

Learning longer-term dependencies via grouped distributor unit

Wei Luo, Feng Yu*

College of Biomedical Engineering and Instrument Science, Yuquan Campus, Zhejiang University, 38 Zheda Road, Hangzhou 310027, China

ARTICLE INFO

Article history:

Received 29 April 2019

Revised 14 March 2020

Accepted 24 June 2020

Available online 6 July 2020

Communicated by Long Cheng

Keywords:

Recurrent neural network

Sequence learning

Long-term memory

ABSTRACT

Learning long-term dependencies remains difficult for recurrent neural networks (RNNs) despite their success in sequence modeling recently. In this paper, we propose a novel gated RNN structure, which contains only one gate. Hidden states in the proposed grouped distributor unit (GDU) are partitioned into groups. For each group, the proportion of memory to be overwritten in each state transition is limited to a constant and is adaptively distributed to each group member. In other words, every separate group has a fixed overall update rate, yet all units are allowed to have different paces. Information is therefore forced to be latched in a flexible way, which helps the model to capture long-term dependencies in data. Besides having a simpler structure, GDU is demonstrated experimentally to outperform other models such as LSTM and GRU on both pathological problems and tasks on natural datasets.

© 2020 Elsevier B.V. All rights reserved.

1. Introduction

Recurrent Neural Networks (RNNs, [1,2]) are powerful dynamic systems for tasks that involve sequential inputs, such as audio classification, machine translation and speech generation. As they process a sequence one element at a time, internal states are maintained to store information computed from the past inputs, which makes RNNs capable of modeling temporal correlations between elements from any distance in theory.

In practice, however, it is difficult for RNNs to learn long-term dependencies in data by using back-propagation through time (BPTT, [1]) due to the well-known *vanishing* and *exploding gradient* problem [3]. Besides, training RNNs suffers from gradient conflicts (e.g. *input conflict* and *output conflict* [4]) which make it challenging to latch long-term information while keeping mid- and short-term memory simultaneously. Various attempts have been made to increase the temporal range in which credit assignment takes effect for recurrent models during training, including adopting a much more sophisticated Hessian-Free optimization method instead of stochastic gradient descent [5,6], using *orthogonal* weight matrices to assist optimization [7,8] and allowing direct connections to model inputs or states from the distant past [9–11]. Long short-term memory (LSTM, [4]) and its variant, known as gated recurrent units (GRU, [12]) mitigate gradient conflicts by using *multiplicative gate units*. Moreover, the vanishing gradient problem is alleviated by the additivity in their state transition

operator. Simplified gated units have been proposed [13,14] yet the ability to capture long-term dependencies has not been improved. Recent work also supports the idea of partitioning the hidden units in an RNN into separate modules with different processing periods [15].

In this paper, we introduce the Grouped Distributor Unit (GDU), a new gated recurrent architecture with additive state transition and only one gate unit. Hidden states inside a GDU are partitioned into groups, each of which keeps a constant proportion of previous memory at each time step, forcing information to be latched. The vanishing gradient problem, together with the issue of gradient conflict, which impedes the extraction of long-term dependencies are thus alleviated.

We empirically evaluated the proposed model against other models such as LSTM and GRU on both synthetic problems which are designed to be pathologically difficult and natural datasets containing long-term components. On synthetic problems, our proposed model outperforms LSTM and GRU with a simpler structure and fewer parameters. In tasks on natural datasets, GDU outperforms other related models that were recently proposed.

2. Background and related work

An RNN is able to encode sequences of arbitrary length into a fixed-length representation by folding a new observation \mathbf{x}_t into its hidden state \mathbf{s}_t using a transition operator T at each time step t :

$$\mathbf{s}_t = T(\mathbf{x}_t, \mathbf{s}_{t-1}). \quad (1)$$

* Corresponding author.

E-mail addresses: willi4m@zju.edu.cn (W. Luo), osfengyu@zju.edu.cn (F. Yu).

Simple recurrent networks (SRN, [16]), known as one of the earliest variants, make T as the composition of an element-wise non-linearity with an affine transformation of both \mathbf{x}_t and \mathbf{s}_{t-1} ¹:

$$\mathbf{s}_t = \phi_s(\mathbf{W}_s \mathbf{x}_t + \mathbf{U}_s \mathbf{s}_{t-1} + \mathbf{b}_s), \quad (2)$$

where \mathbf{W}_s is the input-to-state weight matrix, \mathbf{U}_s is the state-to-state recurrent weight matrix, \mathbf{b}_s is the bias and ϕ_s is the nonlinear activation function. For the convenience of the following descriptions, we denote this kind of operator as $\eta(\cdot, \cdot, \phi)$, and a subscript can be added to distinguish different network components. Thus in SRN, $\mathbf{s}_t = \eta_s(\mathbf{x}_t, \mathbf{s}_{t-1}, \phi_s)$.

During training via BPTT, the error obtained from the output of an RNN at time step t (denoted as \mathcal{L}_t) travels backward through each state unit. The corresponding error signal propagated back to time step τ (denoted as $\epsilon_{\tau \leftarrow t} = \frac{\partial \mathcal{L}_t}{\partial \mathbf{s}_\tau}$, $\tau < t$) contains a product of $t - \tau$ Jacobian matrices:

$$\epsilon_{\tau \leftarrow t} = \epsilon_{t \leftarrow t} \prod_{t \geq i > \tau} \frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}. \quad (3)$$

From Eq. (3) we can easily find a *sufficient condition* for the *vanishing gradient* problem to occur, i.e., $\forall \tau < i \leq t, \|\frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}\| < 1$. Under this condition, a bound $\xi \in \mathcal{R}$ can be found such that $\forall i, \|\frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}\| \leq \xi < 1$, and

$$\|\epsilon_{\tau \leftarrow t}\| = \|\epsilon_{t \leftarrow t} \prod_{t \geq i > \tau} \frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}\| \leq \xi^{t-\tau} \|\epsilon_{t \leftarrow t}\|. \quad (4)$$

As $\xi < 1$, long term contributions (for which $t - \tau$ is large) go to 0 exponentially fast with $t - \tau$.

In SRN, $\frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}$ is given by $\mathbf{U}_s^T \text{diag}(\phi'_s(\mathbf{W}_s \mathbf{x}_i + \mathbf{U}_s \mathbf{s}_{i-1} + \mathbf{b}_s))$. As a result, if the derivative of the nonlinear function is bounded in SRN, namely, $\exists \kappa \in \mathcal{R}$, s.t. $|\phi'_s(x)| \leq \kappa$, it will be *sufficient* for $\lambda_1 < \frac{1}{\kappa}$, where λ_1 is the largest singular value of the recurrent weight matrix \mathbf{U}_s , for $\epsilon_{\tau \leftarrow t}$ to vanish (as $t \rightarrow \infty$) [17].

Any RNN architecture with a long-term memory ability should at least be designed to make sure the norm of its transition Jacobian will not easily be bounded by 1 for a long period as it goes through a sequence.

2.1. Gated additive state transition (GAST)

Long short-term memory (LSTM, [4]) introduced a memory unit with a self-connected structure, which can maintain its state over time, and non-linear gating units (originally input and output gates), which control the information flow into and out of it. Since the initial proposal in 1997, many improvements have been made to the LSTM architecture [18,19]. In this paper, we refer to the variant with forget gate and without peephole connections, which has a comparable performance with more complex variants [20]:

$$\mathbf{f}_t = \eta_f(\mathbf{x}_t, \mathbf{h}_{t-1}, \sigma), \quad (5a)$$

$$\mathbf{i}_t = \eta_i(\mathbf{x}_t, \mathbf{h}_{t-1}, \sigma), \quad (5b)$$

$$\mathbf{o}_t = \eta_o(\mathbf{x}_t, \mathbf{h}_{t-1}, \sigma), \quad (5c)$$

$$\bar{\mathbf{s}}_t = \eta_s(\mathbf{x}_t, \mathbf{h}_{t-1}, \tanh), \quad (5d)$$

$$\mathbf{s}_t = \mathbf{f}_t \odot \mathbf{s}_{t-1} + \mathbf{i}_t \odot \bar{\mathbf{s}}_t, \quad (5e)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{s}_t). \quad (5f)$$

Here σ denotes the sigmoid activation and \odot denotes element-wise multiplication. Note that \mathbf{h}_t should also be considered as a hidden state vector besides \mathbf{s}_t .

