SEQUENCE-LEVEL FEATURES: HOW GRU AND LSTM CELLS CAPTURE N-GRAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern recurrent neural networks (RNN) such as Gated Recurrent Units (GRU) and Long Short-term Memory (LSTM) have demonstrated impressive results on tasks involving sequential data in practice. Despite continuous efforts on interpreting their behaviors, the exact mechanism underlying their successes in capturing sequence-level information have not been thoroughly understood. In this work, we present a study on understanding the essential features captured by GRU/LSTM cells by mathematically expanding and unrolling the hidden states. Based on the closed-form approximations of the hidden states, we argue that the effectiveness of the cells may be attributed to a type of sequence-level representations brought in by the gating mechanism, which enables the cells to encode sequence-level features along with token-level features. Specifically, we show that under certain mild assumptions, the essential components of the cells would consist of such sequence-level features similar to those of N-grams. Based on such a finding, we also found that replacing the standard cells with approximate hidden state representations does not necessarily degrade performance on the sentiment analysis and language modeling tasks, indicating such features may play a significant role for GRU/LSTM cells. We hope that our work can give inspiration on proposing new neural architectures for capturing contextual information within sequences.

1 INTRODUCTION

Long Short-term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Chung et al., 2014) are widely used and investigated for tasks that involve sequential data. They are generally believed to be capable of capturing long-range dependencies while being able to alleviate gradient vanishing or explosion issues (Hochreiter & Schmidhuber, 1997; Karpathy et al., 2015; Sutskever et al., 2014). While such models were empirically shown to be successful across a range of tasks, certain fundamental questions such as "what essential features are GRU or LSTM cells able to capture?" have not yet been fully addressed. Lacking answers to them may limit our ability in designing better architectures.

One obstacle can be attributed to the non-linear activations used in the cells that prevent us from obtaining explicit closed-form expressions for hidden states. One possible solution is to expand the non-linear functions using the Taylor series (Arfken & Mullin, 1985) and represent hidden states with explicit input terms. Literally, each hidden state can be viewed as the combination of constituent terms capturing features of different levels of complexity. However, there is a prohibitively large number of polynomial terms involved and they can be difficult to manage. Inspired by saturated RNNs (Chandar et al., 2019; Merrill et al., 2020) that focused on the scenarios with large inputs in terms of magnitude, we focused on the alternative scenario where the magnitude of the inputs were sufficiently small, thereby making high-order terms in the series negligible. By unrolling GRU/LSTM and expanding the hidden states, we found that the gating mechanisms play a role in creating sequence-level representations through a form of matrix-vector multiplications, and enabled hidden states to capture both token-level and sequence-level features.

Although analyzed under the assumption of sufficiently small inputs, we hypothesize that GRU or LSTM cells yield similar behaviors in real scenarios in terms of capturing such sequence-level features. We observed that the token-level and sequence-level representations derived from a GRU or LSTM cell were able to reflect different properties on negation phrases for the sentiment analysis

tasks. Furthermore, in both the sentiment analysis and language modeling tasks, we replaced the GRU or LSTM cell with corresponding approximate hidden state representations directly during training, and found that such models behaved similarly to the standard GRU or LSTM based models. This indicated that the sequence-level features might be significant for GRU or LSTM cells.

2 Related Work

There have been plenty of prior works aiming to explain the behaviors of RNNs along with the variants. Early efforts were focused on exploring the external behaviors of recurrent neural networks (RNNs). Li et al. (2015) proposed a visualization approach to analyze intermediate representations of the LSTM-based models, certain interesting patterns could be observed. However, it might not be easy to extend to models with high-dimension representations. Greff et al. (2016) explored the performances of LSTM variants on representative tasks such as speech recognition, handwriting recognition, and argued that none of the proposed variants could significantly improve upon the standard LSTM architecture. Karpathy et al. (2015) studied the existence of interpretable cells that could track long-range dependencies such as line lengths, quotes and brackets. However, those works did not involve the internal mechanism of GRUs or LSTMs.

Later, we witnessed further efforts to understand the internal behaviors of RNNs. Arras et al. (2017) applied an extended technique Layer-wise Relevance Propagation (LRP) to a bidirectional LSTM for sentiment analysis, produced reliable explanations of which words are responsible for attributing sentiment in individual text. Murdoch et al. (2018) leverage contextual decomposition methods to conduct analysis on the interactions of terms for LSTMS, which could produce importance scores for words, phrases and word interactions. A RNN unrolling technique was proposed by Sherstinsky (2018) based on signal processing concepts, transforming the RNN into the "Vanilla LSTM" network through a series of logical arguments, and Kanai et al. (2017) discussed the conditions that could prevent gradient explosions by looking into the dynamics of GRUs. Merrill et al. (2020) examined the properties of saturated RNNs and linked the update behaviors to weighted finite-state machines. Their works gave us more inspirations to explore RNNs from different perspectives.

Moreover, Melis et al. (2020) and Krause et al. (2017) found that creating richer interaction between contexts and inputs on top of standard LSTMs could result in improvements. Their efforts actually pointed out the significance of rich interactions between inputs and contexts for LSTMs. We sought to find out and understand the essential underlying features captured by GRU and LSTM cells.

3 Model Definitions

GRU The representation of a GRU cell can be written as [1]:

$$\begin{aligned} \boldsymbol{r}_t &= \sigma(\boldsymbol{W}_{ir}\boldsymbol{x}_t + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}), \\ \boldsymbol{z}_t &= \sigma(\boldsymbol{W}_{iz}\boldsymbol{x}_t + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1}), \\ \boldsymbol{n}_t &= \tanh(\boldsymbol{W}_{in}\boldsymbol{x}_t + \boldsymbol{r}_t \odot \boldsymbol{W}_{hn}\boldsymbol{h}_{t-1}), \\ \boldsymbol{h}_t &= (1 - \boldsymbol{z}_t) \odot \boldsymbol{n}_t + \boldsymbol{z}_t \odot \boldsymbol{h}_{t-1}, \end{aligned} \tag{1}$$

where $\boldsymbol{h}_t \in \mathbb{R}^d$, $\boldsymbol{x}_t \in \mathbb{R}^{d_x}$ are the hidden state and input at time step t respectively, $\boldsymbol{h}_{t-1}$ is the hidden state of the layer at time $(t-1)$ or the initial hidden state. $\boldsymbol{r}_t \in \mathbb{R}^d$, $\boldsymbol{z}_t \in \mathbb{R}^d$, $\boldsymbol{n}_t \in \mathbb{R}^d$ are the reset, update, and the new gates respectively. $\boldsymbol{W}_{\star\star}$ refers to a weight matrix. σ is the element-wise sigmoid function, and $\odot$ is the element-wise Hadamard product.

