac{\partial \boldsymbol{b}_t}{\partial \boldsymbol{b}_{t-1}} = \boldsymbol{\Lambda}_1\boldsymbol{W}_b + \boldsymbol{\Lambda}_2\frac{\partial \widetilde{\boldsymbol{b}}_t}{\partial \boldsymbol{b}_{t-1}} + \boldsymbol{\Lambda}_3 \tag{7}$$

where $\boldsymbol{\Lambda}_1$, $\boldsymbol{\Lambda}_2$ and $\boldsymbol{\Lambda}_3$ are diagonal matrices in $\mathbb{R}^{d\times d}$, in which $(\boldsymbol{\Lambda}_1)_{ii} = (\widetilde{\boldsymbol{b}}_t)_i(\boldsymbol{g}_t)_i(1-(\boldsymbol{g}_t)_i)$, $(\boldsymbol{\Lambda}_2)_{ii} = (\boldsymbol{g}_t)_i$ and $(\boldsymbol{\Lambda}_3)_{ii} = (1-(\boldsymbol{g}_t)_i)$. It is clear that the gating mechanism smoothes the gradient flow as it jitters the diagonal terms in the Jacobian matrix, and now we focus on the following derivative which carries the main information of our proposed TPRU.

$$\frac{\partial \widetilde{\boldsymbol{b}}_t}{\partial \boldsymbol{b}_{t-1}} = \frac{\partial \widetilde{\boldsymbol{b}}_t}{\partial \boldsymbol{f}_t}\frac{\partial \boldsymbol{f}_t}{\partial \boldsymbol{f}_{b,t}}\frac{\partial \boldsymbol{f}_{b,t}}{\partial \boldsymbol{b}_{t-1}} = (\boldsymbol{W}_r\boldsymbol{E})\,(\boldsymbol{J}_t\boldsymbol{N}_t\boldsymbol{S}_{b,t})\,(\boldsymbol{E}^\top\boldsymbol{W}_u^\top\boldsymbol{V}_b) \tag{8}$$

where $\boldsymbol{N}_t, \boldsymbol{S}_{b,t} \in \mathbb{R}^{N\times N}$ are diagonal matrices, in which $(\boldsymbol{N}_t)_{nn} = 2/(\widetilde{\boldsymbol{f}}_t)_n$ and $(\boldsymbol{S}_{b,t})_{nn} = \mathbb{1}_{(\boldsymbol{f}_{b,t})_n>0}$, and $(\boldsymbol{J}_t)_{mn} = (\boldsymbol{f}_{b,t})_m(\delta_{mn} - (\boldsymbol{f}_{b,t})_n)$. Appendix A shows that $\boldsymbol{J}_t$ is a positive semi-definite matrix, and eigenvalues are all smaller than $\max_n(\boldsymbol{f}_{b,t})_n \leq 1$. The following analysis shows that certain modifications and simplifications could lead to issues during learning.

Simplification I. $\boldsymbol{W}_r = \boldsymbol{W}_u$, $\boldsymbol{V}_b = \boldsymbol{V}_x = \boldsymbol{I}_d$ and softmax normalisation in Eq. 4.

The derivative of the softmax normalisation of outputs w.r.t.the inputs is simply $\boldsymbol{J}_t$ as $\boldsymbol{S}_{b,t} = \boldsymbol{N}_t = \boldsymbol{I}_N$.[1] Given the conditions, the derivative can be simplified to the one shown below:

$$\partial\widetilde{\boldsymbol{b}}_t/\partial\boldsymbol{b}_{t-1} = \boldsymbol{W}_r\boldsymbol{E}\boldsymbol{J}_t\boldsymbol{N}_t\boldsymbol{S}_{b,t}\boldsymbol{E}^\top\boldsymbol{W}_u^\top\boldsymbol{V}_b = \boldsymbol{W}_r\boldsymbol{E}\boldsymbol{J}_t\boldsymbol{E}^\top\boldsymbol{W}_r^\top \succeq 0 \tag{9}$$

In this case, as the derivative gets multipled several times by backpropagation, the gradient w.r.t.each parameter matrix either explodes or vanishes depending on the spectrum of $\boldsymbol{W}_r\boldsymbol{E}$. Also, a careful initialisation is required otherwise the gradient could overflow or underflow after first iteration.

Simplification II. $\boldsymbol{W}_r = \boldsymbol{W}_u$, and softmax normalisation.

Here we allow learnable parameter matrices $\boldsymbol{V}_b$ and $\boldsymbol{V}_x$ to transform the binding complex from the previous time step $\boldsymbol{b}_{t-1}$ and the input vector from the current time step $\boldsymbol{x}_t$ first respectively, thus, the derivative of $\widetilde{\boldsymbol{b}}_t$ w.r.t. $\boldsymbol{b}_{t-1}$ and that to i-th row of $\boldsymbol{V}_b$ are shown below:

$$\partial\widetilde{\boldsymbol{b}}_t/\partial\boldsymbol{b}_{t-1} = \boldsymbol{W}_r\boldsymbol{E}\boldsymbol{J}_t\boldsymbol{E}^\top\boldsymbol{W}_r^\top\boldsymbol{V}_b \qquad \partial\widetilde{\boldsymbol{b}}_t/\partial(\boldsymbol{V}_b)_{i\cdot} = (\boldsymbol{W}_r\boldsymbol{E}\boldsymbol{J}_t\boldsymbol{E}^\top\boldsymbol{W}_r^\top)_{\cdot i}\boldsymbol{b}_{t-1} \tag{10}$$

