Learning to Selectively Update State Neurons in Recurrent Networks

Thomas Hartvigsen
Worcester Polytechnic Institute
twhartvigsen@wpi.edu

Cansu Sen
Worcester Polytechnic Institute
csen@wpi.edu

Xiangnan Kong
Worcester Polytechnic Institute
xkong@wpi.edu

Elke Rundensteiner
Worcester Polytechnic Institute
rundenst@wpi.edu

ABSTRACT

Recurrent Neural Networks (RNNs) are the state-of-the-art approach to sequential learning. However, standard RNNs use the same amount of computation to generate their hidden states at each timestep, regardless of the input data. Recent works have begun to tackle this rigid assumption by imposing a priori-determined patterns for updating the states at each step. These approaches could lend insights into the dynamics of RNNs and possibly speed up inference. However, the pre-determined nature of the current update strategies limits their application. To overcome this, we instead design the first fully-learned approach, SA-RNN, that augments any RNN by predicting discrete update patterns at the fine granularity of individual hidden state neurons. This is achieved through the parameterization of a distribution of update-likelihoods driven by the input data. Unlike related methods, our approach imposes no assumptions on the structure of the update patterns. Better yet, our method adapts its update patterns online, allowing different dimensions to be updated conditionally based on the input. To learn which dimensions to update, the model solves a multi-objective optimization problem, maximizing task performance while minimizing the number of updates based on a unified control. Using five publicly-available datasets spanning three sequential learning settings, we demonstrate that our method consistently achieves higher accuracy with fewer updates compared to state-of-the-art alternatives. We also show the benefits of learning to sparsely-update a large hidden state as opposed to densely-update a small hidden state. As an added benefit, our method can be directly applied to a wide variety of models containing RNN architectures.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Supervised learning**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM '20, October 19–23, 2020, Virtual Event, Ireland
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412018

KEYWORDS

Recurrent Neural Networks, Conditional Computation, Sequential Data

ACM Reference Format:
Thomas Hartvigsen, Cansu Sen, Xiangnan Kong, and Elke Rundensteiner. 2020. Learning to Selectively Update State Neurons in Recurrent Networks. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3412018

1 INTRODUCTION

Background on Sequential Learning. Recurrent Neural Networks (RNN) are the state-of-the-art approach to many sequential learning problems including language modeling [26], machine translation [36], and sequence generation [12, 37]. However, RNNs typically rely on often computationally-taxing updates to their entire hidden state at each timestep, a cost that grows with hidden state size. As demonstrated by the success of gating mechanisms such as the GRU [8] and LSTM [16], all dimensions rarely need to be re-computed from scratch at each timestep. By *discretely* selecting which dimensions to update at each timestep via a learned *update pattern*, RNNs with a large hidden state could potentially be trained with lower computational requirements [3, 27], inference in long RNNs may be expedited [5], and hidden representations might be made more robust to misleading inputs such as outliers or noise.

State-of-the-Art and Limitations. Selective activation in RNNs has recently gained attention in the literature [5, 17, 20, 27, 31]. The most popular methods hand-craft specific *update patterns*, dictating which dimensions of the hidden state will update at which timesteps according to prior knowledge of a task [20, 27]. This imposes undue challenges in implementation, limits extensibility, and ignores the data-driven curation of information-flow through the RNN, a signature property of recurrent memory cells [8, 16]. More recent methods learn to react to input data but impose strict relationships between the update patterns across both hidden dimensions and time [5, 17, 31]. Designed for tasks with clear hierarchical components, such as modeling character-level text [9, 22], this hierarchical structure in update patterns may limit the expressiveness of learned update patterns for tasks where this assumption is not applicable.

Problem Description. Specifically, we study the problem of generating a binary *update-pattern* for the hidden states learned by an RNN given input sequential data. The learned update-pattern defines which dimensions of the hidden state to update at each

timestep, similar to the motivation for Residual Networks [14, 34] and Highway Networks [32, 38]. Ideally, only a small subset of the hidden state's dimensions needs to be updated at each timestep, which is especially important for high-dimensional hidden states. In this way, powerful representations can be learned while both solving a sequential learning task and minimizing the number of updates. This can result in a reduction of the overall computational time. A solution to this multi-objective optimization problem should have a comparable accuracy to a traditional constantly-updating RNN, while at the same time saving the majority of (unnecessary) computation steps along the way, ultimately targeting accelerated training and inference [27].

Technical Challenges. Despite the potential for reducing computation required by RNNs, learning update patterns is a challenging problem for the three following reasons:

- First, binary-output neurons making discrete decisions, such as whether or not to update a hidden state dimension, in the interior of a neural network is a classic challenge to gradient-based learning. Such decisions are by nature non-differentiable and therefore back-propagation cannot be directly used for training [3]. To generate flexible and data-reactive update patterns (which are by nature binary), the network must learn to make such discrete decisions.
- Second, the quality of an update pattern is unsupervised and thus the only feedback available is task-specific. This lack of supervision discourages a priori assumptions of the update patterns, which may not be optimal for a particular task.
- Third, task performance and sparse updates to the hidden state are contradictory goals. An RNN that never updates its hidden state will by definition learn nothing about the input sequence. Balancing these two targets is a multi-objective optimization problem, which must be balanced case-by-case.

Our Proposed Approach: SA-RNN. To address the aforementioned challenges, we propose the *Selective-Activation* RNN, or SA-RNN, which parameterizes a distribution of update-likelihoods, one per hidden state dimension, from which update-decisions can be made at each timestep, removing the need for a priori assumptions on the form of the update decisions. We achieve this by augmenting an RNN with an update *Coordinator* that adaptively controls which coordinate directions to update in the hidden state on the fly. The *Coordinator* is modeled as a lightweight neural network that observes incoming data at each timestep and makes a discrete decision about whether or not enough information is stored in each individual hidden dimension to warrant an update. Subsequently, each hidden dimension is either computed by the RNN or copied from the previous timestep, avoiding the RNN update.

The *Coordinator* is kept as simple as possible so the complexity of the RNN can scale without outsourcing computation to another network, similar to the controller in [13]. In contrast to other recent approaches [5, 17, 20, 24, 27, 31] we impose no assumptions about which individual hidden dimensions should or should not be updated together. Instead, we illustrate that using an entirely-learned approach indeed results in complex task-specific update patterns.

Contributions Our contributions can be described as follows:

- Ours is the first method to predict fully-learned update patterns for RNNs. Our approach relaxes assumptions of previous methods and demonstrates that this strategy is a feasible and lower-bias approach to reducing computation in RNNs.
- We compare against four state-of-the-art methods and two baselines on five datasets spanning three sequential learning settings. On all tasks, SA-RNN achieves better task performance while using far fewer updates to its hidden states.
- Our results show that sparsely updating a large hidden state is superior to densely updating small hidden states in RNNs.