LSTM The representation of a LSTM cell can be written as:

$$\begin{aligned} \boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ii}\boldsymbol{x}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}), \\ \boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{if}\boldsymbol{x}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1}), \\ \boldsymbol{o}_t &= \sigma(\boldsymbol{W}_{io}\boldsymbol{x}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}), \\ \boldsymbol{c}'_t &= \tanh(\boldsymbol{W}_{ic}\boldsymbol{x}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1}), \\ \boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{c}'_t, \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t), \end{aligned} \tag{2}$$

[1] For brevity, we suppressed the bias for both GRU and LSTM cells here.

where $\boldsymbol{h}_t \in \mathbb{R}^d$, $\boldsymbol{x}_t \in \mathbb{R}^{d_x}$ are the hidden state and input at time step t respectively, $\boldsymbol{i}_t$, $\boldsymbol{f}_t$, $\boldsymbol{o}_t \in \mathbb{R}^d$ are the input gate, forget gate, output gate respectively. $\boldsymbol{c}'_t \in \mathbb{R}^d$ is the new memory, $\boldsymbol{c}_t$ is the final memory. $\boldsymbol{W}_{\star\star}$ refers to a weight matrix.

4 UNROLLING GRU/LSTM

Using the Taylor series, the activations $\tanh(x)$ and $\sigma(x)$ can be expanded (at 0) as:

$$\tanh(x) = x + O(x^3), |x| < \frac{\pi}{2}, \tag{3}$$

$$\sigma(x) = \frac{1}{2} + \frac{1}{4}x + O(x^3), |x| < \pi. \tag{4}$$

When the input is far smaller than 1, the first-order Taylor series can approximate the sigmoid and $\tanh$ functions well. In this work, we do not seek to approximate the GRU or LSTM cells precisely, but to explore what features the portion of the first-order Taylor series can possibly capture.

4.1 GRU

Let us write $\boldsymbol{x}_t^r = \boldsymbol{W}_{ir}\boldsymbol{x}_t$, $\boldsymbol{x}_t^z = \boldsymbol{W}_{iz}\boldsymbol{x}_t$, $\boldsymbol{x}_t^n = \boldsymbol{W}_{in}\boldsymbol{x}_t$, $\boldsymbol{h}_{t-1}^r = \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}$, $\boldsymbol{h}_{t-1}^z = \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1}$, $\boldsymbol{h}_{t-1}^n = \boldsymbol{W}_{hn}\boldsymbol{h}_{t-1}$. Based on Equations 3 and 4, the gates and hidden states of a GRU cell can be described as:

$$\boldsymbol{r}_t = \frac{1}{2} + \frac{1}{4}(\boldsymbol{x}_t^r + \boldsymbol{h}_{t-1}^r) + O\left[(\boldsymbol{x}_t^r + \boldsymbol{h}_{t-1}^r)^3\right], \tag{5}$$

$$\boldsymbol{z}_t = \frac{1}{2} + \frac{1}{4}(\boldsymbol{x}_t^z + \boldsymbol{h}_{t-1}^z) + O\left[(\boldsymbol{x}_t^z + \boldsymbol{h}_{t-1}^z)^3\right], \tag{6}$$

$$\boldsymbol{n}_t = \boldsymbol{x}_t^n + \boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}^n + O\left[(\boldsymbol{x}_t^n + \boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}^n)^3\right]. \tag{7}$$

With these equations, we can expand the hidden state at time step t, then combine like terms with respect to the *order* of $\boldsymbol{h}_{t-1}$ and represent it as:

$$\begin{aligned}
\boldsymbol{h}_t = & \underbrace{\frac{1}{2}\boldsymbol{x}_t^n - \frac{1}{4}\boldsymbol{x}_t^n \odot \boldsymbol{x}_t^z}_{zeroth-order} \\
& + \underbrace{\frac{1}{2}\boldsymbol{h}_{t-1} + \frac{1}{4}\boldsymbol{h}_{t-1}^n + \frac{1}{4}\boldsymbol{x}_t^z \odot \boldsymbol{h}_{t-1} - \frac{1}{4}\boldsymbol{x}_t^n \odot \boldsymbol{h}_{t-1}^z + \frac{1}{8}(\boldsymbol{x}_t^r - \boldsymbol{x}_t^z) \odot \boldsymbol{h}_{t-1}^n - \frac{1}{16}\boldsymbol{x}_t^r \odot \boldsymbol{x}_t^z \odot \boldsymbol{h}_{t-1}^n}_{first-order} \\
& + \underbrace{\frac{1}{4}\boldsymbol{h}_{t-1}^z \odot \boldsymbol{h}_{t-1} + \frac{1}{8}(\boldsymbol{h}_{t-1}^r - \boldsymbol{h}_{t-1}^z) \odot \boldsymbol{h}_{t-1}^n - \frac{1}{16}\boldsymbol{x}_t^r \odot \boldsymbol{h}_{t-1}^z \odot \boldsymbol{h}_{t-1}^n - \frac{1}{16}\boldsymbol{x}_t^z \odot \boldsymbol{h}_{t-1}^r \odot \boldsymbol{h}_{t-1}^n}_{second-order} \\
& - \underbrace{\frac{1}{16}\boldsymbol{h}_{t-1}^z \odot \boldsymbol{h}_{t-1}^r \odot \boldsymbol{h}_{t-1}^n}_{third-order} + \boldsymbol{\xi}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}),
\end{aligned} \tag{8}$$

where $\boldsymbol{\xi}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$ refers to the higher-order terms of $\boldsymbol{x}_t$, $\boldsymbol{h}_{t-1}$ as well as their interactions. Equation 8 shows that there are many interaction terms in the hidden state.

