

Understanding Feature Selection and Feature Memorization in Recurrent Neural Networks

Bokang Zhu¹ Yongyi Mao² Richong Zhang¹ Dingkun Long¹

Abstract

In this paper, we propose a test, called Flagged-1-Bit (F1B) test, to study the **intrinsic capability** of recurrent neural networks in sequence learning. Four different recurrent network models are studied both analytically and experimentally using this test. Our results suggest that in general there exists a **conflict between feature selection and feature memorization** in sequence learning with recurrent neural networks. Such a conflict can be resolved either using a **gating mechanism** as in LSTM or by **increasing the state dimension** as in Vanilla RNN. Gated models resolve this conflict by adaptively adjusting their state-update equations, whereas Vanilla RNN resolves this conflict by assigning different dimensions different tasks. Insights into feature selection and memorization in recurrent networks are given.

1. Introduction

Over the past decade, the revival of neural networks has revolutionized the field of machine learning, in which deep learning(LeCun et al., 2015) now prevails. Despite the stunning power demonstrated by the deep neural networks, fundamental understanding concerning how these networks work remains nearly vacuous. As a consequence, designing deep network models in practice primarily relies on experience, heuristics and intuition. At present, deep neural networks are still regarded as “black boxes”. Understanding and theorizing the working of these networks are seen, by many, as the utmost important topic in the research of deep learning(Shwartz-Ziv & Tishby, 2017).

Among various frameworks in deep learning, recurrent neu-

ral networks (RNNs) (Elman, 1990; Hochreiter & Schmidhuber, 1997; Cho et al., 2014a) have been shown particularly suited for modelling sequences, natural languages, and time-series data(Sundermeyer et al., 2012; Mikolov et al., 2010; Wu & King, 2016). Their success in sequence learning has been demonstrated in a large variety of practical applications (Cho et al., 2014b; Kalchbrenner et al., 2016; Cho et al., 2014b; Cheng & Lapata, 2016; Cao et al., 2015; Nallapati et al., 2016; Wen et al., 2015; Lewis et al., 2017; Geiger et al., 2014; Zeyer et al., 2017). But, in line with the overall development of deep learning, very limited understanding is available to date regarding how RNNs really work. For example, the following questions yet await for answers: **How does an RNN effectively detect and continuously extract the relevant temporal structure as the input sequence is feeding through the network?** **How does the network maintain the extracted feature in its memory so that it survives in the incoming stream of noise or disturbances?** Investigating these questions is the primary interest of this paper.

In this work, we construct a particular test, referred to the “Flagged-1-Bit” (F1B) test to answer these questions. In this test, we feed a random sequence of “bit pairs”, (X_t^I, X_t^F) to a recurrent network. The X_t^F -components of the sequence contain a single flag (+1) at a random time L ; the X_t^I -components of the sequence are independent bits (± 1 ’s), among which only the value at time L carries the desired feature. The objective of the test is to check if an RNN, *irrespective of the training algorithm*, is able to **select** the feature and **memorize** it until the entire bit-pair sequence is fed to the network.

The F1B test has obviously a simplified setting. However, the feature selection and memorization challenges designed in this test are important aspects of learning with RNN. Specifically, detecting the desired temporal structure and maintaining it throughout the sequence duration must both be carried out through effectively using the network’s size-limited memory, or dimension-limited state. Such challenges arguably exist nearly universally.

We study the behaviour of several RNNs in this test. The studied models include Vanilla RNN(Elman, 1990), LSTM(Hochreiter & Schmidhuber, 1997), GRU(Cho et al., 2014a) and PRU(Long et al., 2018). We note that among

*Equal contribution ¹BDBC and SKLSDE, School of Computer Science and Engineering, Beihang University, 37 Xueyuan Road, Beijing 100191, China ²School of Electrical Engineering and Computer Science, University of Ottawa, 800 King Edward Avenue, K1N 6N5 ON, Canada. Correspondence to: Yongyi Mao <yymao@eecs.uottawa.ca>.

these models, except for Vanilla RNN, all other models exploit some “gating mechanism”.