2 RELATED WORK

Recurrent neuron update patterns have gained much interest in recent literature [9, 17, 20, 24, 27, 31]. All of these methods boast fewer updates to the hidden states than standard RNN architectures. However, there are several limitations of these methods.

First, the most popular methods rely on extensively-handcrafted update patterns consisting of periodic neuron activations [20, 24, 27]. This requires either prior knowledge of sampling frequencies or seasonal patterns present in the data, reducing the potential extension to many sequential learning problems. Clockwork RNN [20] requires hand-picking rates at which to update fixed-sized contiguous blocks of a vanilla RNN's hidden state. PhasedLSTM [27] and its variant, HE-LSTM [24], sample rates at which to update independent neurons, then learn individual periods for each neuron on the training set which remains fixed during testing. These input-agnostic periodic updates are fixed prior to inference and thus remain rigid, failing to adapt their update patterns, regardless of the input data. The choice of update periods heavily impacts the performance of the model, and sequences with irregular information flow are challenging to model without massive states and carefully hand-picked parameters.

Second, more recent works allow for *data-reactive* update patterns [5, 17, 31] but assume temporal hierarchies in the input sequences and typically study settings where this effect is exceedingly obvious (for example, character-level sentence modeling [9, 22]). In many real-world settings, temporal hierarchies are often subtle and forcing this assumption into the architectural design may limit application. VC-RNN [17] predicts a proportion $p \in [0, 1]$ of the hidden state to update, then updates that proportion of neurons, ordered by index. SkipRNN [5] predicts a binary value $p \in \{0, 1\}$, deciding whether or not to update the entire hidden state, updating all neurons together in lock-step. SkipRNN also assumes that the likelihood of updating the state increases monotonically, which may not be appropriate for a variety of settings. This assumption that one neuron's update should be conditional to its neighbors is unnecessary since the dimensions of the hidden state are not computed with respect to their location in the network, and does not need to be reflected in the update pattern.

Our work is also related to *conditional computation* in neural networks, which selectively activates different sub-networks of large neural networks, conditioned on input data [2, 7, 23]. This area has recently gained much interest as it could extend the application of deep neural networks without increasing the size of the models.

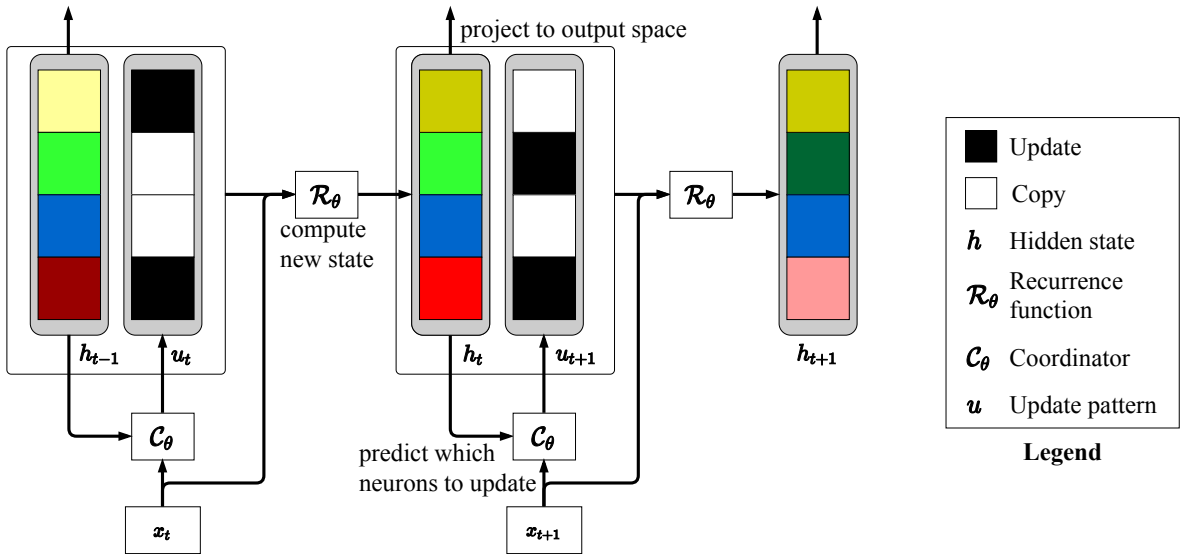


Figure 1: Computing new hidden states with SA-RNN. Prior to computing h_t, the hidden state at timestep t, the *Coordinator* C_θ predicts a binary mask given current input X_t and previous hidden state h_{t-1}, determining which dimensions of h_t should be computed anew. The *Recurrence function* $\mathcal{R}_\theta$ is typically a gated recurrent memory cell, such as a GRU or LSTM. C_θ is modeled as a light-weight neural network and can augment a wide variety of RNN architectures. In this example, when computing h_t, hidden dimensions 1 and 4 are updated, while dimensions 2 and 3 remain unchanged. This figure is best viewed in color.

In many cases, when a concept can be represented by only a sub-network of a large neural network, computation can be preserved by learning to activate only the said sub-network [29, 30].

3 SELECTIVE ACTIVATION FOR RECURRENT NEURAL NETWORKS

We introduce the **S**elective-**A**ctivation **RNN**, or SA-RNN, a broadly-applicable augmentation to RNNs which minimizes the computation required for RNNs by facilitating unimpeded information-flow across timesteps for individual dimensions of the hidden state. At its core, SA-RNN learns a data-driven strategy for discretely reading and writing information to the latent state space through the learned parameterization of an *update-likelihood* distribution. Despite leaving hidden dimension update patterns independent from one another, complex strategies still arise naturally depending on the sequential learning task at hand. In this section, we describe the training process of SA-RNN with D-dimensional hidden states on sequences of length T for input data x with V variables. We omit biases from affine transformation equations and use notation for one training instance for ease of readability. An overview of the forward pass through SA-RNN is shown in Figure 1.

3.1 Computing Hidden States

RNNs compute a sequence of hidden states one timestep at a time [10], each computed by a parametric transition function $\mathcal{R}(\cdot)$: $h_t = \mathcal{R}(h_{t-1}, x_t|\theta_r)$. The result is a sequence of vector representations $H = \{h_1, \ldots, h_T\}$ where each $h_t \in \mathbb{R}^D$ represents temporal dynamics of an input sequence up to step t with respect to a task, preserving not only temporal dependencies but also the ordering of the inputs.