The higher-order terms of $\boldsymbol{h}_{t-1}$ such as $\boldsymbol{h}_{t-1}^r \odot \boldsymbol{h}_{t-1}^z$ in Equation 8 can produce many new complex high-order terms during unrolling. We assume that they are insignificant, and will focus on the terms involving zeroth-order and first-order terms of $\boldsymbol{h}_{t-1}$. Then the hidden state at time step t can be written as:

$$\begin{aligned}
\boldsymbol{h}_t = & \frac{1}{2}\boldsymbol{x}_t^n - \frac{1}{4}\boldsymbol{x}_t^n \odot \boldsymbol{x}_t^z \\
& + \frac{1}{2}\boldsymbol{h}_{t-1} + \frac{1}{4}\boldsymbol{h}_{t-1}^n + \frac{1}{4}\boldsymbol{x}_t^z \odot \boldsymbol{h}_{t-1} - \frac{1}{4}\boldsymbol{x}_t^n \odot \boldsymbol{h}_{t-1}^z + \frac{1}{8}(\boldsymbol{x}_t^r - \boldsymbol{x}_t^z - \frac{1}{2}\boldsymbol{x}_t^z \odot \boldsymbol{x}_t^r) \odot \boldsymbol{h}_{t-1}^n \\
& + \boldsymbol{\xi}'(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}),
\end{aligned} \tag{9}$$

where $\boldsymbol{\xi}'(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$ refers to the high-order terms of $\boldsymbol{h}_{t-1}$ plus $\boldsymbol{\xi}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$. We can transform the Hadamard products into matrix-vector multiplications ($\boldsymbol{a} \odot \boldsymbol{b} = \text{diag}(\boldsymbol{a})\boldsymbol{b}$) and obtain the following :

$$\begin{aligned}\boldsymbol{h}_t &= \frac{1}{2}\boldsymbol{x}_t^n - \frac{1}{4}\boldsymbol{x}_t^n \odot \boldsymbol{x}_t^z \\ &+ \left[\frac{1}{2}\boldsymbol{I} + \frac{1}{4}\boldsymbol{W}_{hn} + \frac{1}{4}\text{diag}(\boldsymbol{x}_t^z) - \frac{1}{4}\text{diag}(\boldsymbol{x}_t^n)\boldsymbol{W}_{hz} + \frac{1}{8}\text{diag}(\boldsymbol{x}_t^r - \boldsymbol{x}_t^z - \frac{1}{2}\boldsymbol{x}_t^z \odot \boldsymbol{x}_t^r)\boldsymbol{W}_{hn}\right]\boldsymbol{h}_{t-1} \\ &+ \boldsymbol{\xi}'(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}).\end{aligned} \tag{10}$$

For brevity, let us define two functions of $\boldsymbol{x}_t$:

$$\begin{aligned}\boldsymbol{g}(\boldsymbol{x}_t) &= \frac{1}{2}\boldsymbol{x}_t^n - \frac{1}{4}\boldsymbol{x}_t^n \odot \boldsymbol{x}_t^z, \\ \boldsymbol{A}(\boldsymbol{x}_t) &= \frac{1}{2}\boldsymbol{I} + \frac{1}{4}\boldsymbol{W}_{hn} + \frac{1}{4}\text{diag}(\boldsymbol{x}_t^z) - \frac{1}{4}\text{diag}(\boldsymbol{x}_t^n)\boldsymbol{W}_{hz} + \frac{1}{8}\text{diag}(\boldsymbol{x}_t^r - \boldsymbol{x}_t^z - \frac{1}{2}\boldsymbol{x}_t^z \odot \boldsymbol{x}_t^r)\boldsymbol{W}_{hn}.\end{aligned} \tag{11}$$

Note that both $\boldsymbol{g}(\boldsymbol{x}_t)$ and $\boldsymbol{A}(\boldsymbol{x}_t)$ are only functions of $\boldsymbol{x}_t$. Then we can rewrite Equation 10 as:

$$\boldsymbol{h}_t = \boldsymbol{g}(\boldsymbol{x}_t) + \boldsymbol{A}(\boldsymbol{x}_t)\boldsymbol{h}_{t-1} + \boldsymbol{\xi}'(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}). \tag{12}$$

Throughout all the previous time steps (assuming the initial state are 0s), the approximate hidden state at time step t can be finally unrolled as:

$$\begin{aligned}\boldsymbol{h}_t &= \boldsymbol{g}(\boldsymbol{x}_t) + \sum_{i=1}^{t-1} \underbrace{\boldsymbol{A}(\boldsymbol{x}_t)\boldsymbol{A}(\boldsymbol{x}_{t-1})...\boldsymbol{A}(\boldsymbol{x}_{i+1})}_{\boldsymbol{M}_{(i+1):t}} \boldsymbol{g}(\boldsymbol{x}_i) + \boldsymbol{\epsilon}(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_t) \\ &= \boldsymbol{g}(\boldsymbol{x}_t) + \sum_{i=1}^{t-1} \underbrace{\boldsymbol{M}_{(i+1):t}\boldsymbol{g}(\boldsymbol{x}_i)}_{\boldsymbol{\Phi}_{i:t}} + \boldsymbol{\epsilon}(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_t),\end{aligned} \tag{13}$$

where $\boldsymbol{M}_{(i+1):t} = \prod_{k=t}^{i+1} \boldsymbol{A}(\boldsymbol{x}_k) \in \mathbb{R}^{d \times d}$ is the matrix-matrix product from time step t to $i+1$, $\boldsymbol{\epsilon}(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_t)$ are the unrolled representations from the higher-order terms in Equation 12. The function $\boldsymbol{g}(\boldsymbol{x}_t)$ solely encodes current input, namely token-level features, we call it *token-level representation*. The matrix-vector product $\boldsymbol{\Phi}_{i:t} = \boldsymbol{M}_{(i+1):t}\boldsymbol{g}(\boldsymbol{x}_i)$ encodes the tokens starting from time step i and ending at time step t. If the matrices are different and not diagonal, any change of the order will result in a different product. Therefore, $\boldsymbol{\Phi}_{i:t}$ is able to capture the feature of the token sequence between time step i and t in an *order-sensitive* manner. We call it a *sequence-level feature*. Such features are calculated sequentially from the left to the right through a sequence of vector/matrix multiplications, leading to features reminiscent of the classical N-grams commonly used in natural language processing (NLP). Let us use $\hat{\boldsymbol{h}}_t$ to denote the approximation of $\boldsymbol{h}_t$ (by keeping the first two terms in Equation 13) as:

$$\hat{\boldsymbol{h}}_t = \boldsymbol{g}(\boldsymbol{x}_t) + \sum_{i=1}^{t-1} \boldsymbol{\Phi}_{i:t}. \tag{14}$$

The approximate hidden state representation $\hat{\boldsymbol{h}}_t$ at time step t is able to encode current token input and all the token sequences starting from time step $i \in \{1, 2, ..., t-1\}$ and ending at time step t given an instance. In other words, it is a linear combination of current token-level input feature (can be understood as the *unigram* feature) and sequence-level features of all the possible N-grams ending at time step t. Bidirectional GRUs would be able to capture sequence level features from both directions.