In this paper, we prove that all studied gated RNNs are capable of passing the test at state dimension 1, whereas Vanilla RNN fails the test at state dimension 1 but passes it at state dimension 2. Our theoretical analysis and experimental study reveal that there is in general a conflict between selecting and memorizing features in sequence learning. We show that such a conflict can be resolved by introducing gates in RNNs, **which allows the RNN to dynamically switch between selection and memorization**. Without using gates, e.g., in Vanilla RNN, such a conflict can only be resolved by increasing the state dimension. Specifically, we show that with adequate state space, Vanilla RNN is capable of splitting feature selection and feature memorization, and assigning the two tasks each to a different state dimension.

To the best of our knowledge, this work is the first to formulate the conflict between feature selection and feature memorization in sequence modelling. It is also the first rigorous analysis of RNN’s capability in resolving this conflict. Although the FIB test has a rather simple structure, the microscopic insights we obtain concerning the behaviour of gated and non-gated RNNs can arguably be extrapolated to more complex settings of sequence learning.

In this paper, in addition to presenting the key results, we also make an effort to give some insights in the problem scope. Due to space limitation, detailed proofs, some further discussions and additional results are presented in Supplementary Material, to which we sincerely invite the reader.

2. Recurrent Neural Networks

A generic *recurrent neural network* (RNN) is essentially a dynamic system specified via the following two equations.

$$s_t = F(s_{t-1}, x_t) \quad (1)$$

$$y_t = G(s_t, x_t) \quad (2)$$

where t denotes discrete time, x_t , y_t and s_t are respectively the *input*, *output* and *state* variables. Note that here all these variables can be in general vectors.

In machine learning, the use of RNN is primarily for learning temporal features in the input sequence x_1, x_2, \dots, x_n . The extraction of temporal information is expected to be carried out by the *state-update equation*, or, (1), where the state s_t at time t serves as a complete summary of the “useful” information contained in the past inputs up to time t . The network uses (2), or the *output equation*, to generate output if needed.

In this paper, since we are only concerned with how temporal features are extracted and maintained in the state variable, we largely ignore the exact form of the output equation. That

is, we regard each recurrent network as being completely defined by its *state-update function* F in (1). We use Θ to denote the parameter of function F .

In this setting an *RNN model* \mathcal{F} is a family of state-update functions having the same parameterization. We now give a concise description of several RNN models, all of which are studied in this work.

Vanilla RNN, denoted by \mathcal{F}_{RNN} , is among the earliest neural network models. It is conventionally known as “recurrent neural network” or simply “RNN” (Elman, 1990). In the modern literature, it is referred to as Vanilla RNN. Its state-update equation is given by (3).

$$s_t = \tanh(Wx_t + Us_{t-1} + b) \quad (3)$$

LSTM, denoted by $\mathcal{F}_{\text{LSTM}}$, was first presented in (Hochreiter & Schmidhuber, 1997) and (Gers & Schmidhuber, 2000). In this paper, we adopt the formalism of LSTM in (Gers & Schmidhuber, 2000). Specifically, to better control the gradient signal in back propagation, LSTM introduces several “gating” mechanisms to control its state-update equation, via (4) to (9).

$$i_t = \sigma(W^i x_t + U^i d_{t-1} + V^i c_{t-1} + b^i) \quad (4)$$

$$f_t = \sigma(W^f x_t + U^f d_{t-1} + V^f c_{t-1} + b^f) \quad (5)$$

$$o_t = \sigma(W^o x_t + U^o d_{t-1} + V^o c_{t-1} + b^o) \quad (6)$$

$$\tilde{c}_t = \tanh(W^{\tilde{c}} x_t + U^{\tilde{c}} d_{t-1} + V^{\tilde{c}} c_{t-1} + b^{\tilde{c}}) \quad (7)$$

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1} \quad (8)$$

$$d_t = o_t \odot \tanh(c_t) \quad (9)$$

Note that we use $\sigma(\cdot)$ to denote the sigmoid (i.e. logistic) function and \odot to denote element-wise vector product. In the context of neural networks, gating usually refers to using a dynamically computed variable in $(0, 1)$ to scale a signal, as in (8) and (9).