The most popular augmentations to the RNN add a series of gates between h_{t-1} and h_t to alleviate the vanishing gradient problem [4]. For example, the Gated Recurrent Unit (GRU) is composed of an intuitive set of gating functions that balance newly-computed hidden states h_t with previous states h_{t-1}:

$$r_t = \sigma(W_r h_{t-1} + U_r x_t) \quad (1)$$

$$z_t = \sigma(W_z h_{t-1} + U_z x_t) \quad (2)$$

$$s_t = \phi(W_c x_t + U_c(r_t \odot h_{t-1})) \quad (3)$$

$$\tilde{h}_t = (1 - z_t) \odot h_{t-1} + z_t \odot s_t \quad (4)$$

where Ws and Us are matrices of learnable parameters of shape $D \times D$ and $D \times V$ respectively, $x_t \in \mathbb{R}^V$ is the input data at timestep t, $\odot$ represents the element-wise multiplication, σ represents the sigmoid function, and ϕ represents a non-linearity (traditionally the hyperbolic tangent function). Its design is motivated heavily by the LSTM [16]. Similar to the LSTM, the GRU performs soft read/write operations, recomputing the entire vector h_t at each timestep since gate $z \in [0, 1]^D$, the space of vectors with values inclusively between 0 and 1. Instead, we propose that all dimensions do *not* need to be updated at each timestep, as the position of the hidden state in many dimensions may often encode enough of the modeled input. In the next section, we describe how to compute h_t, the final hidden state for timestep t which is subsequently used to compute h_{t+1}.

3.2 Selective Neuron Activation

To reduce computation required to generate state representations, we update representations via a sequence of binary decisions – at each step, each neuron will either be updated or not. Intuitively

the usefulness of the hidden state will often not be decayed by allowing neurons to remain unchanged regardless of input data. As shown in Figure 2, not all dimensions of the hidden state change to the same degree when updated. For dimensions that would not change much either way, such as dimension 2 in our example, there is little geometric difference between updating h_2 or leaving it unchanged. Thus, we propose a learned *update coordinator*, which generates a binary mask for each hidden dimension, forecasting which dimensions need to be updated at the next timestep.

First, an *update-likelihood* $\tilde{u}_t$ is computed for each neuron, informed by both the data observed at the current timestep and the previous update-likelihoods: $\tilde{u}_t = \sigma(W_u h_{t-1} + W_i x_t)$ where $W_u \in \mathbb{R}^{D \times D}$ is a diagonal matrix of trainable parameters which dictate the linear relationship between h_{t-1} and $\tilde{u}_t$. W_u is kept diagonal to maintain relationships between update-decisions of the same dimension while avoiding the extensive computation of a fully-connected layer, similar to hidden state decay [6]. $W_i \in \mathbb{R}^{D \times V}$ encodes the influence of the input data on the current *update-likelihood* and $\sigma(\cdot)$ represents the hard sigmoid function, bounding $\tilde{u}$ according to a slope α. Thus $\tilde{u}_t \in [0, 1]^D$, with one *update-likelihood* per hidden dimension.

To discretize $\tilde{u}_t$, allowing information to flow unimpeded, element-wise binarization is applied:

$$u_t = \text{binarize}(\tilde{u}_t), \text{where} \tag{5}$$

$$\text{binarize}(a) = \begin{cases} 1 & \text{if } a > 0.5, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Intuitively, we then apply this final discrete *update decision* as a binary gating mechanism since $u_t \in \{0, 1\}^D$, augmenting the previously-described GRU transition function:

$$h_t = u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1} \tag{7}$$

In practice, the computation of the new hidden state can be directed at only the needed updates once u_t has been calculated, and the binary gating between $\tilde{h}_t$ and h_{t-1} can be avoided. This idea is similar to the *input* gate, in the LSTM cell, which decides how much of the new input information to add to the existing hidden state. However, the LSTM's input gate still updates every dimension of the hidden state to some degree. To reduce computation in the hidden states, our approach instead uses binary update decisions. Thus when $\tilde{u}_t^n$, the *update decision* for the n-th dimension in h, is 1, h_t^n is updated according to the new information present in $\tilde{h}_t^n$.

We note that this update-decision strategy does not impose the inter-neuron assumptions of [5, 17, 20, 22, 31] while still allowing such strategies to be learned if they are found to be optimal by the model since decisions are made with respect to previous decisions. We hypothesize that updating dimensions in groups may generally be beneficial since complex temporal dependencies often require high-dimensional representations, as discussed in [20].

Since binary output transformations are inherently not differentiable, barring the direct use of back-propagation, we approximate the gradient of the binarization via straight-through gradient estimation [3], ensuring no added computational cost:

$$\frac{\partial \text{binarize}(x)}{\partial x} = 1. \tag{8}$$

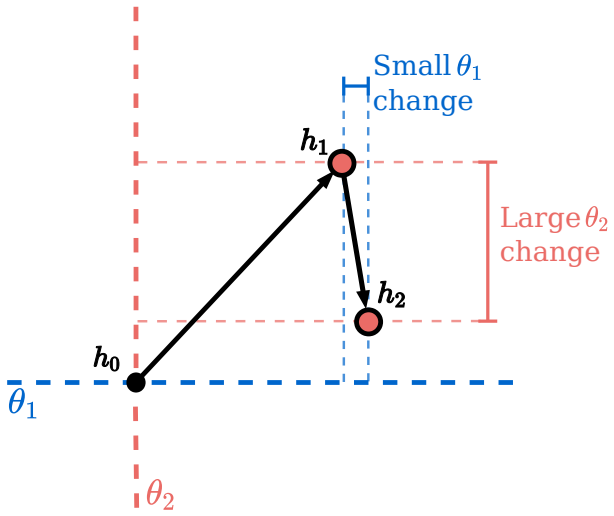
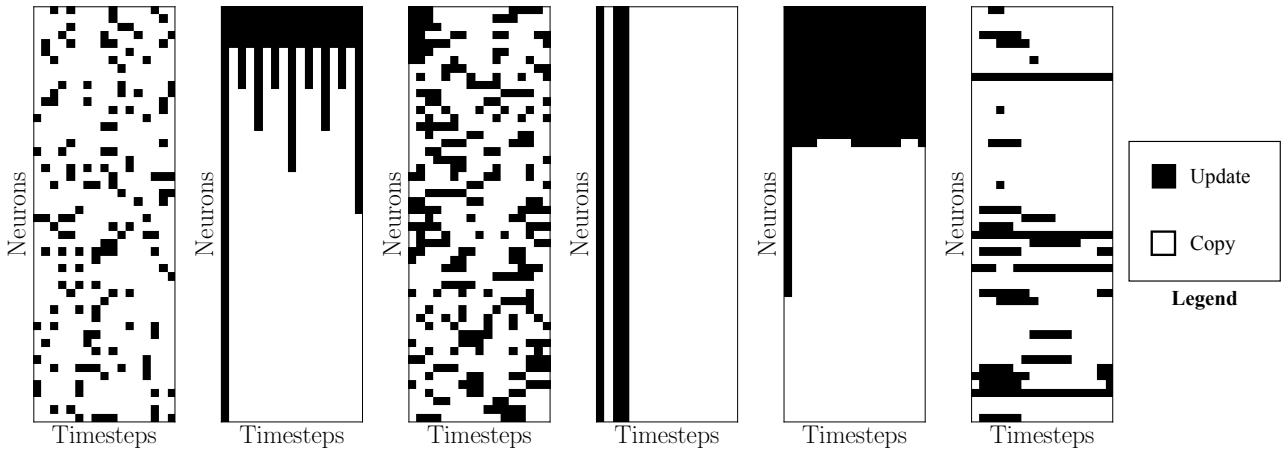