Cho et al. [12] proposed a similar architecture with gating units called gated recurrent unit (GRU). Different from LSTM, GRU exposes all its states to the output and use a linear interpolation between the previous state \mathbf{s}_{t-1} and the candidate state $\bar{\mathbf{s}}_t$:

$$\mathbf{r}_t = \eta_r(\mathbf{x}_t, \mathbf{s}_{t-1}, \sigma), \quad (6a)$$

$$\mathbf{z}_t = \eta_z(\mathbf{x}_t, \mathbf{s}_{t-1}, \sigma), \quad (6b)$$

$$\bar{\mathbf{s}}_t = \eta_s(\mathbf{x}_t, \mathbf{r}_t \odot \mathbf{s}_{t-1}, \tanh), \quad (6c)$$

$$\mathbf{s}_t = \mathbf{z}_t \odot \mathbf{s}_{t-1} + (1 - \mathbf{z}_t) \odot \bar{\mathbf{s}}_t. \quad (6d)$$

Previous work has indicated the advantages of the gating units over the more traditional recurrent units [21]. Both LSTM and GRU perform well in tasks that require capturing long-term dependencies. However, the choice of these two structures may depend heavily on the dataset and corresponding task.

It is easy to notice that the most prominent feature shared between these units is the additivity in their state transition operators. In another word, both LSTM and GRU keep the existing states and add the new states on top of it instead of replacing previous states directly, as it did in traditional recurrent units like SRN. Another important ingredient in their transition operator is the gating mechanism, which regulates the information flow and enables the network to form skip connections adaptively. In this paper we refer to this kind of transition operators as the *Gated Additive State Transition* (GAST) with a general formula:

$$\bar{\mathbf{s}}_t = \eta_s(\mathbf{x}_t, \gamma_t(\mathbf{s}_{t-1}), \phi), \quad (7a)$$

$$\mathbf{s}_t = \beta_t(\mathbf{s}_{t-1}) + \alpha_t(\bar{\mathbf{s}}_t), \quad (7b)$$

where α_t, β_t , and γ_t are called *gate operators* with subscript t indicating that values of the corresponding gating units change over time (see Fig. 1 (left)). In LSTM:

$$\gamma_t(\mathbf{s}_{t-1}) = \mathbf{o}_{t-1} \odot \tanh(\mathbf{s}_{t-1}), \quad (8a)$$

$$\beta_t(\mathbf{s}_{t-1}) = \mathbf{f}_t \odot \mathbf{s}_{t-1}, \quad (8b)$$

$$\alpha_t(\bar{\mathbf{s}}_t) = \mathbf{i}_t \odot \bar{\mathbf{s}}_t, \quad (8c)$$

whilst in GRU:

$$\gamma_t(\mathbf{s}_{t-1}) = \mathbf{r}_t \odot \mathbf{s}_{t-1}, \quad (9a)$$

$$\beta_t(\mathbf{s}_{t-1}) = \mathbf{z}_t \odot \mathbf{s}_{t-1}, \quad (9b)$$

$$\alpha_t(\bar{\mathbf{s}}_t) = (1 - \mathbf{z}_t) \odot \bar{\mathbf{s}}_t. \quad (9c)$$

We denote the gate vector used in a gate operator T at time step t as \mathcal{G}_{T_t} . Note that except Eq. (8a), gate operators T_t have a common form³:

$$T_t(\mathbf{s}) = \mathcal{G}_{T_t} \odot \mathbf{s}, \quad (10)$$

where \mathbf{s} is a state vector to be gated. We use $\beta_t = 1 - \alpha_t$ to indicate $\mathcal{G}_{\beta_t} = 1 - \mathcal{G}_{\alpha_t}$ as in the case of GRU. According to Eq. (7b), the transition Jacobian of a GAST can be resolved into 4 parts:

$$\frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_{t-1}} = J_{\mathbf{s}_{t-1}} + J_{\bar{\mathbf{s}}_t} + J_{\mathcal{G}_{\alpha_t}} + J_{\mathcal{G}_{\beta_t}}, \quad (11)$$

in which

$$J_{\mathbf{s}_{t-1}} = \text{diag}(\mathcal{G}_{\beta_t}), \quad (12a)$$

$$J_{\bar{\mathbf{s}}_t} = \text{diag}(\mathcal{G}_{\alpha_t}) \frac{\partial \bar{\mathbf{s}}_t}{\partial \mathbf{s}_{t-1}}, \quad (12b)$$

$$J_{\mathcal{G}_{\beta_t}} = \text{diag}(\mathbf{s}_{t-1}) \frac{\partial \mathcal{G}_{\beta_t}}{\partial \mathbf{s}_{t-1}}, \quad (12c)$$

$$J_{\mathcal{G}_{\alpha_t}} = \text{diag}(\bar{\mathbf{s}}_t) \frac{\partial \mathcal{G}_{\alpha_t}}{\partial \mathbf{s}_{t-1}}. \quad (12d)$$

¹ We do not consider RNNs with connections from the past such as NARX RNN [9].
² $\frac{\partial \mathcal{L}_t}{\partial \mathbf{s}_\tau} = (\frac{\partial \mathcal{L}_t}{\partial \mathbf{s}_1^1}, \frac{\partial \mathcal{L}_t}{\partial \mathbf{s}_1^2}, \dots, \frac{\partial \mathcal{L}_t}{\partial \mathbf{s}_1^M})^T$, in which M is the state size, and the k -th component represents the sensitivity of \mathcal{L}_t to small perturbations in the k -th state unit at time step τ .

³ In the following part of this paper, *gate operators* are referred to as being in this form.

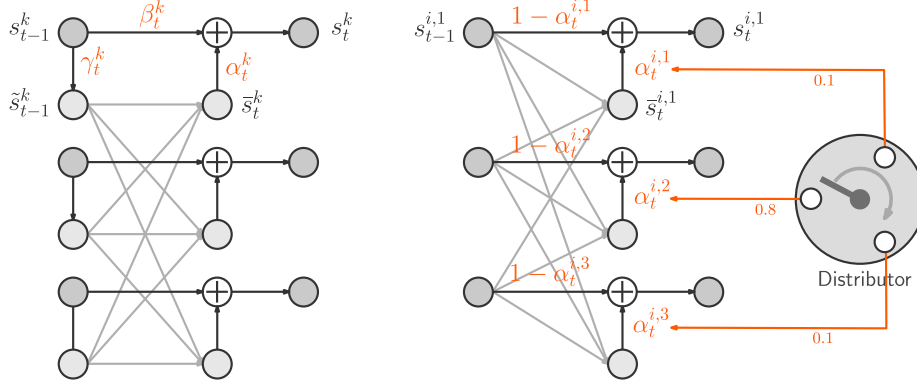


Fig. 1. Left: The gated additive state transition (GAST). Inputs and outputs are not shown. Superscript k denotes the ordinal number of a state unit. In LSTM, \tilde{s}_{t-1}^k corresponds to h_{t-1}^k . In GRU, $\beta_t^k = 1 - \alpha_t^k$. Right: The GAST in a GDU group with size 3 and $\delta_i = 1.0$. Compared to LSTM and GRU, gate operator γ_t is removed and gate operators $\{\alpha_t^{i,k}\}_k$ inside the group are correlated, i.e., $\sum_k \alpha_t^{i,k} = \delta_i = 1$. Any unit assigned with a high $\alpha_t^{i,k}$ will force other group members to latch information.

The gradient property of GAST is much better than that of SRN since it can easily prevent its transition Jacobian norm to be bounded within 1 by saturating part of units in \mathcal{G}_{β_t} nearly at 1. Intuitively, when this happens, the corresponding components of the error signal are allowed to be back-propagated easily through the shortcut created by the additive character of GAST without vanishing too quickly.