4.2 LSTM

Let us write $\boldsymbol{x}_t^i = \boldsymbol{W}_{ii}\boldsymbol{x}_t$, $\boldsymbol{x}_t^f = \boldsymbol{W}_{if}\boldsymbol{x}_t$, $\boldsymbol{x}_t^o = \boldsymbol{W}_{io}\boldsymbol{x}_t$, $\boldsymbol{x}_t^c = \boldsymbol{W}_{ic}\boldsymbol{x}_t$. Similarly, for a LSTM cell, assuming the inputs are sufficiently small, we can expand the memory cell and the hidden state in a similar way. We also focus on the terms that involve the zeroth-order and first-order $\boldsymbol{c}_{t-1}$ as well as

$\boldsymbol{h}_{t-1}$, and write the final memory cell and hidden state together as:

$$\begin{bmatrix}\boldsymbol{c}_t\\ \boldsymbol{h}_t\end{bmatrix} = \begin{bmatrix}\boldsymbol{g}_c(\boldsymbol{x}_t)\\ \boldsymbol{g}_h(\boldsymbol{x}_t)\end{bmatrix} + \begin{bmatrix}\boldsymbol{B}(\boldsymbol{x}_t) & \boldsymbol{D}(\boldsymbol{x}_t)\\ \boldsymbol{E}(\boldsymbol{x}_t) & \boldsymbol{F}(\boldsymbol{x}_t)\end{bmatrix}\begin{bmatrix}\boldsymbol{c}_{t-1}\\ \boldsymbol{h}_{t-1}\end{bmatrix} + \begin{bmatrix}\boldsymbol{\xi}'_c(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1})\\ \boldsymbol{\xi}'_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1})\end{bmatrix}. \tag{15}$$

where:

$$\begin{aligned}
\boldsymbol{g}_c(\boldsymbol{x}_t) &= \frac{1}{4}(\boldsymbol{x}_t^i + \boldsymbol{2}) \odot \boldsymbol{x}_t^c,\ \boldsymbol{B}(\boldsymbol{x}_t) = \frac{1}{4}\text{diag}(\boldsymbol{x}_t^f + \boldsymbol{2}),\\
\boldsymbol{D}(\boldsymbol{x}_t) &= \frac{1}{4}\text{diag}(\boldsymbol{x}_t^i + \boldsymbol{2})\boldsymbol{W}_{hc} + \frac{1}{4}\text{diag}(\boldsymbol{x}_t^c)\boldsymbol{W}_{hi},\\
\boldsymbol{g}_h(\boldsymbol{x}_t) &= \frac{1}{4}(\boldsymbol{x}_t^o + \boldsymbol{2}) \odot \boldsymbol{g}_c(\boldsymbol{x}_t),\ \boldsymbol{E}(\boldsymbol{x}_t) = \frac{1}{4}\text{diag}(\boldsymbol{x}_t^o + \boldsymbol{2})\boldsymbol{B}(\boldsymbol{x}_t),\\
\boldsymbol{F}(\boldsymbol{x}_t) &= \frac{1}{4}\text{diag}(\boldsymbol{x}_t^o + \boldsymbol{2})\boldsymbol{D}(\boldsymbol{x}_t) + \frac{1}{4}\text{diag}(\boldsymbol{g}_c(\boldsymbol{x}_t))\boldsymbol{W}_{ho}.
\end{aligned} \tag{16}$$

$\boldsymbol{\xi}'_c(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1})$ and $\boldsymbol{\xi}'_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1})$ are the higher-order terms.

Let us use the matrix $\boldsymbol{A}(\boldsymbol{x}_t) \in \mathbb{R}^{2d\times 2d}$ to denote $\begin{bmatrix}\boldsymbol{B}(\boldsymbol{x}_t) & \boldsymbol{D}(\boldsymbol{x}_t)\\ \boldsymbol{E}(\boldsymbol{x}_t) & \boldsymbol{F}(\boldsymbol{x}_t)\end{bmatrix}$, then the equation above can be written as:

$$\begin{bmatrix}\boldsymbol{c}_t\\ \boldsymbol{h}_t\end{bmatrix} = \begin{bmatrix}\boldsymbol{g}_c(\boldsymbol{x}_t)\\ \boldsymbol{g}_h(\boldsymbol{x}_t)\end{bmatrix} + \boldsymbol{A}(\boldsymbol{x}_t)\begin{bmatrix}\boldsymbol{c}_{t-1}\\ \boldsymbol{h}_{t-1}\end{bmatrix} + + \begin{bmatrix}\boldsymbol{\xi}'_c(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1})\\ \boldsymbol{\xi}'_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1})\end{bmatrix}. \tag{17}$$

And the final memory cell and hidden state can be unrolled as:

$$\begin{bmatrix}\boldsymbol{c}_t\\ \boldsymbol{h}_t\end{bmatrix} = \begin{bmatrix}\boldsymbol{g}_c(\boldsymbol{x}_t)\\ \boldsymbol{g}_t(\boldsymbol{x}_t)\end{bmatrix} + \sum_{i=1}^{t-1}\underbrace{\left[\prod_{k=t}^{i+1}\boldsymbol{A}(\boldsymbol{x}_k)\right]\begin{bmatrix}\boldsymbol{g}_c(\boldsymbol{x}_i)\\ \boldsymbol{g}_h(\boldsymbol{x}_i)\end{bmatrix}}_{\boldsymbol{\Phi}_{i:t}} + \boldsymbol{\epsilon}(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_t). \tag{18}$$

The matrix-vector product $\boldsymbol{\Phi}_{i:t} \in \mathbb{R}^{2d}$ for the token sequence between time step i and t will be viewed as the sequence-level representation. Similar properties can be inferred. Analogously, we will use $\hat{\boldsymbol{c}}_t$ and $\hat{\boldsymbol{h}}_t$ to approximate $\boldsymbol{c}_t$ and $\boldsymbol{h}_t$ by retaining the first two terms in Equation 18.

5 Experiments

We employed binary sentiment analysis and language modeling tasks to examine whether the sequence-level and token-level representations could capture different properties on negation phrases and explore the capabilities of architectures. All the experiments were conducted on models with one GRU or LSTM layer. The statistics of the datasets are shown in Appendix A.1.

5.1 Interpreting Sequence-level Representations

We first trained the model with the standard GRU or LSTM, then used the learned parameters to calculate the token-level and sequence-level representations for the tokens and sequences of a given instance. We examine the sequence-level and token-level features on instances involving negations, which is a typical language phenomenon. For each experiment, we performed multiple trials and found similar patterns could be observed for each model with different initializations. Thus, only one set of the results were reported for each model.