Figure 2: RNNs learn to project sequential data into sequences of hidden state vectors one step at a time. In this example, when transitioning from an RNN's hidden state h_1 to h_2, dimension 2 of h changes much more than dimension 1. Since dimension 1 changes only slightly, representational capacity of the RNN is inhibited only slightly (or not at all) by skipping this update.

By estimating the gradient in this way we avoid additional loss terms and end up with empirically superior approximations in comparison to other higher-variance methods, such as REINFORCE [5, 9, 35]. For policy gradient methods like REINFORCE, the reward function is often challenging to design and computational complexity can be significant [22]. Additionally, the relationships between particular dimensions of the hidden state and their relevance to the task are complex and latent, limiting the effectiveness of such sampling approaches to update-pattern selection. We further reduce estimation bias through *slope annealing* during training, increasing the slope α of the hard sigma according to the schedule $a = \min(5, 1 + 0.04 * N_{epoch})$ as in [9]. In Section 4.5 we also demonstrate that sampling update patterns leads to poor prediction performance. After computing the sequence of state representations H, they are projected into an output space via affine transformation possibly followed by a non-linearity depending on the task at hand, thus computing the predictions made by the model $\hat{y}$.

3.3 Training

All weights of SA-RNN are updated together using back-propagation and gradient descent to minimize one joint loss function. For readability, we gather all weights into one parameter matrix θ. Our loss function $J(\theta)$ consists of two parts: a task-driven loss (denoted as $\mathcal{L}_{\text{task}}$) and an update-budget. The *task-driven* component depends entirely on the task (for example, cross entropy for classification or mean squared error for regression). The *update-budget* encourages sparser updates to the hidden dimensions, as shown in Equation 9 where N is the number of training examples, $\hat{y}$ is the model's



(a) Random (b) CW-RNN (c) Phased LSTM (d) SkipRNN (e) VC-GRU (f) SA-RNN

Figure 3: Sample skipping patterns from compared methods for the `Seizures` task with 50-dimensional state representations. Black squares indicate *update* while white squares indicate *copy*. Time progresses from left to right. The standard RNN would be all black.

prediction, and y is the label.

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{L}_{task}(y^{(i)}, \hat{y}^{(i)}) + \lambda \sum_{t=1}^{T} \tilde{u}_t^i\right) \tag{9}$$

λ tunes the emphasis on the minimization of $\tilde{u}$, the *update probability*, and in practice it is reasonable to set $\lambda = 0$, encouraging the model to make *update decisions solely with respect to the learning task*. As λ is increased and predicted update likelihoods thus decrease, the hidden dimension is update fewer times until eventually $h_0 = h_T$ where the hidden state is not updated at all. Additionally, since $\mathcal{L}_{\text{task}}$ differs by task, the appropriate range for λ may vary.

4 EXPERIMENTS

We compare the performance of SA-RNN to four state-of-the-art alternatives and two baselines. Since RNNs are the primary solution to a wide variety of problem settings, we use three separate sequential learning settings, each of which tests a different capacity of RNNs while reducing the number of updates to their hidden states using each compared method. First, we use three common publicly-available time series classification tasks. Here, an RNN must summarize the relevant information in an entire sequence with respect to a class label prediction. Second, we use a common and complex language modeling task, requiring an RNN to predict future words in a long sentence, one at a time. Third, we use the classic Adding task to evaluate long-term dependency modeling in gated RNNs. Across all tasks, we demonstrate that SA-RNN consistently achieves superior task-specific performance while updating its hidden states fewer times than the state-of-the-art alternatives.

4.1 Alternative Algorithms

We compare SA-RNN to two baselines and four recent state-of-the-art related methods for computing update-patterns for RNNs:

- **GRU** [8]: The Gated Recurrent Unit (GRU) is a widely-used gated RNN architecture and is described in detail in Section 3.1. The entire hidden state is updated at every timestep.
- **Random Updates**: This method is effectively the random version of our proposal. At each step, random hidden dimensions are updated. This is similar to *Zoneout* regularization [21] however we maintain random updating during testing.
- **Clockwork RNN** [20]: Hidden dimensions are updated in groups at pre-determined "clock" rates. For example, the first five dimensions in the hidden state may update every step while the second five neurons update every 3 steps. This pattern remains fixed during training and testing.
- **Phased LSTM** [27]: Hidden dimensions are updated independently at sampled rates where the user defines the distribution from which to sample. Each hidden dimension has its own update pattern. During testing, the update periods remain unchanged. The dynamics of the updates are user-designed.
- **VC-RNN** [17]: A value $p \in [0, 1]$ is predicted at each step indicating the proportion of the hidden state to update. Then, the first $p * D$ dimensions are updated, imposing a hierarchical structure to the update patterns. While optimizing this method, a "target" update proportion is included in the loss function, penalizing deviations from this target value. This method only uses the first $p * D$ neurons in the hidden state at each step.
- **SkipRNN** [5]: A value $p \in \{0, 1\}$ is predicted at each step. Hidden dimensions update together in lock-step according to p so the entire state is either modified or left unchanged at each step. Additionally, update likelihood increases monotonically after each update, regardless of the input data.

In our experiments all methods are implemented using GRU update equations except for the PhasedLSTM, which is designed specifically for LSTM cells. VC-RNN and PhasedLSTM both employ *soft*

Method	$\sqrt{\text{Acc.} \times \text{Skip \%}}$ (↑)	Skip % (↑)	Accuracy (%) (↑)	FLOPs ($\times 10^5$) (↓)
GRU [8]	0	0	84.7 (0.8)	25.7
CW-RNN [20]	73.7	81	67.1 (0.4)	1.6
PhasedLSTM [27]	74.9	69	81.4 (1.4)	10.6
VC-RNN [17]	68.7	66	71.6 (3.4)	8.9
SkipRNN [5]	69.2	82	58.4 (5.5)	4.8
Random Updates	62.3	76	51.1 (0.1)	6.2
SA-RNN	**78.8**	76	81.6 (0.6)	6.4

Table 1: Performance on the `Seizures` task. $\sqrt{\text{Accuracy} \times \text{Skip\%}}$ is the geometric mean of Accuracy and Skip Percent. The first section contains the baseline GRU, the second contains non-reactive methods, the third contains learned methods, and the fourth is our proposed method and random baseline. For SA-RNN we use $\lambda = 2e^{-4}$.