The original LSTM [4] uses *full gate recurrence* [20], which means that all neurons receive recurrent inputs from all gate activations at the previous time step besides the block outputs. Nevertheless, it still follows Eqs. (7). Another difference is that the original LSTM does not use forget gate, i.e., $\beta_t(s_{t-1}) = s_{t-1}$. Thus in Eq. (12a), $J_{s_{t-1}}$ is a unit diagonal matrix \mathcal{I} . In addition, gradients are truncated by replacing the other components in its transition Jacobian (i.e., Eqs. (12b), (12c), and (12d)) by zero, forming a *constant error carousel* (CEC), where $\frac{\partial s_t}{\partial s_{t-1}} = \mathcal{I}$. It is noticeable, however, that if the gradients are not truncated, Eq. (3) does not hold for LSTMs since the gate vector α_{t-1} used in γ_t is calculated at the previous time step, see Eq. (8a). In this condition, a concatenation of s_t and $h_t = \gamma_t(s_t)$ should be used in the analysis of its transition Jacobian, as in Fig. 7.

Simplifying GAST has drawn researchers' interest recently. GRU itself reduces the number of gate units to 2 compared to LSTM, which has 3 gate units, by coupling forget gate and input gate into one update gate, namely making the gate operator β_t equals to $1 - \alpha_t$. In this paper, we denote this kind of GAST as cGAST, with the prefix c short for *coupled*. Based on GRU, the Minimal Gated Unit (MGU, [14]) reduced the gate number further to only 1 by letting $\gamma_t = \beta_t = 1 - \alpha_t$ without losing GRU's accuracy benefits. The Update Gate RNN (UGRNN, [13]) entirely removed γ_t operator. However, none of these models has shown superiority over LSTM and GRU on long-term tasks with single-layer hidden states.

GAST can also be designed to be sophisticated. The ordered neurons LSTM (ON-LSTM, [22]) explicitly integrates latent tree structures into its GAST by ordering the state neurons. Low-ranking neurons will store short-term information, which can be rapidly forgotten, while high-ranking neurons will store long-term information, which is kept for a large number of steps, providing shortcuts for gradient backpropagation which helps with the long-term dependency problem.

2.2. Units partitioning

Although the capacity of capturing long-term dependencies in sequences is of the crucial importance of RNNs, it is worthwhile to notice that the flowing data is usually embedded with both

slow-moving and fast-moving information, of which the former corresponds to long-term dependencies. Along with the existence of both long- and short-term information in sequences, the training process always has *gradient conflict* existing. Here *gradient conflict* mainly refers to the contradiction between error signals back-propagated to the same time step but injected at different time steps during training via BPTT. This issue may hinder the establishment of long-term memory even without the gradient vanishing problem.

Consider a task in which a GRU is given one data point at a time and assigned to predict the next, e.g., *mERG* (see Section 4.3). If the correct prediction at time step t_1 is heavily depending on the data point appeared at time step t_0 , namely x_{t_0} , where $t_0 \ll t_1$, we can say a long-term dependency exists between x_{t_0} and x_{t_1+1} . GRU can capture this kind of dependency by learning to encode x_{t_0} into some state units and latch it until t_1 . For simplicity, let us focus on a single state unit s^k and assume that the information of x_{t_0} has been stored in $s_{t_0}^k$. At time step t ($t_0 < t < t_1$), state unit s_t^k will often receive conflicting error signals. The error signal $e_{t-t_1}^k$ injected at time step t_1 may attempt to make s_t^k keep its value until t_1 . While other error signals injected before t_1 , say, t_2 , may hope that s_t^k helps to do the prediction at time step t_2 , thus it may attempt to make s_t^k to be overwritten by a new value. This conflict makes the GRU model hesitate to shut the update gate for s^k by setting α_{t_2} to 0. In GRU, we also observed that state units latching long-term memories (with corresponding neurons in \mathcal{G}_{β_t} staying active for a long time) are usually sparse (see Fig. 6 (left)), which impedes the back-propagation of effective long-term error signals, since short-term error signals dominate. As a result, learning can be slow.

El Hichi and Bengio first showed that RNNs can learn both long- and short-term dependencies more easily and efficiently if state units are partitioned into groups with different timescales [23]. The clockwork RNN (CW-RNN) [15] implemented this by assigning each state unit a fixed temporal granularity, making state transition happens only at its prescribed clock rate. It can also be seen as a member of cGAST family. More specifically, a UGRNN with a special gate operator β_t in which each gate vector value $\alpha_{\beta_t}^k$ is explicitly scheduled to saturate at either 0 or 1. CW-RNN does not suffer from gradient conflict for it inherently has the ability to latch information. However, the clock rate schedule should be tuned for each task.

The Auxiliary Memory Unit (AMU, [24]) also organize state neurons into groups, in each of which only one neuron serves as the output. In addition, AMU provides an auxiliary memory neuron to maintain memory in particular for each group. By separating

the memory and output in a group, gradient conflicts are alleviated and long-term dependencies can be preserved due to the linear additive update of the memory.

3. Grouped distributor unit

As introduced in Section 2, a network combining the advantages of GAST and the idea to partition state units into groups seems promising. Further, we argue that a dynamic system with memory does not need to overwrite the vast majority of its memory based on relatively little input data. For cGAST models in which $\beta_t = 1 - \alpha_t$, we define the proportion of states to be overwritten at time step t as:

$$\mathcal{P}_{\alpha_t} = \frac{1}{K} \sum_{k=1}^K \mathcal{G}_{\alpha_t}^k, \quad (13)$$

where K is the state size. On the other hand, the proportion of previous states to be kept is:

$$\mathcal{P}_{\beta_t} = \frac{1}{K} \sum_{k=1}^K \mathcal{G}_{\beta_t}^k = 1 - \mathcal{P}_{\alpha_t}. \quad (14)$$

Hence in our view, if a model input \mathbf{x}_t contains relatively a small amount of information compared to system memory \mathbf{s}_{t-1} , \mathcal{P}_{α_t} should be kept low to protect the previous states. For cGAST family members, a lower \mathcal{P}_{α_t} leads to more active units in \mathcal{G}_{β_t} (see Fig. 6 (right)) and thus less prone to be affected by gradient conflict.

To put a limit on \mathcal{P}_{α_t} , we start by a plain UGRNN and partition its state units into N groups:

$$\mathbf{s}_t = \left\{ \left\{ \mathbf{s}_t^{ij} \right\}_{j=1}^{M_i} \right\}_{i=1}^N, \quad (15)$$

where the i -th group contains M_i units. At each time step, for each i , we let a positive constant $\delta_i < M_i$ to be **distributed** to the corresponding components in \mathcal{G}_{α_t} , namely,

$$\sum_{j=1}^{M_i} \mathcal{G}_{\alpha_t}^{ij} = \delta_i, i = 1, 2, \dots, N. \quad (16)$$

Thus \mathcal{P}_{α_t} becomes a constant given by

$$\mathcal{P}_{\alpha_t} = \frac{\sum_{i=1}^N \delta_i}{\sum_{i=1}^N M_i} = \frac{1}{K} \sum_{i=1}^N \delta_i \in (0, 1). \quad (17)$$