5.1.1 Polarity Score

We would like to understand how well such learned sequence-level features are able to capture polarity information associated with word sequences within a sentence. We follow the work of Sun & Lu (2020), and examined two types of "polarity scores" to quantify such polarity information, *token-level polarity score* and *sequence-level polarity score*. Such scores are able to capture the degree of association between a token or a sequence and a specific label. There are four layers in sequence for our sentiment analysis: embedding layer, GRU/LSTM layer, linear layer and sigmoid layer. For a GRU cell, the two scores can be calculated as:

$$s_t^g = \boldsymbol{w}^\top \boldsymbol{g}(\boldsymbol{x}_t),\ s_{i:t}^{\boldsymbol{\Phi}} = \boldsymbol{w}^\top \boldsymbol{\Phi}_{i:t}. \tag{19}$$

Table 1: Polarity score distribution for tokens, negation phrases and double negation phrases.

Type		GRU Polarity Score	LSTM Polarity Score
positive tokens (10)	tokens	4.25± 0.13	4.79± 0.27
	negation phrases	-6.13± 0.15	-9.32± 0.31
	double negation phrases	12.45± 0.33	21.39± 0.74
negative tokens (12)	tokens	-3.72± 0.06	-4.49± 0.24
	negation phrases	4.82± 0.11	3.13 ±0.27
	double negation phrases	-11.29± 0.23	-7.65± 0.66

For a LSTM cell, the sequence-level representation can be split into two parts: $\mathbf{\Phi}_{i:t} = [\mathbf{\Phi}^c_{i:t}, \mathbf{\Phi}^h_{i:t}]$ ($\mathbf{\Phi}^c_{i:t}, \mathbf{\Phi}^h_{i:t} \in \mathbb{R}^d$). The polarity scores will be calculated as:

$$s^g_t = \boldsymbol{w}^\top \boldsymbol{g}_h(\boldsymbol{x}_t),\ s^{\mathbf{\Phi}}_{i:t} = \boldsymbol{w}^\top \mathbf{\Phi}^h_{i:t}. \tag{20}$$

where $\boldsymbol{w} \in \mathbb{R}^d$ is the linear layer weight vector, s^g_t is the token-level polarity score at time step t and $s^{\mathbf{\Phi}}_{i:t}$ is the sequence-level polarity score for the sequence between time step i and t.

5.1.2 Synthetic Dataset

We created a binary sentiment analysis dataset that consists of positive and negative instances (more details in Appendix A.2). The instances incorporate given polarity tokens, corresponding negation phrases as well as double negation phrases for both positive and negative tokens. Table 1 shows that the sequence-level representations could reflect polarity scores matching the roles of the negation and double negation phrases. 10 positive and 12 negative adjective words along with their negation and double negation were considered here. The negation token is "*not*".

We also observed that the double negation phenomenon may not be well captured if we remove the corresponding double negation phrase from the training set. As shown in Figure 1, in this case, the double negation phrases do not necessarily have polarity scores with the same signs as the token-level polarity scores. This indicates that in order for such models as GRU/LSTM to learn complex language structures, such as compositional semantic information, it may be essential for the model to have enough exposure to relevant structures during the training phase.

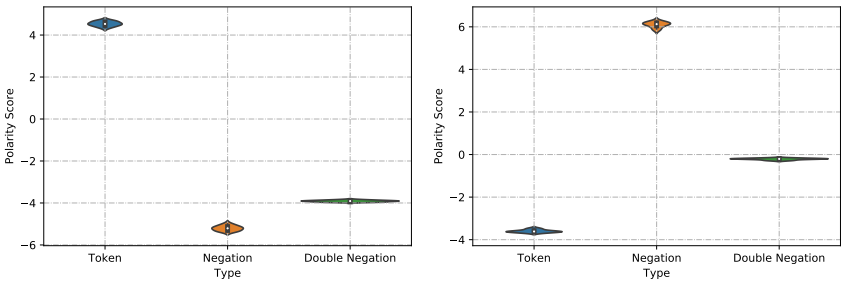

Figure 1: Left, positive tokens; right, negative tokens. Polarity score distribution for tokens, corresponding negation phrases and double negation phrases. Model with one GRU layer on the synthetic dataset without double negation phrases.

Figure 2 shows the polarity score for each sequence ending with the token "*charming*" in the sentence "*those filmmakings were not not charming*". The result is consistent with our analysis.

To understand the impact of sequences with different lengths, we selected positive and negative N-grams (N=1-3, with result for $N = 4$ in Appendix A.2) ending with the last token of the instances based on their association with positive and negative labels respectively. It appears that the token-level polarity scores for unigrams and sequence-level polarity scores for bigrams can reflect their association with labels better than that for trigrams (and four-grams, see Appendix) as shown in Figure 3. This demonstrates that although the sequence-level features can be well captured and are important for sentiment analysis tasks, the shorter sequences in general may be playing more crucial roles than the longer ones.

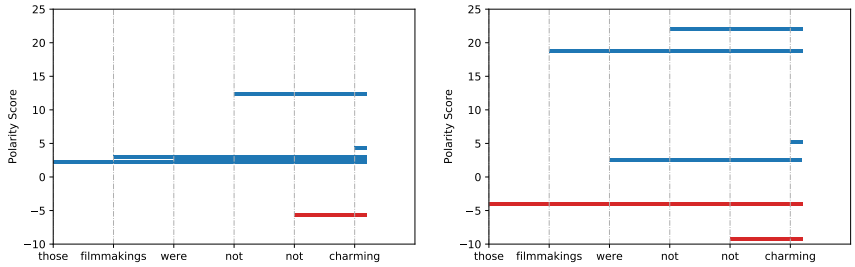

Figure 2: Left, result from a GRU cell. Right, result from a LSTM cell. Both the token "charming" and the double negation phrase "not not charming" have large positive polarity scores (in blue), whereas the negation phrase "not charming" has a large negative polarity score (in red). Each bar represents a sequence or a single token. Model trained on the synthetic dataset.

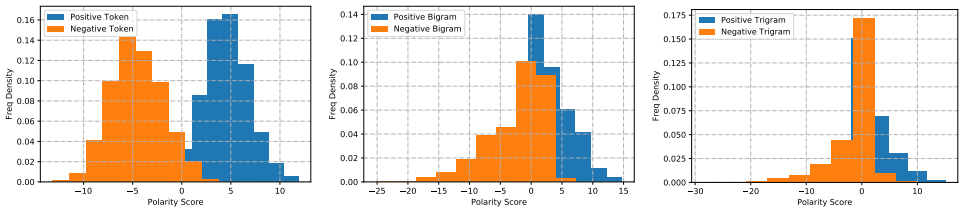

Figure 3: Polarity score distribution for the N-grams (N=1,2,3) that have strong association with a specific label. Result from a GRU cell on the SST dataset.