Since $\boldsymbol{W}_r\boldsymbol{E}\boldsymbol{J}_t\boldsymbol{E}^\top\boldsymbol{W}_r^\top \succeq 0$, $\partial\widetilde{\boldsymbol{b}}_t/\partial(\boldsymbol{V}_b)_{i\cdot}$ always has a non-negative inner product with $\boldsymbol{b}_{t-1}$, which means that the update of $\boldsymbol{V}_b$ always contains a positive portion of $\boldsymbol{b}_{t-1}$. The same situation also applies when updating $\boldsymbol{V}_x$ as it keeps a positive portion of $\boldsymbol{x}_t$ as well. It limits the learning system from eliminating accumulated useless information. Although it is still possible that, after backpropagation through several time steps, the positive portion of $\boldsymbol{b}_{t-1}$ could be suppressed, we set $\boldsymbol{W}_r$ and $\boldsymbol{W}_u$ to be independent matrices so that the above constraint doesn't occur.

Modification I. Softmax normalisation.

The only modification here is to replace the normalisation step in Eq. 4 in our TPRU to a softmax function that also normalises $\widetilde{\boldsymbol{f}}_t$ to form a distribution $\boldsymbol{f}_t$. At each time step, the derivative $\widetilde{vb}_t$ w.r.t. any parameter matrix before the normalisation step contains $\partial\boldsymbol{f}_t/\partial\widetilde{\boldsymbol{f}}_t$, and in the softmax normalisation, only the relative difference, $(\widetilde{\boldsymbol{f}}_t)_n - \max_m(\widetilde{\boldsymbol{f}}_t)_m, \forall n \in \{1, 2, ..., N\}$, matters.

Therefore, there is no explicit regularisation in the derivative of $\partial\widetilde{\boldsymbol{b}}_t/\partial\boldsymbol{b}_{t-1} = \boldsymbol{W}_r\boldsymbol{E}\boldsymbol{J}_t\boldsymbol{E}^\top\boldsymbol{W}_u^\top\boldsymbol{V}_b$ to penalise extreme values in $\widetilde{\boldsymbol{f}}_t$, which could lead to gradients with large norms and make the learning process unstable. It also explains in the transformer module (Vaswani et al., 2017), layer normalisation (Ba et al., 2016) is necessary as it scales the gradient by the variance of the input.

We don't rule out the possibility that there exists a suitable initialisation which gives stable learning and decent performance, but the normalisation step we adopted from prior work (Yang et al., 2018) avoids heavy tuning of initialisations for stable learning.

Ours. Here, we recall that the derivative of $\partial\widetilde{\boldsymbol{b}}_t/\partial\boldsymbol{b}_{t-1}$ is in Eq. 8. The term $\boldsymbol{N}_t$ scales the derivative by the magnitude of the input $\widetilde{\boldsymbol{f}}_t$ to Eq. 4, which regularises the norm of the gradient to each parameter matrix. Specifically, the following derivative indicates the update of i-th row in $\boldsymbol{W}_u$:

$$\frac{\partial\widetilde{\boldsymbol{b}}_t}{\partial(\boldsymbol{W}_u)_{i\cdot}} = \frac{\partial\widetilde{\boldsymbol{b}}_t}{\partial\boldsymbol{f}_{b,t}}\frac{\partial\boldsymbol{f}_{b,t}}{\partial(\boldsymbol{W}_u)_{i\cdot}} + \frac{\partial\widetilde{\boldsymbol{b}}_t}{\partial\boldsymbol{f}_{x,t}}\frac{\partial\boldsymbol{f}_{x,t}}{\partial(\boldsymbol{W}_u)_{i\cdot}} = \boldsymbol{W}_r\boldsymbol{E}\boldsymbol{J}_t\boldsymbol{N}_t\left(\boldsymbol{S}_{b,t}(\boldsymbol{V}_b\boldsymbol{b}_{t-1})_i + \boldsymbol{S}_{x,t}(\boldsymbol{V}_x\boldsymbol{x}_t)_i\right)\boldsymbol{E}^\top \tag{11}$$

where $\boldsymbol{N}_t$ normalises the contribution from $\boldsymbol{f}_{b,t}$ and $\boldsymbol{f}_{x,t}$, and $\boldsymbol{S}_{b,t}$ and $\boldsymbol{S}_{x,t}$ select positive contributions only, thus the normalisation step in Eq. 4 helps the gradients w.r.t. parameter matrices to stay stable. Eq. 11 also shows that the update of each row in $\boldsymbol{W}_u$ depends on a sparse linear transformation of $\boldsymbol{W}_r$, which could potentially avoid overfitting as not all dimensions in the representation space are utilised and our TPRU learns to discover the representation space.

Given the derivative above, it is easy to see that the presampled matrix $\boldsymbol{E}$ acts similarly as the preconditioning matrix in solving large linear systems, and with samples from a normal distribution or zero-centred uniform distribution, it smooths the spectrum of the Jacobian matrix and stables the learning system.

[1] I_N is an identity matrix in $\mathbb{R}^{N\times N}$.