See Fig. 1 (right), the distribution work in each group is done by a *distributor*, hence the proposed structure is called *Grouped Distributor Unit* (GDU). The distributor is implemented by utilizing the softmax activation over each group individually in calculating \mathcal{G}_{α_t} :

$$\vartheta_t = \mathbf{W}_x \mathbf{x}_t + \mathbf{U}_x \mathbf{s}_{t-1} + \mathbf{b}_x, \quad (18a)$$

$$d_t^{ij} = \frac{\exp(\vartheta_t^{ij})}{\sum_{j=1}^{M_i} \exp(\vartheta_t^{ij})}, \quad (18b)$$

$$\mathcal{G}_{\alpha_t}^{ij} = \begin{cases} \delta_i \cdot d_t^{ij} & \text{if } \delta_i \in (0, 1] \\ \frac{M_i - \delta_i}{M_i - 1} \cdot d_t^{ij} + \frac{\delta_i - 1}{M_i - 1} & \text{if } \delta_i \in (1, M_i) \end{cases}, \quad (18c)$$

here $1 \leq i \leq N, 1 \leq j \leq M_i$ and $\vartheta_t = (\vartheta_t^{1,1}, \dots, \vartheta_t^{1,M_1}, \dots, \vartheta_t^{N,1}, \dots, \vartheta_t^{N,M_N})^T$.⁴ Note that $\mathcal{G}_{\alpha_t}^{ij} \in [0, \delta_i]$ when $\delta_i \in (0, 1]$, and $\mathcal{G}_{\alpha_t}^{ij} \in (\frac{\delta_i - 1}{M_i - 1}, 1]$ when $\delta_i \in (1, M_i)$. The resulting GDU is given by

$$\mathbf{a}_t = \zeta(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{s}_{t-1} + \mathbf{b}_z; \{\delta_i, M_i\}_{i=1}^N), \quad (19a)$$

$$\mathbf{s}_t = (1 - \mathbf{a}_t) \odot \mathbf{s}_{t-1} + \mathbf{a}_t \odot \tanh(\mathbf{W}_s \mathbf{x}_t + \mathbf{U}_s \mathbf{s}_{t-1} + \mathbf{b}_s), \quad (19b)$$

where $\zeta(\cdot, \{\delta_i, M_i\}_{i=1}^N)$ denotes distributor operator with group configuration $\{\delta_i, M_i\}_{i=1}^N$ as is detailed in Eqs. (18). Apparently, the distributor operator is differentiable, thus a GDU model can be trained using standard back-propagation through time (BPTT).

From Eqs. (19) we can see the only difference between GDU and UGRNN is that GDU uses distributor operator as the activation function instead of sigmoid to calculate \mathcal{G}_{α_t} . Both activation functions need to calculate e^ϑ ($e^{-\vartheta}$ in the case of sigmoid) for each input ϑ . After that, the distributor operator needs to sum up and store the results for each group. Counting in the calculation in (18c), the additional operations introduced by distributor operator compared to sigmoid are at most $2K$ additions and K multiplications, namely $O(K)$ operations. Note that (18a) contains $O(K^2)$ operations, which dominates the total computation cost in calculating the gate units. Hence the difference in terms of computation cost between GDU and UGRNN can be neglected. As a matter of fact, the total computation cost of GDU is roughly $\frac{2}{3}$ and $\frac{1}{2}$ of that required by GRU and LSTM respectively.

GDU has an inherent strength to keep a long-term memory since any saturated state unit s^{ij} will force all other group members to latch information. As a result, “bandwidth” is wider for long-term information to travel forward and error signals to back-propagate (see Fig. 6 (right)). Like CW-RNN, we set an explicit rate δ_i for each group. However, instead of making all group members act in the same way, we allow each unit to find its own rate by learning.

In this paper, we let $\delta_i = 1, i = 1, 2, \dots, N$. As a consequence,

$$\mathcal{P}_{\alpha_t} = \frac{N}{\sum_{i=1}^N M_i} = \frac{N}{K}. \quad (20)$$

If the size of each state group is set to a constant M , \mathcal{P}_{α_t} will be further reduced to $\frac{1}{M}$.

4. Experiments

We evaluated the proposed GDU on both pathological synthetic tasks and natural datasets in comparison with other models. The pathological tasks comprise the adding problem (4.1), the 3-bit temporal order problem (4.2), and the multi-embedded Reber grammar (4.3), on which GDU was compared with LSTM and GRU. It is important to point out that although LSTM and GRU have similar performance in nature datasets [21], one model may outperform another by a huge gap in different pathological tasks like the adding problem at which GRU is good and the temporal order problem in which LSTM performs better. As for tasks on natural datasets including sequential image classification (4.4) and spoken word recognition (4.5), GDUs were compared with other various models.

If not otherwise specified, all networks have one hidden layer. Weight variables were initialized via Xavier uniform initializer [25], and the initial values of all internal state variables were set to 0. All networks were trained using Adam optimization method [26] via BPTT, and they shared the same learning steps in all experiments:

- Step 1: Randomly initialize model parameters.
- Step 2: Generate one or a batch of sequences.
- Step 3: Reset the network states and update the model using standard BPTT algorithm on data from Step 2.
- Step 4: Stop training if some certain conditions are met. Otherwise go to Step 2.

⁴ The permutation of $\left\{ \left\{ \vartheta_t^{ij} \right\}_{j=1}^{M_i} \right\}_{i=1}^N$ can be arbitrary.

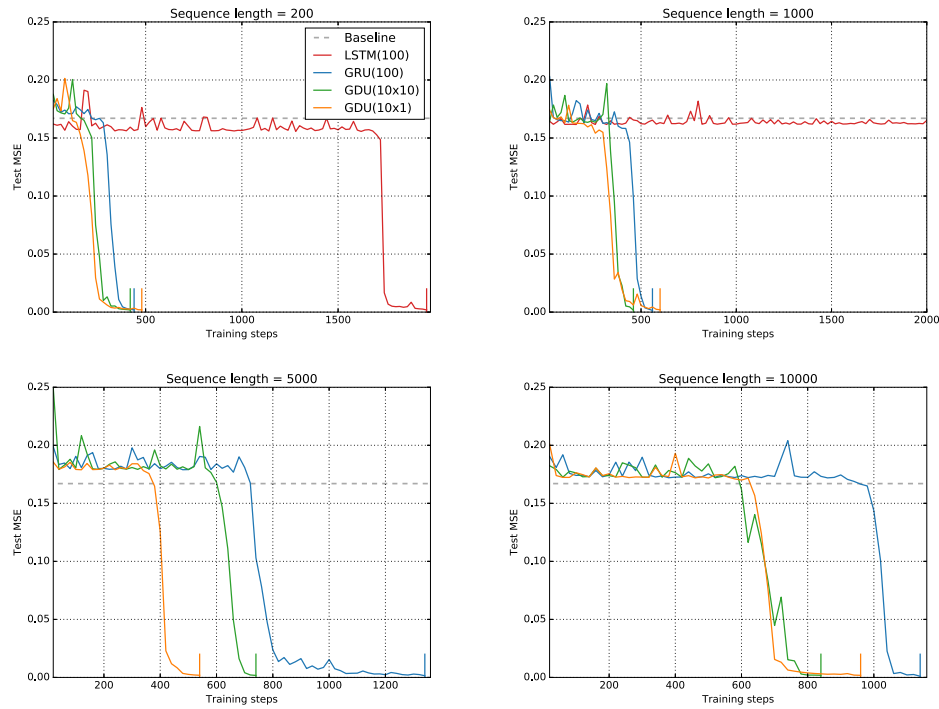


Fig. 2. The results of the adding problem on different sequence lengths. The legends for all sub-figures are the same thus are only shown in the first sub-figure, in which state sizes are specified following model names. For a GDU model, $(M \times N)$ means it has N groups of size M . Each training trial was stopped when the test MSE reached below 0.002, as indicated by a short vertical bar. When training with sequences of length 1000, LSTM(100) failed to converge within 10000 steps and only the curve of the first 2000 steps is shown.