In LSTM, the pair (c_t, d_t) of variables together serve as the state variable s_t , if one is to respect the definition of state variable in system theory¹. However, in the literature of LSTM, it is customary to consider c_t as the state variable, since in the design of LSTM (Hochreiter & Schmidhuber, 1997), c_t is meant to serve as a memory, maintaining the extracted feature and carrying it over to the next time instant. We will also take this latter convention in this paper.

GRU, denoted by \mathcal{F}_{GRU} , was presented in (Cho et al., 2014a). It uses a simpler gating structure compared to

¹In system theory, a state variable s_t is a variable (or a set of variable), given which the future behaviour of the system is independent of its past behaviour.

LSTM. Its state-update equation is given by (10) to (13).

$$r_t = \sigma(W^r x_t + U^r s_{t-1} + b^r) \quad (10)$$

$$z_t = \sigma(W^z x_t + U^z s_{t-1} + b^z) \quad (11)$$

$$\tilde{s}_t = \tanh(W^{\tilde{s}} x_t + U^{\tilde{s}}(r_t \odot s_{t-1}) + b^{\tilde{s}}) \quad (12)$$

$$s_t = z_t \odot \tilde{s}_t + (1 - z_t) \odot s_{t-1} \quad (13)$$

Empirical results have shown that with this simplified gating structure, GRU can still perform comparatively or even better than LSTM (Chung et al., 2014).

PRU, or Prototypical Recurrent Unit, which we denote by \mathcal{F}_{PRU} , is a recently presented model (Long et al., 2018). The state-update equation of PRU contains only a single gate as is given by (14) to (16).

$$z_t = \sigma(W^z x_t + U^z s_{t-1} + b^z) \quad (14)$$

$$\tilde{s}_t = \tanh(W^{\tilde{s}} x_t + U^{\tilde{s}} s_{t-1} + b^{\tilde{s}}) \quad (15)$$

$$s_t = z_t \odot \tilde{s}_t + (1 - z_t) \odot s_{t-1} \quad (16)$$

It was shown in (Long et al., 2018) that even with this minimum gate structure, PRU still performs very well. With the same total number of parameters, PRU was shown to outperform GRU and LSTM.

In summary, LSTM, GRU, and PRU are all “gated models”. In particular, their state-update equations are all in a “gated combination” form, as in (8), (13), and (16). Vanilla RNN is not gated.

2.1. Related Works

Despite their numerous applications, fundamental understanding of the working of RNN remains fairly limited.

In (Siegelmann & Sontag, 1992), Vanilla RNN is shown to be Turing-complete. In (Bengio et al., 1994; Pascanu et al., 2013), the difficulty of gradient-based training of Vanilla RNN is demonstrated and the gradient vanishing/-exploding problem is explained. Significant research effort has been spent on curing this gradient problem, including, e.g., (Arjovsky et al., 2016; Dorobantu et al., 2016; Wisdom et al., 2016). In (Collins et al., 2016), trainability and information-theoretic capacities of RNNs are evaluated empirically. Recently (Chen et al., 2018) proposes a theory, based on the notion of dynamical isometry and statistical mechanics, to model signal propagation in RNN and to explain the effectiveness of gating.

3. Flagged-1-Bit Test

Definition 1 (Flagged-1-Bit (F1B) Process) A Flagged-1-Bit Process $F1B(n)$ of length n is a random sequence X_1, X_2, \dots, X_n , defined as follows. Each $X_t := \begin{bmatrix} X_t^I \\ X_t^F \end{bmatrix}$

Table 1. Two sample paths of $F1B(9)$. Top: positive, with $L = 3$. Bottom: negative, with $L = 5$.

t	1	2	3	4	5	6	7	8	9
x_t^I	+1	+1	+1	-1	-1	+1	-1	+1	-1
x_t^F	-1	-1	+1	-1	-1	-1	-1	-1	-1

t	1	2	3	4	5	6	7	8	9
x_t^I	+1	-1	+1	-1	-1	+1	-1	+1	-1
x_t^F	-1	-1	-1	-1	+1	-1	-1	-1	-1

is a length-2 vector, where both X_t^I and X_t^F take values in $\{\pm 1\}$. The two values X_t^I and X_t^F are respectively referred to as an “information bit” and a “flag bit”. To generate the information bits, for each $t = 1, 2, \dots, n$, X_t^I is drawn independently from $\{\pm 1\}$ with equal probabilities. To generate the flag bits, first a random variable L is drawn uniformly at random from the set of integers $\{1, 2, \dots, n\}$, then X_L^F is set to 1, and X_t^F is set to -1 for every $t \neq L$.