Time Complexity in Forward Computation

Table 1: **Complexity**

Model	Time Complexity	# Params
TPRU	$O(2Nd + 6d^2)$	$4d^2 + 2dd'$
LSTM	$O(8d^2)$	$4d^2 + 4dd'$
GRU	$O(6d^2)$	$3d^2 + 3dd'$

Table 1 presents the time complexity of calculating each of our proposed TPRU, LSTM and GRU for a single time step. In the calculation, we only kept the terms that involve second order of variables. As shown, the time complexity of TPRU is in between LSTM and GRU when the number of role vectors N is less than the dimension of the recurrent unit d. However, our TPRU has fewer parameters than a LSTM, and similar to a GRU.

In addition, since our TPRU contains only N times of $\max(x, 0)^2$ and d times of sigmoid functions, which is fewer than $5d$ times of sigmoid/tanh squashing functions in an LSTM and $3d$ times in a GRU. The nonlinear operations are indeed reduced in our TPRU, and the following sections show that our TPRU still provides strong performance on evaluation tasks.

5 Tasks and Training Details

Three entailment tasks, including an abstract logical entailment task (Evans et al., 2018) and two relatively realistic natural language entailment task, including Multi-genre Natural Language Inference (MNLI, Williams et al. (2017)) and Question Natural Language Inference (QNLI, Wang et al. (2018)), as well as a language modelling task on a large corpus and a small one, which are WikiText-103 and WikiText-2 (Merity et al., 2018), are considered to demonstrate the learning ability of our proposed TPRU. Each of the three entailment tasks provides pairs of samples, and for each pair, the model needs to tell whether the first (the premise) entails the second (the hypothesis). Language modelling is used to demonstrate the ability of our proposed model on sequential modelling.

Experiments are conducted in PyTorch (Paszke et al., 2017) with the Adam optimiser (Kingma & Ba, 2014) and gradient clipping (Pascanu et al., 2013). Reported results are averaged from the results of three random initialisations. Our proposed TPRU has only one input gate and also has similar number of parameters, so the GRU is the most relevant comparison partner. LSTM is also included as it is widely-used as well and it generally yields better performance than GRU does. For each model, hyperparameters are searched based on the small selected validation set on each task, including dropout rates, learning rate and batch size. For entailment tasks, we tested out both plain and bidaf architecture (Seo et al., 2017). Specifically, on both NLI tasks, ELMo (Peters et al., 2018) encoding is applied for embedding words into contextualised vectors. More details about the tasks and the training details can be found in Section B and C in Appendix.

Table 2: **Accuracy of each model on the propositional Logical Entailment task.** ***Overall, our proposed TPRU outperforms its closest comparison counterpart GRU, and provides comparable performance to the LSTM.*** The value besides 'TPRU' indicates the number of role vectors. Results of BiDAF models are parenthesised. 'Mean $2^{\#\text{Vars}}$' is the mean number of worlds of proposition-values; here and below, **bold** indicates the best result among all models with the same architecture.

model	valid	test					# params
		easy	hard	big	massive	exam	
Mean $2^{\#\text{Vars}}$	75.7	81.0	184.4	3310.8	848,570.0	5.8	
Plain (BiDAF) Architecture - dim 64							
GRU	84.3 (**91.3**)	68.2 (76.0)	84.3 (91.3)	77.0 (80.0)	57.4 (**70.4**)	66.8 (74.3)	49.1k (172.4k)
TPRU - 512	**88.6** (90.9)	73.1 (**77.18**)	**88.4** (**90.6**)	**79.0** (**82.0**)	62.0 (69.9)	71.8 (**74.9**)	98.3k (213.0k)
LSTM	**88.6** (89.1)	**76.3** (75.4)	**88.4** (89.1)	**79.0** (75.0)	**64.7** (66.7)	**76.4** (**74.9**)	131.1k (327.7k)

6 Discussion

On the Logical Entailment task in Table 2, when it comes to larger dimensionality, our TPRU appears to be more stable than GRU during learning as we observed that the GRU with the BiDAF architecture failed to converge in two out of three trials. On certain test sets, our (singly-gated)

TPRU slightly underperforms the LSTM, possibly due to the three gates (and more parameters) of the LSTM: future research could incorporate more sophisticated gating mechanisms into the TPRU.

On the MNLI tasks in Table 3, our proposed TPRU consistently outperforms both the LSTM and GRU when a large number of role vectors is deployed. Unexpectedly, all models perform better on the mismatched dev set than on the matched one, perhaps because it is easier.