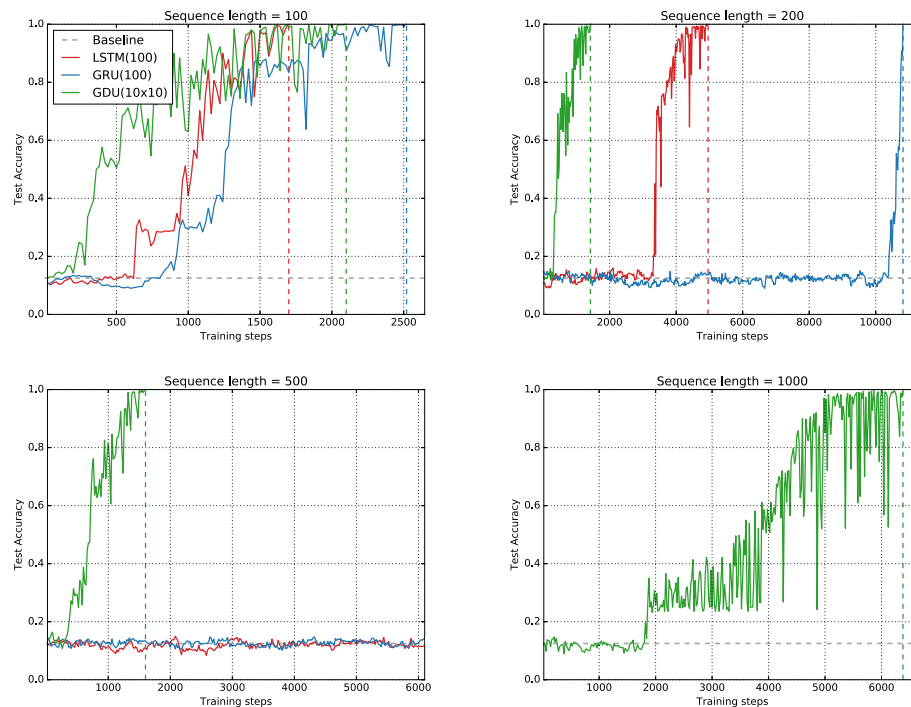


Fig. 3. The results of the 3-bits temporal order problem on different sequence lengths. The legends containing the information of model size are only shown in the first sub-figure. Each trial was stopped if all sequences in the testing set are classified correctly, as indicated by a dashed vertical line. When sequence length is 500, both LSTM and GRU failed within 50000 training steps, and their accuracy curves, which keep fluctuating around the baseline, are partially plotted.

Note that we did not use the truncated BPTT algorithm since the whole sequence was processed continuously in each training step.

In other words, gradients propagated back without being truncated and the credit assignment path can be as long as 10000 steps. In

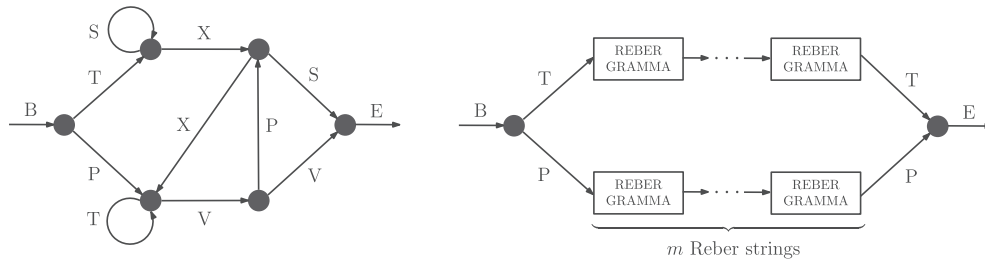


Fig. 4. Left: Transition diagram for the Reber grammar. Right: Transition diagram for the multi-embedded Reber grammar. Each box represents a Reber string.

GDU models, $\delta_i = 1$ applies to all groups. All models were implemented using Tensorflow [27] and codes for reproducing the results can be found here⁵.

4.1. The adding problem

The adding problem is a sequence regression problem originally proposed in [4] to examine the ability of recurrent models to capture long-term dependencies. Two sequences of length L are taken as input. The first one consists of real numbers sampled from a uniform distribution in $[0, 1]$. While the second sequence serves as indicators with exactly two entries being 1 and the remaining being 0. We followed the settings in [28] where L is a constant and the first 1 entry is located uniformly at random in the first half of the indicator sequence, whilst the second 1 entry is located uniformly at random in another half. The target of this problem is to add up the two entries in the first sequence whose corresponding indicator in the second sequence is 1. A naive strategy of outputting 1 regardless of the inputs yields a mean squared error of 0.167, which is the variance of the sum of two independent uniform distributions over $[0, 1]$. We took it as the baseline.

Four different lengths of sequences, $L \in \{200, 1000, 5000, 10000\}$ were used in this experiment. For each length, 500 sequences were generated for testing, while a batch of 20 sequences was randomly generated at each training step. The learning rate for all models was set to 0.01. Four models, an LSTM with 100 hidden states, a GRU with 100 hidden states, a GDU with 10 groups of size 10 and a GDU with only 1 group of size 10 were compared, with the corresponding parameter number 41.3K, 31.0K, 20.7K, and 271. A simple linear layer without activation is stacked on top of the recurrent layer in each model.

The results are shown in Fig. 2. Obviously, GRU outperforms LSTM in these trials. LSTM fails to converge within 10000 training steps when L is 1000 while GRU can learn this task within 1300 steps even trained with sequences of length 10000. Our GDU models perform slightly better than GRU with fewer parameters. As L increases, this advantage becomes more obvious. Note that a GDU with only one group of size 10 has comparable performance with a much bigger one, which indicates that GDU can efficiently capture simple long-term dependencies even with a tiny model.

4.2. The 3-bit temporal order problem

The 3-bit temporal order problem is a sequence classification problem to examine the ability of recurrent models to extract information conveyed by the temporal order of widely separated inputs of recurrent models [4]. The input sequence consists of randomly chosen symbols from the set $\{a, b, c, d\}$ except for three elements at position t_1, t_2 and t_3 , which are either X or Y . Each symbol is represented locally using 6 units, and position t_k is randomly

chosen between $\lfloor \frac{(k-1)L}{3} \rfloor$ and $\lfloor \frac{(k-1)L}{3} \rfloor + 10$, where $k = 1, 2, 3$, and L is the sequence length. The target is to classify the order (either XXX , XXY , XYX , YYY , YXX , YXY , YYX , YYY), which is represented using 8 units.

Four different lengths of sequences (i.e., $L \in \{100, 200, 500, 1000\}$) are used in this experiment. Same with the settings in 4.1, we generated 500 testing sequences for each length, and randomly generated a batch of 20 sequences for each training step. The learning rate for all models was set to 0.001. Accuracy is used as the metric on testing set, and the baseline is 0.125. We compared an LSTM model with 100 hidden states, a GRU model with 100 hidden states, and a GDU with 10 groups of size 10 on these datasets. The parameter numbers are 43.6K, 32.9K, and 22.2K, respectively.

The results are shown in Fig. 3. In contrast to the results of the adding problem, LSTM outperforms GRU on this task. However, both LSTM and GRU fail in learning to distinguish the temporal order when the sequence length increases to 500. The GDU model with $\mathcal{P}_{\alpha_i} = 0.1$ always starts learning earlier. When trained with relatively longer sequences, GDU outperforms these 2 models by a large margin with much fewer parameters.

4.3. Multi-embedded Reber grammar

Embedded Reber grammar (ERG) [29,4] is a good example containing dependencies with different time scales. This task needs RNNs to read strings, one symbol at a time, and to predict the next symbol (error signals occur at every time step). To correctly predict the symbol before last, a model has to remember the second symbol. However, since it allows for training sequences with short time lags (of as few as 9 steps), using it to evaluate a model's ability to learn long-term dependency is not appropriate. In order to make the training sequences longer, we modified the ERG by having multiple Reber strings embedded between the second and the last but one symbol (See Fig. 4).

We refer to this variant as the *multi-embedded Reber grammar* (*mERG*) and simply use the prefix m to indicate the number of embedded Reber strings. For example, “BT(BPVVE)(BTSXSE)(BTXXVVE)TE” is a 3ERG sequence. Since each Reber string has a minimal length of 5, the shortest *mERG* sequence has a length of $5m + 4$.

Learning *mERG* requires a recurrent model to have the ability to latch long-term memory while keeping mid- and short-term memory (provided m is big) in the meantime. Further, there may be two legal successors from a given symbol and the model will never be able to do a perfect job of prediction. During training, the rules defining the grammar are never presented. Thus the model will see contradictory examples, sometimes with one successor and sometimes the other, which requires it to learn to activate both legal outputs. What's more, a model must remember how many Reber strings it has read to make a correct prediction of the next symbol if the current symbol is an “E”. In other words, models must learn to **count**.