5.1.3 SST Dataset

We also trained a bidirectional GRU/LSTM model on the SST dataset with subphrase labels (Socher et al., 2013), and examined the two types of polarity scores. Labeled Subphrases were also viewed as instances during training. Figure 4 shows the token-level and sequence-level polarity scores from a bidirectional GRU/LSTM cell given an instance from the SST dataset, which indicates that the bidirectional cells could capture sequence-level features from the sequences in both directions. Glove (Pennington et al., 2014) was used as the pre-trained embeddings.

5.2 Training with Approximate Hidden State Representations

We trained models by replacing the GRU or LSTM cell with the corresponding approximate representations (e.g., $\hat{\boldsymbol{h}}_t$ in GRU). We examined their performances on both sentiment analysis and language modeling tasks and compared them with the standard models. Additionally, we created a baseline model named "Simplified" by removing all the terms involving $\boldsymbol{x}_t$ from $\boldsymbol{A}(\boldsymbol{x}_t)$ in Equation 12 and 17, then the representations will not capture sequence-level information.

For the sentiment analysis task evaluated on the SST dataset, we found both the standard models and our approaches with approximations behaved similarly, as shown in Table 2.

Table 2: Accuracy (%) on SST

Type		Valid	Test
GRU	Standard	85.89	88.13
	Approx	86.35	88.30
	Simplified	82.91	83.69
LSTM	Standard	85.78	87.86
	Approx	86.93	87.92
	Simplified	83.02	84.79

We ran language modeling tasks on the Penn Treebank (PTB) dataset (Marcus et al., 1993), Wikitext-2 dataset and Wikitext-103 dataset (Merity et al., 2016) respectively[2]. The embedding size and hidden size were both set as 128 for PTB and Wikitext-2, and 256 for Wikitext-103. *Adpative softmax* (Joulin et al., 2017) was used for Wikitext-130. We can see that using the approximate representations can yield comparable results as the standard GRU or LSTM cells, as shown in Table 3. We can see a sharp drop in the performances as shown in "simplified" row of Table 3. This demonstrated that the

[2]Our model is a word-level language model, we used the *torchtext* package to obtain and process the data.

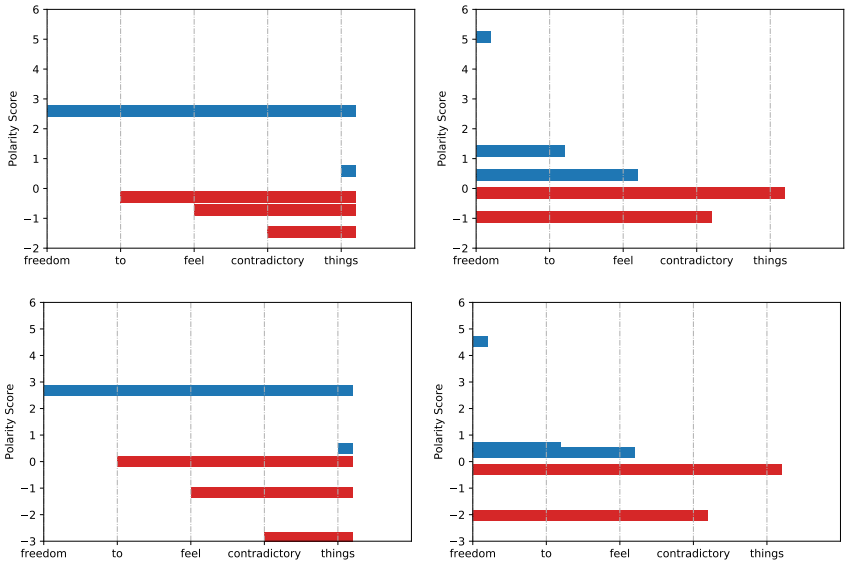

Figure 4: Left, forward direction; right, backward direction. Top, GRU; bottom, LSTM. In the forward direction, the sequence "*contradictory things*" has a significant negative polarity score whereas the entire sequence "*freedom to feel contradictory things*" has a significant positive polarity score. In the backward direction, the sequence "*to freedom*" and token "*freedom*" have significant positive polarity scores.

sequence-level features might be a major contributor to the performances of a GRU or LSTM cell apart from the token-level features.

Although using the approximate representations perform well on both tasks, we cannot rule out the contributions of other underlying complex features possibly captured by the standard cells. This can be observed from the text perplexities obtained from the approximate hidden states, which are generally slightly lower than those obtained from the standard GRU or LSTM models.

We also believe our analysis can provide some inspirations on designing alternative (and potentially more efficient) sequence encoders, for which we provide some further discussion in Appendix.

Table 3: Perplexities on the validation and test datasets.

Type	**Variants**	**PTB**		**Wikitext-2**		**Wikitext-103**	
		Valid	Test	Valid	Test	Valid	Test
GRU	Standard	89.08	77.19	91.79	80.68	121.77	116.70
	Approx	90.46	78.32	94.53	82.69	121.70	116.78
	Simplified	100.30	85.58	105.50	92.28	151.74	146.54
LSTM	Standard	89.22	77.11	92.23	80.74	105.25	101.08
	Approx	90.21	77.82	94.35	81.21	118.21	114.03
	Simplified	101.06	85.64	105.67	92.13	143.80	138.97

6 Conclusion

In this work, we explore the underlying features captured by GRU and LSTM cells. We unrolled GRU or LSTM and expanded their internal states, and obtained approximate state representations with closed-form expressions in terms of inputs.

Theoretically, we found that the approximate state representations were able to encode both token-level and sequence-level features, and found their close connection with the N-gram information captured by classical sequence models. Empirically, we examined the use of approximate representations based on our finding, and showed that they could behave similarly to the standard GRU or LSTM models on the sentiment analysis and language modeling tasks. Our results confirm the importance of sequence-level features as captured by GRU or LSTM, but at the same time, we cannot rule out the contributions of alternative, possibly richer and more complex features captured by the standard models.

REFERENCES

George Arfken and Albert A Mullin. *Mathematical Methods for Physicists, 3rd ed.*, chapter 5, pp. 303–313. Orlando, FL: Academic Press, 3 edition, 1985.

Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. Explaining recurrent neural network predictions in sentiment analysis. In *Proceedings of WASSA@EMNLP*, 2017.