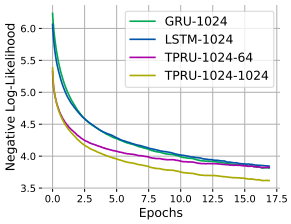
Table 3: **Results on MNLI and QNLI.** ***Our TPRU outperforms LSTM.*** The number of role vectors in each of our models is indicated by the value beside 'TPRU'. As shown, our proposed TPRU overall outperforms LSTM on two tasks.

model	MNLI - dev		QNLI - dev	# params
	matched	mismatched		
Plain (BiDAF) Architecture - dim 512				
TPRU - 256	74.2 (77.7)	74.9 (78.0)	78.2 (81.2)	
TPRU - 512	**74.8** (**78.1**)	**75.6** (77.8)	78.8 (80.1)	
TPRU - 1024	74.4 (77.7)	74.8 (**78.5**)	78.8 (80.8)	8.4m (16.8m)
LSTM	73.4 (76.5)	73.9 (77.0)	77.2 (79.1)	12.6m (27.3m)
Plain (BiDAF) Architecture - dim 1500				
TPRU - 512	74.5 (77.9)	75.4 (**78.8**)	79.3 (**82.5**)	
TPRU - 1024	74.8 (77.9)	**76.1** (78.3)	**80.4** (82.2)	
TPRU - 2048	**75.1** (**78.7**)	75.2 (**78.8**)	80.0 (**82.5**)	48.3m (108.4m)
LSTM	74.8 (77.1)	75.7 (77.7)	78.8 (81.2)	60.6m (162.9m)
SOTA (ALICE ensemble)	88.2	87.9	95.7	

On the language modelling task on WikiText-103 and WikiText-2, our TPRU provides similar perplexity on the test set to the LSTM, as shown in Table 4. The learning plot in Figure 4 shows that our TPRU learns faster on the training set than the GRU and the LSTM. Given the performance on the dev set and the test set, our TPRU also generalises better. The learning plot and the performance table both serve as evidental support for the gradient analysis of our model.

Table 4: **Perplexity of language modelling on WikiText-2 and WikiText-103.** (Lower is better.) **(left):** Our TPRU with 1024 role vectors gives similar perplexity score as the widely used LSTM does. **(right):** Negative Log-likelihood on training set of language modelling task on WikiText-103. (The plotted models are trained without dropout layers.) Our TPRU converges faster on the training set than the GRU and LSTM. However, GRU doesn't generalise very well on the dev test, so we omitted the results from GRU in the table.

model	WikiText-103	WikiText-2	#Params
Two-layer Architecture - dim 1024			
TPRU	41.4	76.8	12.6m
LSTM	42.3	76.0	16.8m
SOTA	16.4 (Krause et al., 2019)	39.1 (Gong et al., 2018)	257m / 35m



6.1 POLYSEMY

In TPRs (Smolensky, 1990), multiple senses of a single polysemous word can be encoded by different role vectors $\{\boldsymbol{r}_i\}$, and by binding them with the same symbol that represents the polysemous word through outer product, TPR is composing a distributed representation that keeps multiple senses of a polysemous word and recovering each sense with no information loss if a large number of dimensions is available. In our case, the magnitude of each entry in $\boldsymbol{f}_t$ implies the contribution of each role vector to the update candidate $\widetilde{\boldsymbol{b}}_t$.

We train our TPRU with a BiDAF architecture on MNLI dataset, and word vectors are obtained from fastText (Bojanowski et al., 2017) as the input to our model so that the vector representation of a polysemous word stays the same even when it is presented in different contexts. After learning, our model provides an interface for us to check whether a specific role vector correlates to a specific sense of a polysemous word in different phrases in the dev set by computing $\beta = \arg\max_n(\boldsymbol{f}_t)_n$.

For example, a common polysemous word bank has two senses when it is used as a noun. In the dev set of MNLI, the only case when bank is referred in the river-context is in the phrase "*bank fishing*", and the rest belong to the other sense in the context of economy. We observed that β is the same when bank occurs in "*bank fishing*" and we denote it as β_{river}, thus, $P(\beta = \beta_{\text{river}}|\text{bank in the river-context}) = 1$. It is also important to see if the specific β_{river} is dedicated to the word bank in the river context. After examinating the dev set, we have $P(\text{bank in the river-context}|\beta = \beta_{\text{river}}) = P(\text{bank in "}\textit{the west bank}\text{"}|\beta = \beta_{\text{river}}) = 0.5$, where the phrase "*the west bank*" refers to the landlocked territory. As there is no clear learning signal to specify "*the west bank*" as a named entity, given the context, our proposed TPRU learns to categorise it into the river-context rather than the economy-context which is a more reasonable decision than the other way around.

6.2 POS TAGGING

We calculated the pointwise mutual information (PMI) between the following two terms: 1) The maximally selected role vector at each time step $\beta_t = \arg\max_n(\boldsymbol{f}_t)_n$ and 2) The part-of-speech tag (POS Tags) provided by a pretrained tagger.[2] For a given POS tag, the PMIs of role vectors are sorted in a descending order, and Table 5 presents role vectors that have the top two highest PMI values if the gap between them is larger than 0.5, which has 7 out of 17 fine-grained POS tags.

Table 5: **PMI between POS Tags and Maximally Selected Role Vectors.**

POS Tags	ADV	DET	INTJ	NOUN	PART	PROPN	VERB
Top 2 PMI	7.37 / 5.83	6.06 / 5.41	7.13 / 6.46	7.23 / 5.84	7.54 / 6.91	7.27 / 6.47	7.22 / 6.28

As the model is only trained on MNLI to predict the entailment relationship between a given pair of sentences without access to POS tags, it is interesting that some role vectors become associated with specific POS tags. The POS tags provide structured information of a given sentence, and since our learnt TPRU implicitly demonstrates the property of being informative about the POS tags, we can conclude that our model is capable of learning structured representations.