Method	$\sqrt{\text{Acc.} \times \text{Skip \%}}$ (↑)	Skip % (↑)	Accuracy (%) (↑)	FLOPs ($\times 10^4$) (↓)
GRU [8]	0	0	82.8 (2.2)	56.9
CW-RNN [20]	72.1	81	64.2 (0.8)	3.65
PhasedLSTM [27]	70.2	70	70.5 (1.8)	22.8
VC-RNN [17]	61.9	66	58.1 (2.2)	19.7
SkipRNN [5]	72.2	87	60.0 (1.1)	7.5
Random Updates	61.2	76	49.4 (0.7)	13.1
SA-RNN	**76.1**	77	75.4 (3.0)	25.2

Table 2: Performance on the `TwitterBuzz` task. For SA-RNN we use $\lambda = 2e^{-3}$.

updates to the dimensions of the hidden state, simply altering the scale of the contribution of previous states to current states, similar to the update gates in LSTMs and GRUs. To compute their numbers of updates, we discretize their final masks via a ceiling function since a reduced-size update is still an update.

4.2 Implementation Details

For the Time Series Classification and Adding task experiments, we use 80% of the dataset for training, 10% for validation, and 10% for testing. However, for the PTB dataset, there are given test sets. The training data are used to tune the parameters of the models; the validation data are used to validate hyperparameter selection; while the testing data are finally used to report final performances. We randomly repeat this process ten times to compute confidence intervals for all performance metrics.

For λ selection in SkipRNN and our proposed method, SA-RNN, we search in a log-space ranging from 0.0 to 0.1 in 11 steps. On the `Adding`, `Seizures`, `Yahoo`, and `Twitter` tasks, we found that very-small λ values covered the entire *Skip %* space, from 0 to 100, for SA-RNN. For PTB, however, larger λ values were required, increasing up to ~ 5.0 due to the differences in the scale of $\mathcal{L}_{\text{task}}$.

For all methods on all tasks, we search for learning rates from a log-space ranging from $1e^{-05}$ to $1e^{-01}$ using the validation datasets. All models are optimized using Adam [19]. While we describe our method in the case of the GRU network architecture, it can also be directly applied to other gating mechanisms such as the LSTM. The code for our method is implemented in PyTorch [28] and is available at `https://github.com/thartvigsen/sarnn`.

4.3 Experiments with Time Series

First, we show each model's performance on three classification tasks using the following common publicly-available datasets.

`Seizures` [1]: From 11,800 178-timestep time series, the task is to detect which EEGs contain evidence of epileptic seizure activity. Since there are only 2,300 cases of such activity, we down-sample an equal number from the negative class, resulting in a balanced dataset with 4,600 time series. Finally, we center the time series around zero and compute the mean value of every 10-timestep chunk, summarizing each series into 17 final timesteps.

`TwitterBuzz`[1] [18]: To predict buzz events on Twitter, we work with 77-dimensional time series with labels indicating whether or not a spike in tweets on a particular topic is observed. Starting with over 140,000 timesteps, we compute the mean of every five steps, center the time series around zero, and break the resulting 28,000 timesteps into 2,800 length-15 sequences. We then extract the 776 time series containing any buzz events and balance the dataset by randomly selecting an equal number of no-buzz time series, resulting in 1552 labeled time series.

`Yahoo`[2]: We re-frame this outlier detection dataset as a classification problem: whether or not a sequence contains an anomaly. We first chunk the time series into subsequences of length 25. Then, we create a balanced dataset by selecting all subsequences with anomalies present along with an equal number of randomly-selected subsequences with no anomalies to serve as our negative class. Thus we end up with a dataset containing 418 length-25 time series.

[1] http://ama.liglab.fr/resourcestools/datasets/buzz-prediction-in-social-media/
[2] https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70

Method	$\sqrt{\text{Acc.} \times \text{Skip \%}}$ (↑)	Skip % (↑)	Accuracy (%) (↑)	FLOPs ($\times 10^4$) (↓)
GRU [8]	0	0	64.8 (1.6)	37.9
CW-RNN [20]	68.2	81	57.4 (1.9)	2.4
PhasedLSTM [27]	65.3	70	60.9 (4.0)	15.2
VC-RNN [17]	62.9	66	59.9 (2.8)	13.1
SkipRNN [5]	69.2	88	54.4 (8.1)	4.8
Random Updates	72.2	89	58.6 (1.9)	4.2
SA-RNN	**75.0**	89	63.1 (1.8)	4.4

Table 3: Performance on the `Yahoo` task. For SA-RNN we use $\lambda = 1e^{-3}$.

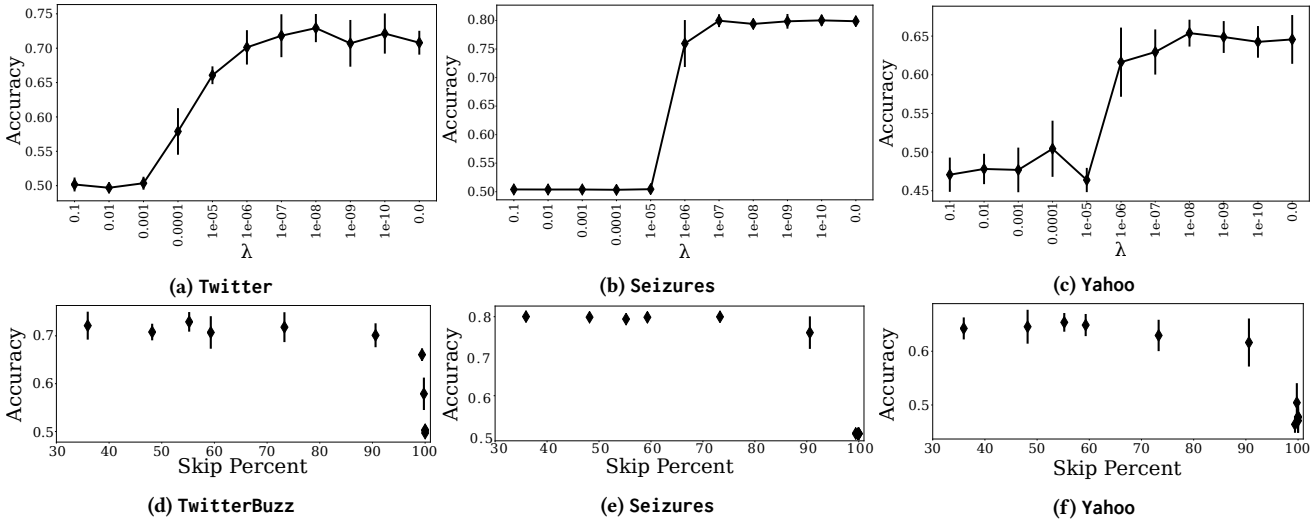


Figure 4: The effect of skip percent and λ on accuracy. Accuracy and skip percent contrast one another, resulting in a trade-off.