⁵ <https://github.com/WilliamRo/gdu>

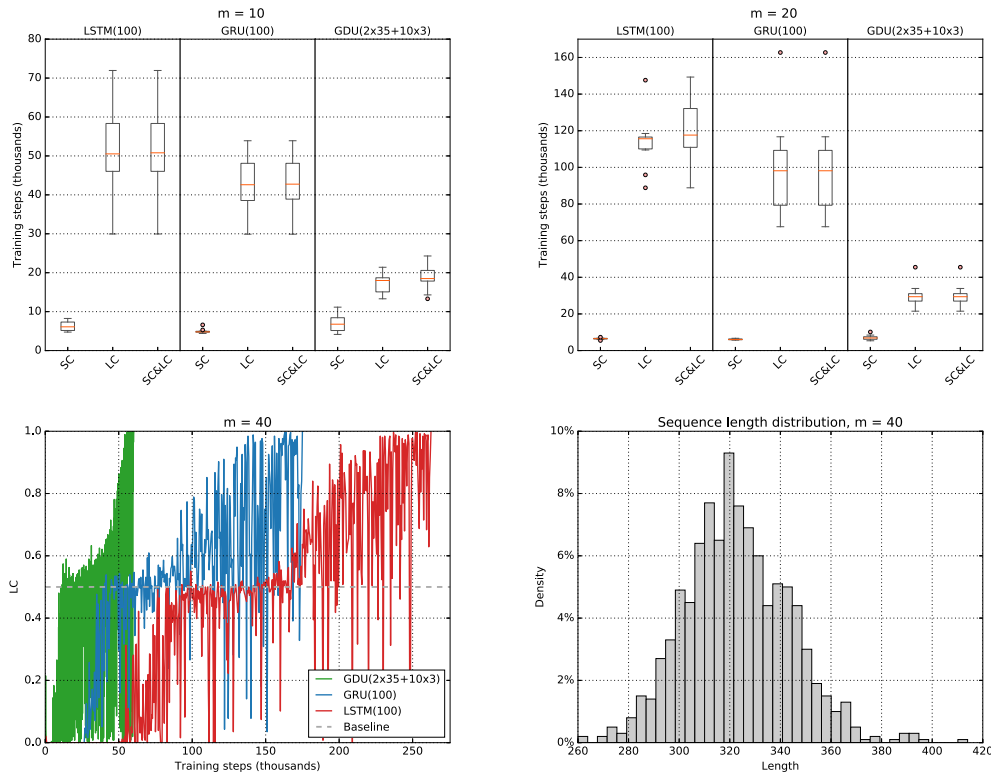


Fig. 5. The results of the multi-embedded Reber grammar. The **upper left and right** figures show the training steps each model takes to satisfy the criteria (reach to 1.0) for $m = 10$ or 20 . Each box-whisker (showing median, 25%, and 75% quantiles, minimum, maximum and outliers) contains the corresponding results of 10 trials. For $m = 40$ we only give the best results of each model in the **bottom left** figure. The **bottom right** figure shows the density histogram of sequence lengths in 40ERG training set.

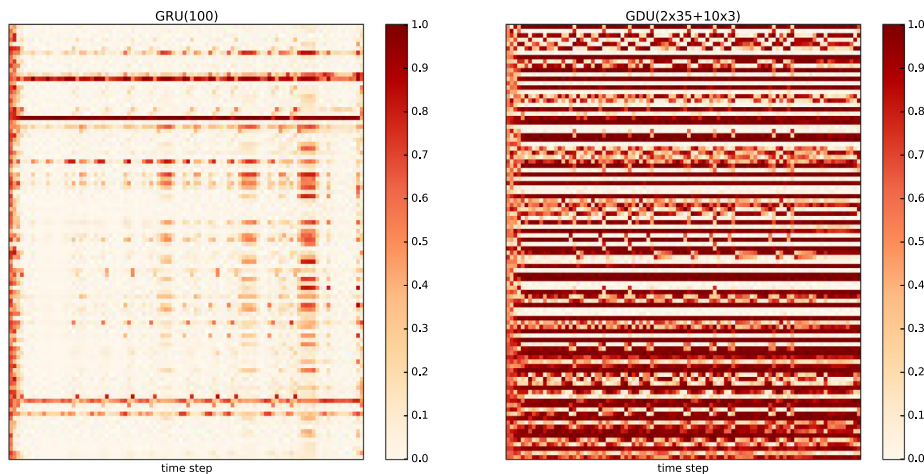


Fig. 6. The activation of \mathcal{G}_H of GRU(100) (**left**) and GDU($2 \times 35 + 10 \times 3$) (**right**) on a same sequence from 10ERG testing set. Each column corresponds to the gate activation at one time step. Each row with continuous dark color corresponds to a gate unit that keeps active and thus latches information.

We set m to be 10, 20 and 40 for this task, with the minimal sequence length being 54, 104, and 204, respectively. One sequence is given at a time and the learning rate of all models was set to 0.0003. As for the symbols with 2 legal successors, a prediction is considered correct if the two desired outputs are the two with the largest values. For each m we generated 1000 sequences for training and 256 sequences for testing. The sequences in the testing set are unique and have never appeared in the training set. The same training and testing sets are used for comparing all models.

We also defined two criteria to test the model's ability to capture long- and short-term dependencies separately. The one for short-term dependency is SC (short for short-term criterion) defined as the percentages of testing sequences each symbol of which is predicted correctly by the model except for the one before last. The other is LC (short for long-term criterion) defined as the percentages of testing sequences whose last but one symbol is predicted correctly. We stopped the training when both SC and LC are satisfied (reach to 1), namely, all symbols in all testing sequences are predicted correctly. A naive strategy of predicting the symbol

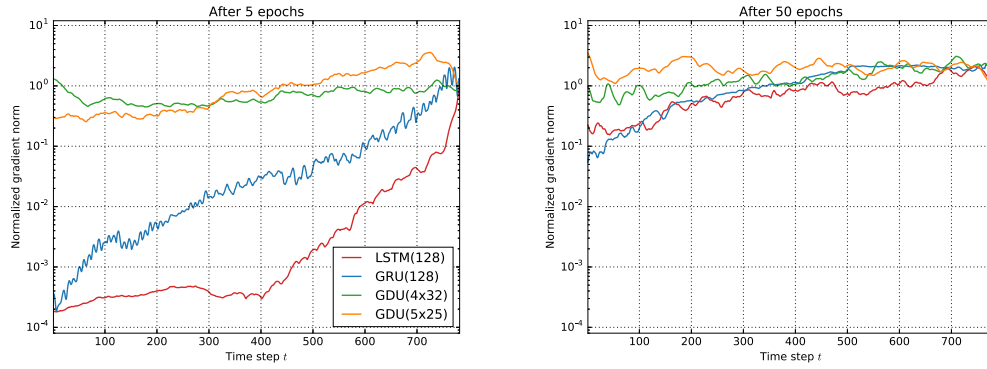


Fig. 7. Norms of the error signal back-propagated to each time step, i.e., $\|\epsilon_{t-784}\| = \|\frac{\partial \mathcal{L}}{\partial \mathbf{s}_t}\|$ after 5 epochs (**left**) and 50 epochs (**right**). For LSTM model, we calculate $\|\frac{\partial \mathcal{L}}{\partial \mathbf{s}_t}\|$ instead of $\|\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}\|$, where \mathbf{s}_t is a concatenation of \mathbf{s}_t and $\mathbf{h}_t = \gamma_t(\mathbf{s}_t)$.

Table 1
Results of sequential image classification on pMNIST.

Model	# parameters (\approx, K)	Test accuracy
Full-capacity uRNN [31]	270	94.1
Dilated GRU [10]	130	94.6
r-LSTM [32]	–	95.2
LSTM with Recurrent BN & Zoneout [33]	–	95.9
Independently RNN [34]	–	96.0
Att-LSTM [35]	–	96.2
GDU(5×51)	133.6	96.34

before last as T or P gives an expected LC of 0.5, which serves as the baseline.