As pointed out in prior work, function words in MNLI play an important role in determining the entailment relationship between a sentence pair, thus POS tags that are related to function words, including NOUN, PROPN, VERB, etc., have a larger gap between the top 2 highest PMI values as they are emphasised by the learning signal. This finding implies that our learnt model could indicate the biases in datasets which would be helpful in improving the fairness and inclusiveness.

The above two empirical observations point out that our proposed TPRU inherits the interpretability of the original TPRs, and provides a certain level of human-readability. It provides a simple interface for us to analyse the learnt model through the explicit binding and unbinding operations.

6.3 EFFECT OF INCREASING THE NUMBER OF ROLE VECTORS

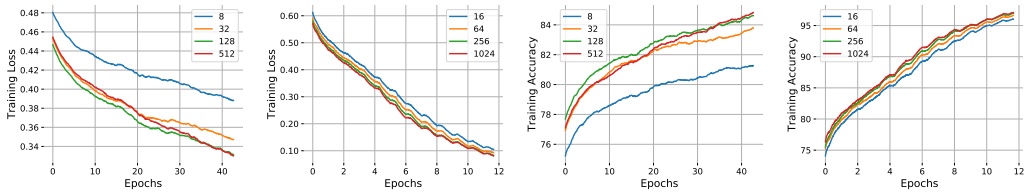
In the original TPRs, the number of role vectors indicates the number of unique symbols that will be used in the final representations of data. Since each symbol is capable of representing a specific substructure of the input data, increasing the number of role vectors eventually leads to more highly structured representations if there is no limit on the dimensionality of role vectors.

Experiments are conducted to show the effect of increasing the number of role vectors on the performance on both Logical Entailment and MNLI. As shown in Table 3, adding more role vectors into our proposed TPRU gradually improves the performance on the two entailment tasks.

Figure 1 presents the learning curves, including training loss and accuracy, of our proposed TPRU with varying number of role vectors on the two entailment tasks. As shown in the graphs, incor-

[2] https://spacy.io/api/doc

porating more role vectors leads to not only better performance, but also faster convergence during training. The observation is consistent on both the Logical Entailment and MNLI tasks.



(a) Logical Entailment (b) MNLI (c) Logical Entailment (d) MNLI

Figure 1: **Training Plots of our TPRU with different number of role vectors**. (The plotted models are trained without dropout layers.) In general, our proposed recurrent unit converges faster and leads to better results on both tasks when more role vectors are available, but the total number of parameters remains the same. (Best viewed in colour.)

Interestingly, on the Logical Entailment task, the performance of our proposed TPRU with only eight role vectors matches that of a GRU with the same dimensionality. The number of role vectors required in our TPRU to match the performance of a GRU and a LSTM on MNLI task is much less than the dimension of the hidden states, which suggests that the distributed vector representations learnt in both the LSTM and GRU are highly redundant and can be reduced to fewer dimensions; this also implies that the LSTM and GRU are not able to extensively exploit the representation space even when the dimensionality of the model is larger the that required for solving the task. Meanwhile, our proposed TPRU explores sparse weighted combinations of role vectors in Eq. 4, and it helps the model to specialise individual role vectors for different purposes, which eventually leads to a more effective exploitation of the representation space.

7 Conclusion

We have proposed a simple recurrent unit that leverages the explicit binding and unbinding operations in previously proposed TPRs in a reduced fashion, therefore it is named the TPRU. The explicit execution arms the proposed model with the ability to learn decomposable representations, and by sharing the role vectors across time steps and sequences, our TPRU provides research an intuitive interface through $\boldsymbol{f}_t$ to study the information encoded in the learnt distributed vector representations.

Since recurrent neural networks are prone to suffer from the gradient vanishing and exploding issues, we conduct a thorough analysis on the derivatives and the gradient update of our proposed TPRU within each time step. The conducted analysis supports our decisions in designing TPRU, and it shows that it encourages relatively more stable gradient flow through BPTT. The time complexity for executing one step of our TPRU is lower than that of LSTM, and it has same or a smaller number of parameters than the GRU does, which justifies the simplicity of our recurrent unit.

The empirical evaluation on five learning tasks in three different categories demonstrate the effectiveness of our TPRU as it performs on a par with or better than the LSTM. The ablation study shows that our model converges faster than GRU and LSTM in general and it is a result of the stable gradient flow through multiple time steps. In addition, our proposed TPRU enables a faster convergence rate and better performance by increasing the number of role vectors.

A study on correlating the role vectors with linguistic properties of languages shows that our TPRU inherits the strong interpretability of TPRs. In the toy example involving a polysemous word bank, our model can differentiate multiple senses of the word, and can assign an ambiguous sense to its closely related one. Also, the PMI values between POS tags and maximally selected role vectors imply that our TPRU is capable of discovering syntactically-structured linguistic information.

Given the current pressing need for interpretable machine learning and learning decomposable representations, our proposed TPRU can be regarded as a more light-weight and human-readable alternative to the widely-used LSTM or GRU. Future work will focus on incorporating full TPRs into the design of an RNN unit and into the Transformer module.

REFERENCES

Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *TACL*, 5:135–146, 2017.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *EMNLP*, 2015.

Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryxepo0cFX`.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *ACL*, 2017.

Grzegorz Chrupaa and Afra Alishahi. Correlating neural and symbolic representations of language. In *ACL*, 2019.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Alexis Conneau, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. In *ACL*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. Can neural networks understand logical entailment? In *ICLR*, 2018.