Sarath Chandar, Chinnadhurai Sankar, Eugene Vorontsov, Samira Ebrahimi Kahou, and Yoshua Bengio. Towards non-saturating recurrent units for modelling long-term dependencies. In *Proceedings of AAAI*, volume 33, pp. 3280–3287, 2019. URL https://www.aaai.org/ojs/index.php/AAAI/article/view/4200.

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. URL http://arxiv.org/abs/1412.3555.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011. URL http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10): 2222–2232, 2016. URL http://arxiv.org/abs/1503.04069.

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

Armand Joulin, Moustapha Cissé, David Grangier, Hervé Jégou, et al. Efficient softmax approximation for gpus. In *International Conference on Machine Learning*, pp. 1302–1310. PMLR, 2017. URL https://arxiv.org/abs/1609.04309.

Sekitoshi Kanai, Yasuhiro Fujiwara, and Sotetsu Iwamura. Preventing gradient explosions in gated recurrent units. In *Proceedings of NeurIPS*, 2017. URL http://papers.nips.cc/paper/6647-preventing-gradient-explosions-in-gated-recurrent-units.pdf.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015. URL http://arxiv.org/abs/1506.02078.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of ICLR*, 2014. URL http://arxiv.org/abs/1412.6980.

Ben Krause, Liang Lu, Iain Murray, and S. Renals. Multiplicative lstm for sequence modelling. *ArXiv*, abs/1609.07959, 2017. URL http://arxiv.org/abs/1609.07959.

Jiwei Li, Xinlei Chen, Eduard H. Hovy, and Dan Jurafsky. Visualizing and understanding neural models in NLP. *CoRR*, 2015. URL http://arxiv.org/abs/1506.01066.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL https://www.aclweb.org/anthology/J93-2004.

Gábor Melis, Tomáš Kočiský, and Phil Blunsom. Mogrifier lstm. In *Proceedings of ICLR*, 2020. URL https://openreview.net/forum?id=SJe5P6EYvS.

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

William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, and Eran Yahav. A formal hierarchy of RNN architectures. In *Proceedings of ACL*, pp. 443–459, July 2020. URL https://www.aclweb.org/anthology/2020.acl-main.43.

W. James Murdoch, Peter J. Liu, and Bin Yu. Beyond word importance: Contextual decomposition to extract interactions from LSTMs. In *Proceedings of ICLR*, 2018. URL https://openreview.net/forum?id=rkRwGg-0Z.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of EMNLP*, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Alex Sherstinsky. Deriving the recurrent neural network definition and rnn unrolling using signal processing. In *Proceedings of Critiquing and Correcting Trends in Machine Learning Workshop at NeurIPS*, volume 31, 2018. URL http://arxiv.org/abs/1808.03314.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, 2013. URL https://www.aclweb.org/anthology/D13-1170.

Xiaobing Sun and Wei Lu. Understanding attention for text classification. In *Proceedings of ACL*, pp. 3418–3428, July 2020. URL https://www.aclweb.org/anthology/2020.acl-main.312.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Proceedings of NeurIPS*, pp. 3104–3112, 2014.

A Appendix

A.1 Dataset Statistics

Table 4: Sentiment analysis dataset

Data	Train/Valid/Test	Train Pos/Neg	Remark
SST	98,794/872/1,821	56,187/42,607	with subphrases
Synthetic	4,120/200/200	2,060/2,060	

Table 5: Language modeling dataset, statistics quoted from Einstein.ai

	PTB			Wikitext-2			Wikitext-103		
	Train	Valid	Test	Train	Valid	Test	Train	Valid	Test
Article Num	-	-	-	600	60	60	28,475	60	60
Token Num	887,521	70,390	78,669	2,088,628	217,646	245,569	103,227,021	217,646	245,569
Vocab Size		10,000			33,278			267,735	

A.2 Synthetic Dataset

We created a vocabulary consisting of positive tokens, negative tokens and neutral tokens. Then we made labeled instances with those tokens. For example, we let the positive token "nice" appear mostly in positive instances, negative token "poor" appear mostly in negative instances. And we let the phrase "not nice" appear in instances with negative labels, the phrase "not poor" appear in instances with positive labels. For the double negation phrases such as "not not nice", "not not poor", we made them appear in positive and negative instances respectively. The tokens, negation and double negation phrases are shown in Table 6.

Embeddings were randomly initialized. The final hidden state of a GRU or LSTM cell was fed into a fully-connected linear layer for binary classification.

Adagrad optimizers (Duchi et al., 2011) were used.

Table 6: Tokens, negation and double negation phrases for the synthetic dataset

Polarity	Token	Negation	Double Negation
Positive	good, nice, charming, awesome, fascinating, attractive, interesting, sweet, stunning, amazing	not mediocre, not pointless, not gross, not bad, not awful, not unfunny, not shallow, not tedious, not poor	not not stunning, not not nice
Negative	awful, bad, uninspiring, dull, boring, tedious, mediocre, shallow, pointless, unfunny, gross, poor	not interesting, not sweet, not attractive, not awesome, not fascinating, not nice, not amazing, not charming, not good	not not mediocre, not not unfunny, not not pointless, not not poor

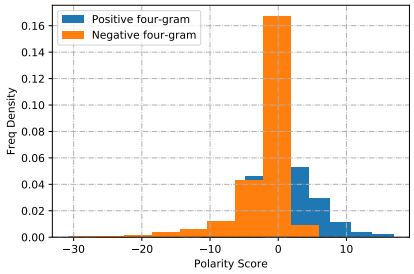

Figure 5: Polarity score distribution for selected Four-grams. Synthetic dataset, GRU model.

A.3 Performance During Training

We make comparisons between the performances of the standard GRU/LSTM cell and the approximate hidden state representations during training. It can be seen from Figure 6 and 7 that the approximate hidden state representations perform similarly to the standard GRU/LSTM cell on the PTB, Wikitext-2 and Wikitext-103 datasets. But there's an apparent difference on the Wikitext-103 dataset. There can be other underlying features captured by LSTM cells. Adam optimizers (Kingma & Ba, 2014) were used.