In this process, we call X_L^I as the *flagged information bit*. A sample path of $F1B(n)$ is called a *positive sample path* if $X_L^I = +1$, and a *negative sample path* if $X_L^I = -1$. Table 1 shows two example sample paths of an $F1B(n)$ process.

Now consider the learning problem with the objective of classifying the sample paths of $F1B(n)$ into the positive and negative classes. We wish to decide whether it is possible for an RNN model, at a given state dimension K , to learn a perfect classifier. We hope this exercise gives some insights as to how RNN detects and remembers the desired features.

To formally introduce the problem, first recall the definition of a binary linear classifier, where we will use \mathbb{R}^K to denote the Euclidean space of dimension K over the field \mathbb{R} of real numbers: A binary *linear* classifier C on \mathbb{R}^K is a function mapping \mathbb{R}^K to $\{\pm 1\}$ defined by

$$C(s) = \begin{cases} +1, & \text{if } \beta^T s + \gamma \geq 0. \\ -1, & \text{otherwise} \end{cases}$$

where $\beta \in \mathbb{R}^K$ and $\gamma \in \mathbb{R}$ are parameters. We denote (β, γ) collectively by Φ .

For an arbitrary RNN with state-update function F (and a fixed parameter setting), suppose that we draw a sample path $X = (X_1, X_2, \dots, X_n)$ from $F1B(n)$ and feed it to the network F . Suppose that C_Φ is a binary linear classifier, parameterized by Φ , on the state space of the network. When C_Φ is applied to the state variable S_n , its error probability $P_e(F, \Phi)$ is $P_e(F, \Phi) := \Pr[C_\Phi(S_n) \neq X_L^I]$, where the probability measure $\Pr[\cdot]$ is induced by the $F1B(n)$ process through the functions F and C_Φ .

Definition 2 (F1B Test) Suppose that \mathcal{F} is an RNN model with state dimension K and parameterized by Θ . The model \mathcal{F} is said to pass the F1B test if for any integer $n > 0$, there exist an initial configuration s_0 of S_0 , a parameter setting

Θ^* of Θ , which defines $F_{\Theta^*} \in \mathcal{F}$, and a setting Φ^* of Φ such that $P_e(F_{\Theta^*}, \Phi^*) = 0$.

In essence, passing the FIB test means that for any given n , it is possible to find a parameter setting (depending on n) for the RNN model such that the positive and negative sample paths of the $F1B(n)$ process are perfectly and linearly separable at the final state of the RNN. This paper studies whether each afore-mentioned RNN model is capable of passing the test and at which state dimension.

Although the proposed FIB test may seem a toy problem, it in fact captures fundamental challenges in sequence learning. Specifically, for a model to pass this test, it needs to detect and select the flagged information bit when it enters the system, and load it (in some representation) in its state variable. Additionally, it must maintain this information, or “memorize” it, by preventing it from being over-written or suppressed by the subsequent noise sequence. The test thus challenges the model in both its ability to select features and its ability to memorizing them.

For more discussion on the rationale of the FIB test, the reader is referred to Supplementary Material (Section 1).