ChengYue Gong, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. Frage: Frequency-agnostic word representation. In *NeurIPS*, 2018.

Sepp Hochreiter and Juergen Schmidhuber. Long short-term memory. *Neural Computation*, 9: 1735–1780, 1997.

Qiuyuan Huang, Paul Smolensky, Xiaodong He, Li Deng, and Dapeng Oliver Wu. Tensor product generation networks for deep NLP modeling. In *NAACL-HLT*, 2018.

Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *CoRR*, abs/1611.01462, 2016.

Sekitoshi Kanai, Yasuhiro Fujiwara, and Sotetsu Iwamura. Preventing gradient explosions in gated recurrent units. In *NIPS*, 2017.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *ArXiv*, abs/1506.02078, 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. Dynamic evaluation of transformer language models. *CoRR*, abs/1904.08378, 2019.

Jiaming Luo, Yuan Cao, and Yonghui Wu. Towards decomposed linguistic representation with holographic reduced representation, 2019. URL `https://openreview.net/forum?id=rkl3-hA5Y7`.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2018.

Hamid Palangi, Paul Smolensky, Xiaodong He, and Li Deng. Deep learning of grammatically-interpretable representations through question-answering. *CoRR*, abs/1705.08432, 2017.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke S. Zettlemoyer. Deep contextualized word representations. In *NAACL-HLT*, 2018.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *EACL*, 2017.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *CoRR*, 2018.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*, 2016.

Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In *International Conference on Artificial Neural Networks*, pp. 632–640. Springer, 2006.

Imanol Schlag and Jürgen Schmidhuber. Learning to reason with third order tensor products. In *Advances in Neural Information Processing Systems*, pp. 9981–9993, 2018.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, 2016.

Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. 2017.

Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artif. Intell.*, 46:159–216, 1990.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018.

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

Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *CoRR*, abs/1704.05426, 2017.

Zhilin Yang, Junbo Jake Zhao, Bhuwan Dhingra, Kaiming He, William W. Cohen, Ruslan Salakhutdinov, and Yann LeCun. Glomo: Unsupervisedly learned relational graphs as transferable representations. *CoRR*, abs/1806.05662, 2018.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *ArXiv*, abs/1311.2901, 2013.

A PROOF

1. $\boldsymbol{J}_t$ is a PSD matrix. For any non-zero vector $\boldsymbol{v}$, we have

$$\boldsymbol{v}^\top \boldsymbol{J}_t \boldsymbol{v} = \sum_{n=1}^{N} (\boldsymbol{f}_{b,t})_n \boldsymbol{v}_n^2 - (\sum_{n=1}^{N} (\boldsymbol{f}_{b,t})_n \boldsymbol{v}_n)^2 \tag{12}$$

$$\geq (\sum_{n=1}^{N} (\boldsymbol{f}_{b,t})_n \boldsymbol{v}_n)^2 - (\sum_{n=1}^{N} (\boldsymbol{f}_{b,t})_n \boldsymbol{v}_n)^2 = 0 \qquad \text{(Jensen's Inequality)} \tag{13}$$

Therefore, $\boldsymbol{J}_t$ is a PSD matrix, so is $\boldsymbol{J}_t$ as $\boldsymbol{J}_t = \boldsymbol{J}_t$.

B DETAILS ABOUT TASKS

Three entailment tasks, including an abstract logical entailment task and two relatively realistic natural language entailment task, as well as a language modelling task on a large corpus and a small one, are considered to demonstrate that the learning ability of our proposed TPRU. Each of the three entailment tasks provides pairs of samples, and for each pair, the model needs to tell whether the first (the premise) entails the second (the hypothesis). Language modelling is used to demonstrate the ability of our proposed model on sequential modelling.

As our goal is to learn structured vector representations, the proposed TPRU serves as an encoding function, which learns to process a proposition or sentence one token at a time, and produce a vector representation. During learning, two vector representations are produced by the same recurrent unit given a pair of samples, then a simple feature engineering method (e.g. concatenation of the two representations) is applied to form an input vector for subsequent classification. In general, with a simple classifier, e.g. a linear classifier or a multi-layer perceptron with a single hidden layer, the learning process forces the encoding function to produce high-quality representations of samples, either propositions or sentences; better representations enable stronger performance.

B.1 LOGICAL ENTAILMENT (PROPOSITIONAL LOGIC)

In propositional logic, for a pair of propositions, A and B, the value of $A \vDash B$ is independent of the identities of the shared variables between A and B, and is dependent only on the structure of the expression and the connectives in each subexpression. Because of α-equivalence, $p \wedge q \vDash q$ holds no matter how we replace variable p or q with any other propositional variables.[3] Logical entailment is thus a good test-bed for evaluating a model's ability to carry out abstract, highly structure-sensitive reasoning (Evans et al., 2018).

Theoretically, it is possible to construct a truth table that contains as rows/worlds all possible combinations of values of variables in both propositions A and B; the value of $A \vDash B$ can be checked by going through every entry in each row. As the logical entailment task emphasises reasoning on connectives, it requires the learnt distributed vector representations to encode the structure of any given proposition to excel at the task.