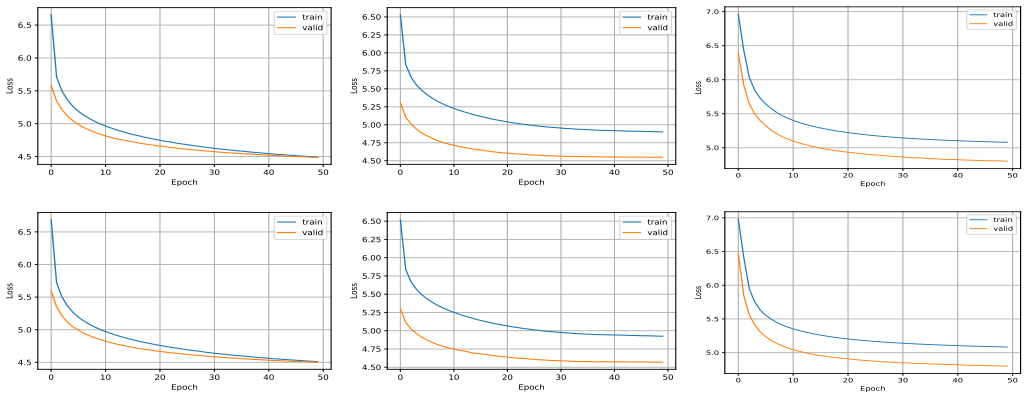

Figure 6: Top, standard GRU Cell; bottom, approximate hidden state representation. From left to right: PTB, Wikitext-2, Wikitext-103. Training and validation losses for language modeling tasks.

A.4 Proposed Architectures

Based on the aforementioned analysis, it is possible that we bypass the gates of GRUs or LSTMs, and create contextualized encoders by re-defining $\boldsymbol{g}(\boldsymbol{x}_i)$ and $\boldsymbol{A}(\boldsymbol{x}_k)$ in Equation 12, $\boldsymbol{g}_c(\boldsymbol{x}_i)$, $\boldsymbol{g}_h(\boldsymbol{x}_i)$ and $\boldsymbol{A}(\boldsymbol{x}_k)$ in Equation 17. We do not change the parameter size compared to the standard cell. We clamp the elements of the approximate hidden states and make them fall in the range (-1.5, 1.5) for Wikitext-103.

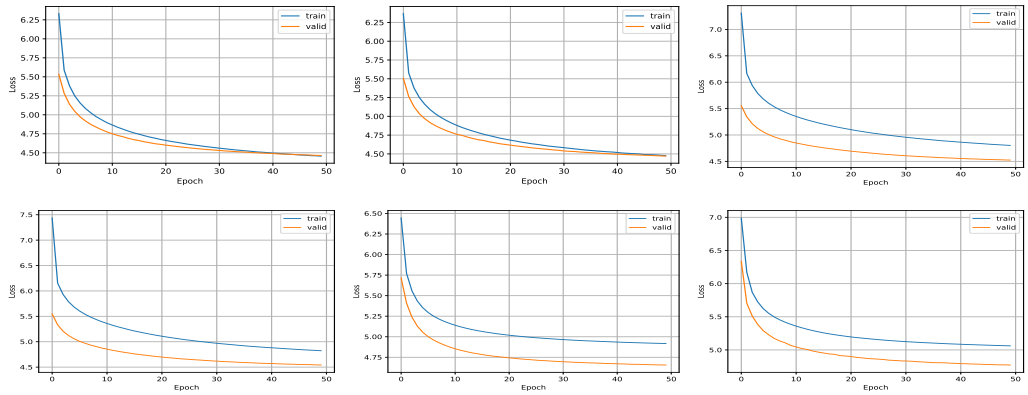

Figure 7: Top, standard LSTM Cell; bottom, approximate hidden state representation. From left to right: PTB, Wikitext-2, Wikitext-103. Training and validation losses for language modeling tasks.

GRU Variant (test perplexity): 76.90 for PTB, 81.42 for Wikitext-2

$$
\begin{aligned}
\boldsymbol{g}(\boldsymbol{x}_t) &= \frac{1}{2}\boldsymbol{x}_t^n - \frac{1}{4}\boldsymbol{x}_t^n \odot \boldsymbol{x}_t^z, \\
\boldsymbol{A}(\boldsymbol{x}_t) &= \frac{1}{4}\boldsymbol{W}_{hn} + \frac{7}{10}\boldsymbol{I} - \frac{1}{4}\text{diag}(\boldsymbol{x}_t^n)\boldsymbol{W}_{hz} + \frac{1}{4}\text{diag}(\boldsymbol{x}_t^z)\boldsymbol{W}_{hr} + \frac{1}{8}\text{diag}(\boldsymbol{x}_t^r - \boldsymbol{x}_t^z - \boldsymbol{x}_t^r * \boldsymbol{x}_t^z)\boldsymbol{W}_{hn}.
\end{aligned} \tag{21}
$$

LSTM Variant (test perplexity): 76.63 for PTB, 80.95 for Wikitext-2

$$
\begin{aligned}
\boldsymbol{g}_c(\boldsymbol{x}_t) &= \frac{1}{4}\boldsymbol{x}_t^i \odot \boldsymbol{x}_t^c + \frac{1}{2}\boldsymbol{x}_t^c, \\
\boldsymbol{B}(\boldsymbol{x}_t) &= \frac{1}{4}\text{diag}(\boldsymbol{x}_t^f + \mathbf{2.4} + \frac{12}{5}\boldsymbol{x}_t^i + 0.4\boldsymbol{x}_t^g), \\
\boldsymbol{D}(\boldsymbol{x}_t) &= \frac{1}{4}\text{diag}(\boldsymbol{x}_t^i + \mathbf{2})\boldsymbol{W}_{hc} + \frac{1}{4}\text{diag}(\boldsymbol{x}_t^c)\boldsymbol{W}_{hi} + \frac{1}{10}\text{diag}(\boldsymbol{x}_t^f)\boldsymbol{W}_{hf}, \\
\boldsymbol{g}_h(\boldsymbol{x}_t) &= \frac{1}{4}\boldsymbol{x}_t^o \odot \boldsymbol{g}_c(\boldsymbol{x}_t) + \frac{1}{2}\boldsymbol{g}_c(\boldsymbol{x}_t), \\
\boldsymbol{E}(\boldsymbol{x}_t) &= \frac{1}{4}\text{diag}(\boldsymbol{x}_t^o)\boldsymbol{B}(\boldsymbol{x}_t) + \frac{1}{2}\boldsymbol{B}(\boldsymbol{x}_t), \\
\boldsymbol{F}(\boldsymbol{x}_t) &= \frac{1}{4}\text{diag}(\boldsymbol{x}_t^o + \mathbf{2})\boldsymbol{D}(\boldsymbol{x}_t) + \frac{1}{4}\text{diag}(\boldsymbol{g}_c(\boldsymbol{x}_t) + \frac{1}{10}\boldsymbol{x}_t^f + \frac{1}{10}\boldsymbol{x}_t^o)\boldsymbol{W}_{ho},
\end{aligned} \tag{22}
$$