An LSTM model and a GRU model both with 100 hidden states were chosen to be compared as previous experiment, with corresponding parameter numbers being 43.9 K and 33.1 K. As for GDU, we chose a model with 35 groups of size 2 and 3 groups of size 10 (denoted as GDU($2 \times 35 + 10 \times 3$)), having totally 100 hidden units and 22.3 K parameters.

From the results presented in Fig. 5, we can see for mERG, models always learn to capture the short-term dependencies first. While the long-term dependency is much more difficult to learn. GRU outperforms LSTM this time, no matter from the aspect of which criterion. GDU is slightly inferior to LSTM and GRU in terms of SC. However, on aspect of LC, it has an obvious advantage.

As discussed in Section 2, learning to latch long-term information in the presence of short-term dependencies is difficult for a traditional GAST model due to the gradient conflict. GDU greatly alleviate this problem by limiting \mathcal{P}_{α_t} in cGAST, namely the proportion of states to be overwritten, which results in a broader “bandwidth” for long-term information flow. Fig. 6 illustrates this by visualizing the \mathcal{G}_{β_t} activation of GAST models on a same 10ERG sequence after the LC has been satisfied.

4.4. Sequential pMNIST classification

The sequential MNIST task [8] can be seen as a sequence classification task in which 28×28 MNIST images [30] of 10 digits are read pixel by pixel from left to right, top to bottom. While the sequential pMNIST [8] is a challenging variant where pixels in each image are permuted by the same randomly generated permutation matrix. This creates many longer term dependencies across pixels, which makes it necessary for a model to learn and remember more complicated dependencies embedded in varying time scales.

For this task we used a GDU model with 255 state units, which are divided evenly into 51 groups. The network was trained with a

learning rate of 0.001 and a mini-batch size of 100. Training was stopped when validation accuracy did not increase for 5 epochs. Furthermore, recurrent dropout [36] of 0.4 was used on state transition in each time step and forward dropout [37] of 0.2 was used on the output of the recurrent layer. The gradients were clipped at value 1.0 to avoid the exploding gradients. The results are shown in Table 1 in comparison with other methods that were recently proposed. It can be seen that GDU achieved better performance than those models.

4.4.1. A comparison of gradients

As discussed in Section 2, controlling $\|\frac{\partial \mathcal{L}}{\partial \mathbf{s}_t}\|$ is the key to avoid the vanishing gradient issue, so that long-term dependencies can be learned. We explored how models propagate gradients by examining $\|\frac{\partial \mathcal{L}}{\partial \mathbf{s}_t}\|$ as a function of t , where \mathcal{L} is the prediction loss. Four models, including LSTM(128), GRU(128), GDU(4x32), and GDU (5x25), were trained for this gradient study. Learning rate and batch size were set to 0.001 and 100, respectively. Gradient norms were computed after 5 and 50 epochs, and the normalized curves are plotted in Fig. 7. For LSTM and GRU, we can see that error signals have trouble in reaching far from where they are injected at the early stage. This problem is reduced after dozens of epochs. GDU models have better gradient properties than LSTM and GRU because of the distributor mechanism in Eqs. (18).

4.5. Spoken word recognition

The last task is to classify audio sequences of spoken words from the TIMIT Speech Recognition Benchmark [38]. The dataset, including 25 different words (classes) arranged in 5 clusters, was constructed as described in [15], and there are 7 examples from different speakers in each class. Words in each cluster share the same suffix, which requires the networks to learn long-term dependencies in order to distinguish them.

Following [15], we randomly selected 5 words for training and 2 words for testing for every class. Further, each audio sequence was preprocessed into a 12-dimensional MFCC vector [39] plus energy, sampled every 10 ms over a 25 ms window with a pre-emphasis coefficient of 0.97.

In this task, the proposed GDU was compared with GRU, AMU, and ON-LSTM. For each type of model, networks with 1–3 stacked recurrent layers were evaluated and the layer sizes were chosen to keep the total number of parameters at around 10 K. All networks used a learning rate of 3×10^{-3} except for AMUs, which used 8×10^{-4} . During training, Gaussian noise with a standard deviation of 0.6 was added to the inputs and the training was stopped once the cross-entropy error on the noise-free training set did not decrease for 7 epochs. No other tricks such as dropout and gradient

Table 2

Mean error and standard deviation (averaged over 100 runs) on both training and testing set for each model on the spoken word recognition task. Number of recurrent units in each layer for each model is also given. For GDU models, group configs are specified. For AMU models, $S \times N$ means each layer contains N AMUs of size S . Note that the unit size for AMUs was set to 3. All networks have roughly 10 K trainable parameters.

Model	# of layers	Units per layer	Training error %	Test error %
GRU	1	48	0.3 ± 1.5	63.5 ± 6.2
	2	30	0.3 ± 0.5	64.6 ± 6.8
	3	24	1.8 ± 1.2	54.3 ± 8.0
ON-LSTM	1	33	0.5 ± 0.7	59.2 ± 7.4
	2	21	4.7 ± 3.6	55.5 ± 8.1
	3	17	16.6 ± 9.6	51.9 ± 7.9
AMU	1	3×30	4.9 ± 4.0	38.7 ± 7.4
	2	3×20	15.5 ± 11.4	46.0 ± 12.2
	3	3×16	41.1 ± 19.2	62.5 ± 14.4
GDU	1	$15 \times 2 + 5 \times 5 + 3 \times 1$	0.2 ± 0.8	19.6 ± 6.3
	2	$15 \times 2 + 5 \times 1 + 2 \times 1$	1.0 ± 2.0	17.9 ± 5.4
	3	$15 \times 1 + 12 \times 1 + 2 \times 1$	0.4 ± 1.1	17.3 ± 4.9

clipping were used. At the end of each trial, the error rate on the test set was evaluated.

Experiment on each network was repeated 100 times with different random initializations, and the results are summarized in Table 2. It can be seen that except for AMU, models with stacked recurrent layers yield more competitive results. GDUs beat all other models by a considerable margin irrespective of the number of recurrent layers. Unlike AMUs and ON-LSTMs, stacked GDUs have similar performance on the training set compared to 1-layer GDUs. Further, stacked structure leads to better generalization ability for GDUs on this task.

5. Conclusions and future work

We proposed a novel RNN architecture with gated additive state transition, which contains only one gate unit. The issues of gradient vanishing and conflict are mitigated by explicitly limiting the proportion of states to be overwritten at each time step. Our experiments comprise challenging pathological problems and tasks on natural datasets. The results were consistent over different tasks and clearly demonstrated that the proposed grouped distributor architecture is helpful to extract long-term dependencies embedded in data.

A plethora of further ideas can be explored based on our findings. For example, various combinations of groups with different sizes and overwrite proportions can be explored. Further, the overwrite proportion δ can be trained. What's more interesting is that the grouped distributor structure can be used spatially to ease gradient-based training of very deep networks. To be more specific, this work can base on the *highway network* [40] in which the distributor operator can be used to calculate the *transform gate*.