The Logical Entailment dataset[4] has balanced positive ($A \vDash B$) and negative ($A \nvDash B$) classes and a representative validation set. Five test sets are provided to evaluate the generalisation ability at different difficulty levels: some test sets have significantly more variables and operators than the training and validation sets (see Table 2).

B.2 NATURAL LANGUAGE INFERENCE

— determining whether a premise sentence entails a hypothesis sentence — combine the structure-dependence of logical entailment with the NLP tasks of inferring word meaning in context as well as the hierarchical relations among sentence constituents. Two datasets are considered, which are Multi-genre NLI and QNLI. (Williams et al., 2017; Wang et al., 2018). Compared to logical entailment, the inference and reasoning in NLI also rely on the identities of words in sentences in

[3] For example, $p \wedge q \vDash q \Leftrightarrow a \wedge b \vDash a$.

[4] https://github.com/deepmind/logical-entailment-dataset

addition to their structure. More importantly, the ambiguity and polysemicity of language lead to the impossibility of creating a truth table that lists all cases. Therefore, NLI is an intrinsically hard task.

The Multi-genre Natural Language Inference (MNLI) dataset (Williams et al., 2017) collects sentence pairs in ten genres; only five genres are available in the training set, while all ten genres are presented in the development set. QNLI (Wang et al., 2018) is collected from Stanford Question Answering dataset (SQuAD) (Rajpurkar et al., 2016), on which a model needs to predict whether the context sentence contains the answer to the question sentence or not. The relation between sentences in a pair must be classified as Entailment, Neutral or Contradiction.

The performance of a model on the five 'mismatched' genres that exist in the development but not the training set reflects how well the structure encoded in distributed vector representations of sentences learnt from genres seen in training generalises to sentence pairs in unseen genres. As the nature of NLI tasks requires inferring both word meaning and structure of constituents in a given sentence, supervised training signals from labelled datasets enforce an encoding function to analyse meaning and structure at the same time during learning, which eventually forces distributed vector representations of sentences produced from the learnt encoding function to be structured. To the extent that the inferences required in an NLI task are demanding of proper analysis of the logical structure implicit in sentence, an inductive bias that enhances the ability to learn to encode the structures of sentences should enhance success on the MNLI task.

B.3 Language Modelling

Language modelling is a fundamental task to test generalisation and sequential modelling. Recent research has shown that models trained on language modelling tasks yield representations that can transfer well to other natural language tasks (Peters et al., 2018; Radford et al., 2018; Devlin et al., 2018). We follow the autoregressive setting of decomposing the joint probability on tokens into a product of conditional probabilities on generating the current token given tokens in the left context.

$$P(w_1, w_2, ..., w_L) = \prod_{l=1}^{L} P(w_l | w_{l-1}, w_{l-2}, ..., w_1). \tag{14}$$

This requires the learnt representation to capture the potentially highly structured information accumulated prior to the current time step in order to make a confident prediction of the current token. A proper inductive bias towards constructing structured representations should enhance performance on the task. Two corpora used in our experiment are WikiText-103 and WikiText-2 (Merity et al., 2018), in which one contains 103 million tokens in the training set while the other only has 2 million. The results will tell us how well our proposed TPRU generalises across different training scales.

C Details about Architecture

C.1 Plain Architecture - Inference:

For both the Logical Entailment task and the MNLI task, we train two-layer recurrent networks built using our proposed TPRU, networks built using its direct comparison counterpart, the GRU, as well as LSTMs. A global max-pooling over time is applied to the hidden states produced by each recurrent unit at all time steps to generate the vector representation for a given logical proposition or sentence. Given a pair of generated representations $\boldsymbol{u}$ and $\boldsymbol{v}$, a vector is constructed to represent the difference between two vectors $[\boldsymbol{u}; \boldsymbol{v}; |\boldsymbol{u} - \boldsymbol{v}|; \boldsymbol{u} \circ \boldsymbol{v}]$, where $\boldsymbol{u} \circ \boldsymbol{v}$ is the Hadamard product and $|\boldsymbol{u} - \boldsymbol{v}|$ is the absolute difference, and the vector is fed into a multi-layer perceptron which has only one hidden layer with ReLU activation function. The feature engineering and the choice of classifier are suggested by prior work (Seo et al., 2017; Wang et al., 2018).

Symbolic Vocabulary Permutation (Evans et al., 2018) is applied as data augmentation during learning on the logical entailment task: exploiting α-equivalence, this systematically replaces the variables in a proposition pair with randomly sampled alternative variables, as only connectives matter on this task. Table 2 presents the results. For the MNLI task, the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015) is included as additional training data as recommended (Wang et al., 2018), and ELMo (Peters et al., 2018) is applied for producing word vectors. The best

model is chosen according to the averaged classification accuracy on the five matched and five mismatched dev sets. The results are presented in Table 3. SOTA models come from Alibaba DAMO NLP Group, and the results are available on the GLUE leaderboard. [5]

C.2 BiDAF Architecture - Inference:

Bi-directional Attention Flow (**BiDAF**) (Seo et al., 2017) has been successfully applied to various NLP tasks (Seo et al., 2017; Chen et al., 2017), and provides strong performance on NLI tasks (Wang et al., 2018). The BiDAF architecture contains a layer for encoding two input sequences, and another for encoding the concatenation of the output from the first layer and the context vectors determined by the bi-directional attention mechanism. In our experiments, the dimensions of both layers are set to be the same, and the same type of recurrent unit is applied across both layers. The same settings are used for experiments on LSTM and our TPRU models. Specifically, for TPRU, the recurrent units in both layers have the same number of role vectors. Other learning details are as in the plain architecture. Tables 2 and 3 respectively present results on the Logical Entailment and the MNLI task, with BiDAF results in parentheses.