4. Theoretical Analysis

Theorem 1 *For any of the four models $\mathcal{F}_{\text{LSTM}}$, \mathcal{F}_{GRU} , \mathcal{F}_{PRU} , \mathcal{F}_{RNN} , if it passes the FIB test at state dimension K , it also passes the test at state dimension $K + 1$.*

This result is intuitive, since increasing the state dimension should only increase the modelling power. For each of the four models, the proof of this result relies on constructing a parameter $\Theta^{(K+1)}$ for the model with state dimension $K + 1$ from a parameter $\Theta^{(K)}$, where $\Theta^{(K)}$ is a parameter that allows the model to pass the test at dimension K . The construction of $\Theta^{(K+1)}$ essentially makes the extra dimension in the model useless (i.e., equal to 0), thereby reducing the model to one with state dimension K and parameter $\Theta^{(K)}$.

Due to this theorem, we only need to determine the smallest state dimension K at which a model passes the test.

4.1. Vanilla RNN

Theorem 2 *At state dimension $K = 1$, \mathcal{F}_{RNN} fails the FIB test.*

Theorem 3 *At state dimension $K = 2$, \mathcal{F}_{RNN} passes the FIB test.*

We note that in the proofs of these two theorems, we replace the $\tanh(\cdot)$ activation function in (3) with a piece-wise

linear function $g(\cdot)$, defined by

$$g(x) := \begin{cases} -1, & \text{if } x < -1 \\ x, & \text{if } -1 \leq x \leq 1 \\ 1, & \text{if } x > 1 \end{cases} \quad (17)$$

It is well-known that replacing \tanh with function g only changes the smoothness of the model, without altering its expressivity (Siegelmann & Sontag, 1992). We now give some intuitive explanation as to why these theorems hold.

Intuition of Theorem 2 Consider the following parameter setting of \mathcal{F}_{RNN} for example: $W = [W_1, W_2]$, $b = 0$. In this setting, when feeding the $F1B(n)$ process to Vanilla RNN, the state S_t evolves according to

$$S_t = \tanh(U S_{t-1} + W_1 X_t^I + W_2 X_t^F) \quad (18)$$

To intuitively analyze what is required for the model to pass the test, we divide our discussion into the following phases.

Noise Suppression ($t < L$): In this phase, the desired flagged information bit has not arrived and all incoming information is noise. Then the network should suppress its previous state value. This would require a small value of U .

Feature Loading ($t = L$): This is the time when the flagged information bit arrives. The network then needs to pick up the information and load it into its state. At this point, the network demands W_1 to be reasonably large so that the flagged bit gets amplified sufficiently when it is blended in state S_t . Only so can the selected feature have sufficient contrast against the background noise in the state.

Feature Memorization ($t > L$): In this phase, the selected feature already resides in the state. The network must then protect the selected feature against the incoming noise. To achieve this, it desires W_1 to be small, so that the incoming noise is sufficiently attenuated. At the same time, it desires U to be large, so as to maintain a strong amplitude of the selected feature.

At this end, we see a conflict in the requirements of U and W_1 . Such a conflict is arguably fundamental since it reflects the incompatibility between feature selection and memorization. Not only arising in the FIB test, such a conflict is expected to exist in much broader context of sequence learning using RNNs.

Intuition of Theorem 3 At state dimension $K = 2$, we express S_t as $[S_t^1, S_t^2]^T$. To make Vanilla RNN pass the test, the key insight is that the two states S_t^1 and S_t^2 need to assume separate functions, specifically, one responsible for memorizing the feature (“memorization state”) and the other responsible for selecting the feature (“loading state”). We now illustrate a construction of the Vanilla RNN parameters in which S_t^1 is chosen as the memorization state and S_t^2 as the loading state.

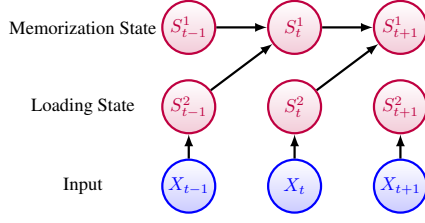


Figure 1. Dependency of variables in Vanilla RNN ($K = 2$) with constructed parameters

For simplicity, we consider the cases where the flag variable X_t^F takes values in $\{0, 1\}$, instead of in $\{\pm 1\}$. We note that this modification does not change the nature of the problem (see Lemma 2 in Supplementary Materials). We also consider the case where the activation function of state update is the piece-wise linear function g .