4.3.1 Contrasting update patterns of alternative algorithms. We begin with a qualitative visual comparison of the update-patterns generated by each baseline algorithm on the `Seizures` task, as shown in Figure 3 in which black cells indicate updates. Both PhasedLSTM and VC-RNN use *partial* updates, and do not make fully-binary update decisions. For this visualization we binarize their update patterns using a ceiling function on non-zero update decisions since the neurons are still updated. Importantly, all algorithms are subject to update-budgets, which clearly affect the observed patterns. For example, since the SkipRNN updates all neurons at the same time, deciding to update has a large cost, eating through the update budget quickly. Meanwhile, other methods such as our own SA-RNN and the PhasedLSTM take much smaller bites into the update budget since the neurons are updated independently. From these visuals, it is clear that ours is the only method which can adaptively learn to update independent neurons for many steps at a time, according to the data. Thus, some neurons may update a few times in the middle of the sequence instead of at the beginning or end, a decision which is driven directly by the data. This allows our method to spend its computational budget in a more informed way, choosing to update some neurons many times while leaving others unchanged. This process is learned entirely with respect to the corresponding task and so naturally different patterns emerge.

4.3.2 Accuracy vs. Update-frequency. To evaluate the degree to which neurons are updated across all methods, we introduce and compute the *Skip %*, which is the average proportion of the hidden state that has been skipped across all T steps in the input sequence:

$$\text{Skip \%} = 1 - \left(\frac{1}{T * H} \sum_{t=0}^{T} \sum_{l=0}^{H} u_t^l \right) \tag{10}$$

A higher *Skip %* is preferred to a lower *Skip %*, indicating that the hidden state was updated fewer times. Additionally, we compute FLOPs, which is a standard approach to evaluating complexity in conditionally-computed neural networks [5]. This metric measures the number of multiplications required to compute the new states across each testing sequence, including the networks used to output the skip decisions. We use this metric as a surrogate for wall-clock time, which is hardware-dependent and may fluctuate dramatically in practice. To assess performance on these binary classification tasks, in addition to *Skip %* and FLOPs, we compute the *Accuracy*, or the proportion of correct label assignments, of each model on a randomly-selected 10% hold-out testing set. Finally, Skip % and Accuracy are contradictory objectives and so to balance these two measures we compute the geometric mean of each model's *Accuracy* and its associated *Skip %*: $\sqrt{\text{Accuracy} \times \text{Skip\%}}$ since *Accuracy* and

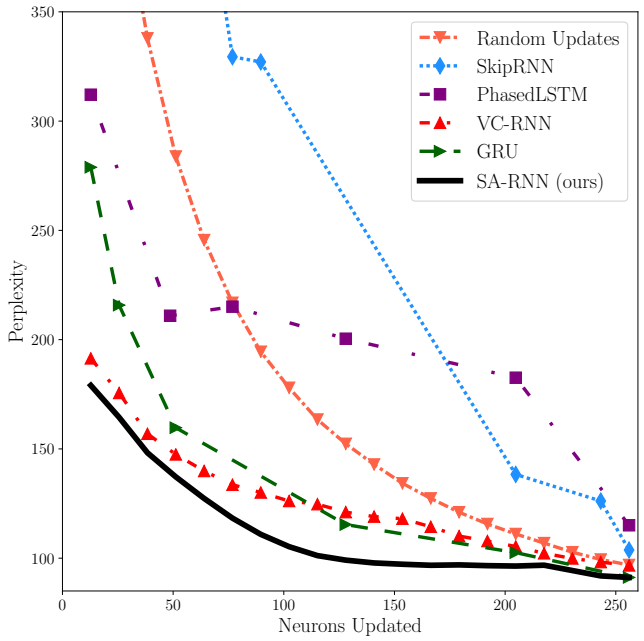


Figure 5: Results from the PTB Language Modeling task. The GRU baseline consists of a set of models, each trained with a hidden state of different dimension, equivalent to the number of neurons updated by the state-of-the-art baselines. Both lower *Perplexity* and lower *Neurons Updated* is preferred and thus the utopian solution is the bottom-left.

Skip % exist in the same range of $[0, 1]$. The geometric mean allows comparison of all methods since they do not all update the exact same number of times. Thus, this is our key performance metric on these tasks, the upper and lower limits of which are 1.0 and 0.0, respectively. During training, we tune each method to skip as close to the same number of updates as possible. For all methods, we use 50-dimensional hidden states and pick the learning rate ($1e^{-3}$) that performed best on the validation set. We set $\mathcal{L}_{\text{task}}$ to be standard binary cross entropy to optimize all methods on the training data.

Across all three datasets, SA-RNN achieves on average higher accuracy with fewer updates, as shown in Tables 1, 2, and 3. In `Yahoo` and `TwitterBuzz`, SA-RNN maintains by far the closest accuracy to that of the baseline GRU, which updates every hidden dimension at every timestep. In `Seizures`, both SA-RNN and PhasedLSTM are similar to the GRU, possibly due to periodic dynamics in the data. We also show in Tables 1 and 2, the accuracy of *Random Updates* is far worse than SA-RNN, indicating that the benefits of our proposed update-strategy comes strongly from the *learning*. The benefits of adaptation are not realized by the CW-RNN or PhasedLSTM.

SA-RNN has nearly-equivalent FLOPs to the other methods that learn update patterns since each adds another affine transformation to map data to update decisions. As shown in Table 2, adapting the update decisions according to the input data adds a significant amount of computation with high-dimensional inputs. To improve this, update patterns may be predicted for multiple steps concurrently, depending on the input data. Given the same number of updates, the FLOPs are equivalent between our method, VC-RNN, and SkipRNN at the limit. It may be possible to only observe the previous hidden state [5], however this results in perpetual lag as information fills the hidden state, thus missing on-line adaptations.

CW-RNN consistently has extremely low FLOPs since there is no data-driven decision making in the update patterns, instead hard-coding them beforehand. Additionally, the recurrence function is not a gated memory cell, which may inhibit its trainability due to the vanishing gradient problem. However, with roughly-equivalent Skip %, CW-RNN consistently performs far worse than the other compared methods in terms of Accuracy.