C.3 Language Modelling:

On the language modelling task, the same two-layer architecture (Merity et al., 2018) is adopted in our experiments on LSTM and our proposed TPRU. For WikiText-103, Bytepair encoding (BPE) vocabulary with 100k merges (10k for WikiText-2) are applied to encode the corpus into indices (Sennrich et al., 2016). The output prediction layer is tied to the input word embedding layer as advised in prior work (Press & Wolf, 2017; Inan et al., 2016). Each model is trained for 64 epochs on the training set; the best one is selected based on perplexity on the validation set, and then evaluated on the test set. Table 4 shows the best performance of each model after the hyperparameter search.

D Polysemies

In the dev set of MNLI, the word <u>bank</u> occurs in the following sentences:

1. according to this plan , areas that were predominantly arab the gaza strip , the central part of the country , the northwest corner , and <u>the west bank</u> were to remain under arab control as palestine , while the southern negev desert and the northern coastal strip would form the new state of israel .
2. <u>the other bank</u> pays the fund interest based upon tiered account levels , more typical of a large commercial account .
3. well uh normally i like to to go out fishing in a boat and uh rather than like <u>bank fishing</u> and just like you try and catch anything that 's swimming because i 've had such problems of trying to catch any type of fish that uh i just really enjoy doing the boat type fishing .

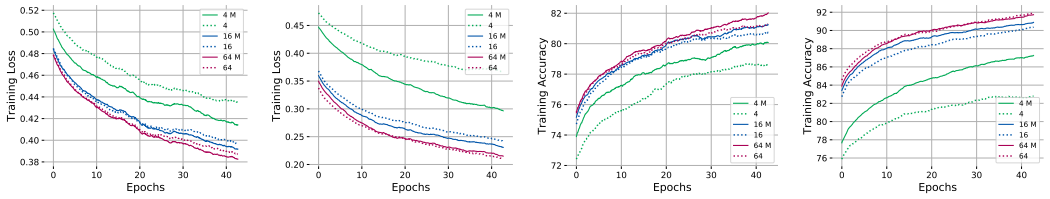
The above sentences are three examples, there are other sentences in which the word <u>bank</u> occurs. As we show in the main paper, after learning, our TPRU categories the first two cases as the same sense, as there is no clear learning signal that specifies the second one as a named entity, and categories the third sentence into another sense.

E Effect of Increasing the Dimensionality of Fillers/Symbols

Both fillers and roles in TPRs (Smolensky, 1990) are tensors with arbitrary dimensions, which empowers the representation ability of TPRs. In our case, as we aim to produce a distributed vector representation of the given input sequence rather than a tensor representation, the fillers are reduced to filler numbers, in which case, the symbol represented by each filler is limited to one-dimensional vector.

In our proposed TPRU, the unbinding operation is instantiated by the dot-product (inner-product) operation between the binding complex and the unbinding vectors. One plausible way to upgrade

[5] https://gluebenchmark.com/leaderboard



(a) Plain Architecture (b) BiDAF Architecture (c) Plain Architecture (d) BiDAF Architecture

Figure 2: **Training Plots for comparing the single- and eight-slice dot-product operations of our TPRU, with both plain and BiDAF architectures.** (The plotted models are trained without dropout layers.) Overall, the eight-slice dot product, which enhances the fillers from one- to eight-dimensional vectors, leads to faster convergence rate and better performance on the training set when the number of role vectors is limited, and its effect diminishes when the number of role vectors reaches a large value. '4' and '4 M' respectively refer to our TPRU with four role vectors and single- or eight-slice dot-products. (Best viewed in colour.)

filler numbers to filler vectors is to split the binding complex into multiple slices, and split each unbinding vector accordingly: then the dot-product is carried out on each pair of slices rather than the entire vector. In this case, the dimension of the resulting filler vectors is determined by the number of slices predefined prior to learning, and is limited by the dimension of the binding complex. The normalisation step is operated on the same dimension on multiple filler vectors independently and in parallel, which results in multiple normalised probability distributions. Each binding role vector is split into the same number of slices, and each distribution is applied to bind the same slices from all binding role vectors into a single slice. All slices are concatenated together to form a single vector which has the same dimensionality as the input binding complex. A similar idea was proposed as the multi-head attention mechanism in prior work and has been shown to be effective in terms of performance gain without additional computational cost (Vaswani et al., 2017). It was widely adopted in prior work on Transformer networks for various tasks.

In our implementation of the Logical Entailment task, the binding complex is split into eight slices, each with the same dimension. Figure 2 shows training logs with eight slices and also with only one slice for three different numbers of role vectors in our proposed TPRU. In general, shifting from filler numbers to filler vectors in our TPRU provides faster convergence and better performance on the training set. However, as the number of role vectors increases, the improvement gained decreases, which leads to an interesting hypothesis that the effect of increasing number of role vectors might be correlated to that of increasing the dimensionality of filler vectors.