CRedit authorship contribution statement

Wei Luo: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Feng Yu:** Conceptualization, Writing - review & editing, Supervision, Project administration, Funding acquisition.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

- [1] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, *Nature* 323 (1986) 533–536, <https://doi.org/10.1038/323533a0>.
- [2] P.J. Werbos, Generalization of backpropagation with application to a recurrent gas market model, *Neural Networks* 1 (1988) 339–356, [https://doi.org/10.1016/0893-6080\(88\)90007-X](https://doi.org/10.1016/0893-6080(88)90007-X).
- [3] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*, in: S.C. Kremer, J.F. Kolen (Eds.), *Field Guide to Dynamical Recurrent Networks*, IEEE Press, 2001.
- [4] S. Hochreiter, J. Schmidhuber, Long short-term memory, *Neural Comput.* 9 (8) (1997) 1735–1780, <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [5] J. Martens, Deep learning via hessian-free optimization, in: *Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10*, Omnipress, USA, 2010, pp. 735–742. URL <http://dl.acm.org/citation.cfm?id=3104322.3104416>.
- [6] J. Martens, I. Sutskever, Learning recurrent neural networks with hessian-free optimization, in: *Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11*, Omnipress, USA, 2011, pp. 1033–1040. URL <http://dl.acm.org/citation.cfm?id=3104482.3104612>.
- [7] A.M. Saxe, J.L. McClelland, S. Ganguli, Exact solutions to the nonlinear dynamics of learning in deep linear neural networks (2013). arXiv:1312.6120.
- [8] Q.V. Le, N. Jaitly, G.E. Hinton, A simple way to initialize recurrent networks of rectified linear units (2015). arXiv:1504.00941.
- [9] T. Lin, B.G. Horne, P. Tino, C.L. Giles, Learning long-term dependencies in narx recurrent neural networks, *Trans. Neur. Netw.* 7 (6) (1996) 1329–1338, <https://doi.org/10.1109/72.548162>.
- [10] S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M.A. Hasegawa-Johnson, T.S. Huang, Dilated recurrent neural networks, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30*, Curran Associates Inc, 2017, pp. 77–87. <http://papers.nips.cc/paper/6613-dilated-recurrent-neural-networks.pdf>.
- [11] R. S. DiPietro, C. Rupprecht, N. Navab, G.D. Hager, Analyzing and exploiting narx recurrent neural networks for long-term dependencies., in: *ICLR (Workshop)*, OpenReview.net, 2018. URL <http://dblp.uni-trier.de/db/conf/iclr/iclr2018w.html#DiPietroRNH18>.
- [12] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation (2014). arXiv:1406.1078.
- [13] J. Collins, J. Sohl-Dickstein, D. Sussillo, Capacity and trainability in recurrent neural networks., in: *ICLR (Poster)*, OpenReview.net, 2017. URL <http://dblp.uni-trier.de/db/conf/iclr/iclr2017.html#CollinsSS17>.
- [14] G.-B. Zhou, J. Wu, C.-L. Zhang, Z.-H. Zhou, Minimal gated unit for recurrent neural networks (2016). arXiv:1603.09420.
- [15] J. Koutnik, K. Greff, F. Gomez, J. Schmidhuber, A clockwork RNN, in: E. P. Xing, T. Jebara (Eds.), *Proceedings of the 31st International Conference on Machine Learning*, Vol. 32 of *Proceedings of Machine Learning Research*, PMLR, Beijing, China, 2014, pp. 1863–1871.
- [16] J.L. Elman, *Finding structure in time*, *Cognitive Sci.* 14 (2) (1990) 179–211.
- [17] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13, JMLR*.

- org, 2013, pp. III–1310–III–1318. URL <http://dl.acm.org/citation.cfm?id=3042817.3043083>.
- [18] F.A. Gers, J.A. Schmidhuber, F.A. Cummins, Learning to forget: Continual prediction with lstm, *Neural Comput.* 12 (10) (2000) 2451–2471, <https://doi.org/10.1162/089976600300015015>.
- [19] F. A. Gers, J. Schmidhuber, Recurrent nets that time and count, in: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, vol. 3, IEEE, 2000, pp. 189–194 vol 3. doi:10.1109/ijcnn.2000.861302. doi: 10.1109/ijcnn.2000.861302..
- [20] K. Greff, R.K. Srivastava, J. Koutník, B.R. Steunebrink, J. Schmidhuber, Lstm: a search space odyssey, *IEEE Trans. Neural Networks Learn. Syst.* 28 (8) (2017) 2222–2232, <https://doi.org/10.1109/TNNLS.2016.2582924>.
- [21] J. Chung, Ç. Gülçehre, K. Cho, Y. Bengio, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, in: NIPS 2014 Workshop on Deep Learning and Representation Learning, DLW 2014, 2014. <http://www.dlworkshop.org/47.pdf?attredirects=0..>
- [22] Y. Shen, S. Tan, A. Sordoni, A. Courville, Ordered neurons: Integrating tree structures into recurrent neural networks (2018). arXiv:1810.09536..
- [23] S.E. Hiji, Y. Bengio, Hierarchical recurrent neural networks for long-term dependencies, in: D.S. Touretzky, M. Mozer, M.E. Hasselmo (Eds.), NIPS, MIT Press, 1995, pp. 493–499. URL <http://dblp.uni-trier.de/db/conf/nips/nips1995.html#HijiB95>.
- [24] J. Wang, L. Zhang, Q. Guo, Z. Yi, Recurrent neural networks with auxiliary memory units, *IEEE Trans. Neural Networks Learn. Syst.* 29 (2018) 1652–1661.
- [25] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Y.W. Teh, D.M. Titterton (Eds.), AISTATS, Vol. 9 of JMLR Proceedings, JMLR.org, 2010, pp. 249–256. <http://dblp.uni-trier.de/db/journals/jmlr/jmlr9.html#GlorotB10>.
- [26] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2014). arXiv:1412.6980..
- [27] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow: Large-scale machine learning on heterogeneous distributed systems (2015). <http://download.tensorflow.org/paper/whitepaper2015.pdf>.
- [28] M. Arjovsky, A. Shah, Y. Bengio, Unitary evolution recurrent neural networks (2015). arXiv:1511.06464..
- [29] S.E. Fahlman, The recurrent cascade-correlation architecture, in: R.P. Lippmann, J.E. Moody, D.S. Touretzky (Eds.), Advances in Neural Information Processing Systems 3, Morgan-Kaufmann, 1991, pp. 190–196. URL <http://papers.nips.cc/paper/330-the-recurrent-cascade-correlation-architecture.pdf>.
- [30] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, *Proc. IEEE* 86 (11) (1998) 2278–2324, <https://doi.org/10.1109/5.726791>.
- [31] S. Wisdom, T. Powers, J. R. Hershey, J.L. Roux, L. Atlas, Full-capacity unitary recurrent neural networks (2016). arXiv:1611.00035..
- [32] T. H. Trinh, A. M. Dai, M.-T. Luong, Q. V. Le, Learning longer-term dependencies in rnns with auxiliary losses (2018). arXiv:1803.00144..
- [33] D. Krueger, T. Maharaj, J. Kramér, M. Pezeshki, N. Ballas, N.R. Ke, A. Goyal, Y. Bengio, A. Courville, C. Pal, Zoneout: Regularizing RNNs by randomly preserving hidden activations (2016). arXiv:1606.01305..
- [34] S. Li, W. Li, C. Cook, C. Zhu, Y. Gao, Independently recurrent neural network (indrn): Building a longer and deeper rnn (2018). arXiv:1803.04831..
- [35] W. Xia, W. Zhu, B. Liao, M. Chen, L. Cai, L. Huang, Novel architecture for long short-term memory used in question classification, *Neurocomputing* 299 (2018) 20–31, <https://doi.org/10.1016/j.neucom.2018.03.020>.
- [36] S. Semeniuta, A. Severyn, E. Barth, Recurrent dropout without memory loss (2016). arXiv:1603.05118..
- [37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, *J. Mach. Learn. Res.* 15 (2014) 1929–1958, URL: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [38] J.S. Garofolo, L.F. Lamel, W.M. Fisher, J.G. Fiscus, D.S. Pallett, N.L. Dahlgren, Darpa timit acoustic-phonetic continuous speech corpus, CD-ROM (1993).
- [39] P. Mermelstein, Distance measures for speech recognition, psychological and instrumental, in: C.H. Chen (Ed.), Pattern Recognition and Artificial Intelligence, Academic Press, New York, 1976, pp. 374–388.
- [40] R.K. Srivastava, K. Greff, J. Schmidhuber, Highway networks (2015). arXiv:1505.00387..



Wei Luo received his B.Sc. degree in Information and Computing Science from Zhejiang University, Hangzhou, China, in 2014. He is currently pursuing the Ph.D. degree in Zhejiang University. His research interests include machine learning, in particular neural networks and optimization.



Dr. Feng Yu is a Professor in the College of Biomedical Engineering and Instrument Science at Zhejiang University. He has a BS in Semiconductor from Nankai University and MS/PhD in Instrumentation Engineering from Zhejiang University. His current primary research interests are in novel computational architectures for biomedical image processing applications.