4.3.3 Effect of budgeting neuron updates. Finally, we assess how the performance of SA-RNN depends on its hyperparameter λ, which budgets the number of permitted updates. We investigate λ values from 0 to 0.1 on a log-scale, since empirically all updates stop when $\lambda > 0.1$ for this task. Interestingly, the relationship between accuracy and λ depends heavily on the task, as demonstrated in in the first row of Figure 4. For example, on the `TwitterBuzz` task, we observe a smooth transition from random guessing (when no updates are allowed) to our peak accuracy (no constraint on updates). Meanwhile, on the `Seizures` task, there is a sharp increase from random predictions to near-peak performance. This could be a feature of the parameter search space, but with already-small changes to λ this indicates high-sensitivity. We also investigate how accuracy changes as a function of the number of neurons skipped, as shown in the second row of Figure 4. Interestingly, there are steep elbows where very few updates are needed to observe near-peak accuracy. This bolsters the intuition behind Residual Networks and others: direct copying is often warranted for many dimensions.

4.4 Experiments with Language Modeling

We next evaluate each model's language modeling capacity on the word-level Penn TreeBank dataset (PTB) [25], which has a vocabulary of 10,000 possible words. The dataset consists of 929,589 training words, 73,760 validation words, and 82,430 testing words. Sentences are broken by <eos> symbols and the task is standard language modeling: Given the current word x_t, predict which of the words in the vocabulary will come next. Importantly, this is a different problem setting than the time series classification tasks. This is a sequence-to-sequence modeling problem where at each step, a new word is input *and* a new prediction is output. As is standard for this task, we minimize Cross Entropy during training:

$$\mathcal{L}_{\text{task}} = -\frac{1}{T}\sum_{t=0}^{T} \log\left(\frac{e^{\hat{y}_t[\text{class}_t]}}{\sum_j e^{\hat{y}^t[j]}}\right) \tag{11}$$

where $\hat{y}_t$ is the logit predicted at step t and class_t is the index of the correct word at step t.

This task has been studied heavily using a wide variety of high-complexity models, the state-of-the-art for which are RNNs comprising of over 20 million parameters [11, 26, 33]. To directly test our hypotheses regarding update-patterns in RNNs, we opt for a simpler architecture. In our experiments, we use single-layer GRU models and settle on 256-dimensional hidden states with 256-dimensional embeddings (via the validation dataset), resulting in roughly 5.5 million parameters across all models. Empirically, 512-dimensional hidden states did not improve performance, while using fewer dimensions decayed performance. We follow standard

Method	Solved	Skip %	FLOPs ($\times 10^6$)
GRU	Yes	0.0	49.7
Random Updates	No	25.0	24.9
Random Updates	No	50.0	12.4
Random Updates	No	90.0	4.97
SkipRNN ($\lambda = 0$)	Yes	2.1 (3.3)	44.8
SkipRNN ($\lambda = 1e^{-7}$)	Yes	21.6 (39.7)	39.6
SA-RNN ($\lambda = 0$)	Yes	25.2 (5.1)	37.5
SA-RNN ($\lambda = 1e^{-5}$)	Yes	49.6 (9.6)	25.3
SA-RNN ($\lambda = 1e^{-4}$)	Yes	90.0 (3.9)	15.3

Table 4: Adding task with 500 timesteps. "Solved" indicates whether or not the model successfully solved the task.

language model evaluation practices and measure *Perplexity*, which quantifies how likely are the words $\hat{y}$ predicted by the model:

$$\text{Perplexity} = e^{-\frac{1}{T}\sum_{t=0}^{T} \ln p(\hat{y}_t)} \tag{12}$$

Since minimizing *Perplexity* is equivalent to maximizing the probability of the correct predictions, the lower the *Perplexity*, the better.

In addition to the state-of-the-art alternative algorithms, in this experiment we compare against GRU models with hidden states of equal size to the number of hidden dimensions updated by the conditional computation models (green line in Figure 5). This comparison tests the intuitive option of simply training a smaller densely-updated RNN instead of updating a larger RNN more sparsely. For this experiment, instead of *Skip %*, we show *Neurons Updated*, allowing us to directly compare the alternative algorithms to these GRU models. These two metrics are direct functions of one another and thus measure the same property, though inversely. We train each model under a large variety of settings, described in Section 4.2, to achieve many different numbers of neurons updated.

As shown in Figure 5, the fully-learned update patterns from SA-RNN have a major positive impact on learning in the presence of long sequences on this complicated task. Of all the compared state-of-the-art methods, SA-RNN clearly preserves the low error rate (91.3) of the fully-updating 256-dimensional hidden state while updating fewer and fewer neurons in the hidden state on the test set. Additionally, we note that the *Perplexity* of the predictions made by smaller and smaller GRUs baseline models increases more rapidly than most state-of-the-art alternatives as its hidden dimension decreases in size. This indicates the value inherent to maintaining a large sparsely-updated hidden state instead of small densely-updated states, partially explaining the value associated with sheer size in the hidden state. We also note that the *Perplexity* of *Random Updates* increases rapidly as the number of neurons update decreases, emphasizing the necessity of *learned* update patterns.

Results from the Clockwork RNN are omitted from these experiments as this method consistently failed to solve the problem adequately (Perplexity $\sim$ 1000). This is likely due to the fact that the Clockwork RNN has no gating mechanisms, and thus the extremely long sequences in this dataset may lead to the vanishing gradient problem solved by gating mechanisms. Notably, as expected SkipRNN performs poorly as fewer of its states are updated. Since this task involves predicting one output per input, skipping an entire hidden state corresponds to ignoring an input word. As soon as words begin being skipped, the prediction of future words is severely impacted. On the contrary, our proposed method and the VC-RNN reduce hidden state computation by updating fewer neurons per hidden state instead of skipping entire updates.

We additionally notice that the PhasedLSTM performs particularly poorly in this setting compared to the other methods. This may be due to this method's unreasonable assumption of update patterns for this particular task: periodic information flow in the input data. This would explain the Clockwork RNN's poor performance as well. This negative impact may be compounded by the use of LSTM cells instead of GRU. In principle, both perform roughly equivalently, but this may vary on a setting-by-setting basis.

4.5 Experiments with the Adding Task

We finally evaluate SA-RNN on the Adding task, which is a foundational synthetic setting for testing RNNs. In this experiment, the network must learn to use a binary mask to sum the corresponding values in a corresponding real-valued sequence. This task is commonly used to validate an RNN's memory capacity in the presence of extremely long-term dependencies [5, 16, 27]. The problem is defined as follows: We are given a data sequence X consisting of of T values ($X = [x_0, \ldots, x_T]$) sampled uniformly between 0 and 1 and a binary masking sequence $M = [m_0 \ldots, m_T]$ containing $T-2$ zeros and 2 ones, which are located at indices i and j. The ground truth y is computed as $y = x_i + x_j$, the sum of the random values at indices i and j. Given a new data sequence X and a masking sequence M, the mean squared error (MSE) between y and $f_\theta(x, m)$, the prediction of the target, should be approximately 0.

We set $T = 500$ to stress-test long-term dependencies and use 128-dimensional hidden states, as suggested in [15]. This is a particularly challenging setting due to the large number of timesteps. To adequately solve the problem, an RNN needs to remember the real values at two given timesteps and disregard all others. Intuitively, this implies that the largest amount of computation should occur only where the signals are, and computation outside these locations is wasted. Additionally, as is typical for the Adding task, we use the MSE between y and $f_\theta(x, m)$ for $\mathcal{L}_{\text{task}}$: $(y^{(i)} - f_\theta(x^{(i)}, m^{(i)}))^2$.

As shown in Table 4, SA-RNN consistently solves the adding task with far higher *Skip %* and lower FLOPs than the baseline GRU and the SkipRNN, a data-reactive method. While SkipRNN theoretically can and should find the proper timesteps to observe, it did not solve the problem consistently, only performing well under a couple of choice settings which updated the vast majority of the hidden states. However, we note that the sequence-length $T = 500$ used in this experiment is ten times greater than was originally explored in [5].

5 CONCLUSIONS

In this paper, we study the problem of reducing the number of updates to the state representations learned by Recurrent Neural Networks, allowing for less computation per timestep. We propose an augmentation to general RNN models, called SA-RNN, which is carefully designed to flexibly skip updates to state dimensions while maintaining task-specific performance through the parameterization of a distribution of update-likelihoods, making binary decisions for each dimension of the hidden state at each step. This not only reduces computation in the hidden state, but could serve

as a regularizer [21]. It thus may lend insights into and lead to the improvement of the robustness of hidden states learned by RNNs.

We conduct extensive experiments on five publicly-available datasets: three real-world time series classification tasks, the complex Penn TreeBank language modeling task, and the classic Adding task for gated RNNs. Each of these tests all our solution as well as state-of-the-art alternative methods on different sequence modeling settings, each of which is commonly solved using RNNs. Across all five datasets, we show that SA-RNN consistently achieves stronger performance while using fewer state updates compared to state-of-the-art alternatives. By using three different settings (regression, classification, language modeling), we also show that SA-RNN is generally more generally applicable. Our results ultimately demonstrate that selectively updating hidden states without imposing high-bias decisions on the update-patterns is not only easier to implement and train, but also superior in terms of performance.

6 ACKNOWLEDGEMENTS

This research was supported by the United States Department of Education grant P200A150306 and the National Science Foundation through grants IIS-1718310, IIS-1815866, IIS-1718310, CNS-1852498, and CNS-1560229. We also thank the Data Science Research Group at Worcester Polytechnic Institute for thoughtful feedback.

REFERENCES

[1] Ralph G Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E Elger. 2001. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Physical Review E* 64, 6 (2001), 061907.

[2] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. 2015. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297* (2015).

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5, 2 (1994), 157–166.

[5] Víctor Campos, Brendan Jou, Xavier Giro-i Nieto, Jordi Torres, and Shih-Fu Chang. 2018. Skip RNN: Learning to Skip State Updates in Recurrent Neural Networks. In *International Conference on Learning Representations*.

[6] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. 2018. Recurrent neural networks for multivariate time series with missing values. *Scientific reports* 8, 1 (2018), 6085.

[7] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282* (2017).

[8] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing*.

[9] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2017. Hierarchical multiscale recurrent neural networks. In *International Conference on Learning Representations*.

[10] Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science* 14, 2 (1990), 179–211.

[11] Chengyue Gong, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2018. Frage: Frequency-agnostic word representation. In *Advances in neural information processing systems*. 1334–1345.

[12] Alex Graves. 2013. Generating sequences with recurrent neural networks. In *arXiv preprint arXiv:1308.0850*.

[13] David Ha and Jürgen Schmidhuber. 2018. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*. 2450–2462.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Computer vision and pattern recognition*. 770–778.

[15] Mikael Henaff, Arthur Szlam, and Yann LeCun. 2016. Recurrent orthogonal networks and long-memory tasks. *International Conference on Machine Learning*.

[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[17] Yacine Jernite, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Variable computation in recurrent neural networks. In *International Conference on Learning Representations*.

[18] François Kawala, Ahlame Douzal-Chouakria, Eric Gaussier, and Eustache Dimert. 2013. Prédictions d'activité dans les réseaux sociaux en ligne. In *4ième conférence sur les modèles et l'analyse des réseaux: Approches mathématiques et informatiques*. 16.

[19] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

[20] Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. 2014. A Clockwork RNN. In *International Conference on Machine Learning*. 1863–1871.

[21] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. 2017. Zoneout: Regularizing rnns by randomly preserving hidden activations. In *International Conference on Learning Representations*.

[22] Zhe Li, Peisong Wang, Hanqing Lu, and Jiang Chen. 2019. Reading selectively via Binary Input Gated Recurrent Unit. In *International Joint Conference on Artificial Intelligence*.

[23] Lanlan Liu and Jia Deng. 2018. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[24] Luchen Liu, Jianhao Shen, Ming Zhang, Zichang Wang, and Jian Tang. 2018. Learning the joint representation of heterogeneous temporal events for clinical endpoint prediction. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[25] Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*. Association for Computational Linguistics, 114–119.

[26] Gábor Melis, Tomáš Kočiskỳ, and Phil Blunsom. 2020. Mogrifier LSTM. In *International Conference on Learning Representations*.

[27] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. 2016. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*. 3882–3890.

[28] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *Advances in neural information processing systems*.

[29] Jürgen Schmidhuber. 2012. Self-delimiting neural networks. *arXiv preprint arXiv:1210.0118* (2012).

[30] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[31] Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. 2019. Ordered Neurons: Incorporating Tree Structures into Recurrent Neural Networks. In *International Conference on Learning Representations*.

[32] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. In *Advances in Neural Information Processing Systems*.

[33] Dilin Wang, Chengyue Gong, and Qiang Liu. 2019. Improving Neural Language Modeling via Adversarial Training. In *International Conference on Machine Learning*. 6555–6565.

[34] Yiren Wang and Fei Tian. 2016. Recurrent residual learning for sequence classification. In *Empirical methods in natural language processing*. 938–943.

[35] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[36] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[37] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*. 2048–2057.

[38] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2017. Recurrent highway networks. In *International Conference on Machine Learning*, Vol. 70. 4189–4198.