Now consider the following construction.

$$U := \begin{bmatrix} 1 & b_1 \\ 0 & 0 \end{bmatrix}, W := \begin{bmatrix} 0 & 0 \\ W_{21} & -b_2 \end{bmatrix}$$

The state-update equation then becomes

$$\begin{aligned} S_t^1 &= g(S_{t-1}^1 + b_1 S_{t-1}^2 + b_1) \\ S_t^2 &= g(W_{21} X_t^I - b_2 X_t^F + b_2) \end{aligned}$$

This induces a dependency structure of the state and input variables as shown in Figure 1. For convenience, the pre-activation term in the S_t^1 equation will be denoted by $h(S_{t-1}^1, S_{t-1}^2)$. That is,

$$h(S_{t-1}^1, S_{t-1}^2) := S_{t-1}^1 + b_1 S_{t-1}^2 + b_1.$$

We also impose the following conditions on the parameters: $W_{21} \in (0, 1)$, $b_2 < 0$, $W_{21} X_t^I + b_2 < -1$, $b_1 \neq 0$, and $b_1 \pm b_1 W_{21} \in (-1, 1)$.

It is possible to verify that under such a setting, four distinct “working mechanisms” exist. These mechanisms, their respective conditions and effects are shown in Table 2.

Let the state be initialized as $S_0^1 = 0$ and $S_0^2 = -1$. The dynamics of Vanilla RNN may then be divided into four phases, each controlled by exactly two mechanisms.

Noise Suppression ($t < L$): In this phase, the “load emptying” and “memorization” mechanisms are invoked, where the former keeps the loading state S_t^2 clean, with value -1 , and the latter assures that the memorization state S_t^1 keeps its initial value 0.

Feature Loading ($t = L$): In this phase, “feature loading” is invoked, the feature X_L^I is then loaded into the loading state S_L^2 . The “memorization” mechanism remains in effect, making the memorization state S_L^1 stay at 0.

Feature Transfer ($t = L + 1$): In this phase, “state mixing” is invoked. This makes the memorization state S_{L+1}^1 switch to a linear mixing of the previous loading and memorization states, causing the feature contained in the previous loading state S_L^2 transferred to the memorization state S_{L+1}^1 . At this time, “load emptying” is again activated, resetting the loading state S_{L+1}^2 to -1 .

Feature Memorization ($t > L + 1$): In this phase, “load emptying” remains active, making the loading state S_t^2 stay at -1 . At the same time, “memorization” is again activated, allowing the memorization state S_t^1 to maintain its previous value, thereby keeping the feature until the end.

For tanh-activated Vanilla RNN with $K = 2$, feature selection and memorization appear to work in a similar manner, in the sense that the two states need to execute different tasks and that similar working mechanisms are involved. However, in that case, the four mechanisms are less sharply defined, and in each phase the state update is *dominated*, rather than *completely controlled*, by its two mechanisms.

4.2. Gated RNN Models

Theorem 4 At state dimension $K = 1$, $\mathcal{F}_{\text{LSTM}}$, \mathcal{F}_{GRU} and \mathcal{F}_{PRU} all pass the FIB test.

The theorem is proved by constructing specific parameters for each of the models and for any given n . Under those parameters, it is possible to track the mean μ_t^+ and variance σ_t^{2+} of state s_t for the positive sample paths, and likewise μ_t^- and σ_t^{2-} for the negative sample paths. It can then be shown that the chosen parameters will make $\mu^+ = -\mu^-$ and $\sigma_t^{2+} = \sigma_t^{2-}$, and the ratio $\sqrt{\sigma_n^{2+}}/\mu_n^+$ can be made arbitrarily small. Then if one uses the threshold 0 on S_n to classify the positive and the negative sample paths, the error probability can be made arbitrarily close to 0 using the Chebyshev Inequality (Feller, 2008). Then via a simple lemma (Lemma 3 in Supplementary Materials), arbitrary small error probability in fact implies zero error probability.

Using GRU as an example, we now explain the construction of the model parameters and give some insights into how such a construction allows both feature selection and feature memorization.

In GRU, consider its parameter in the following form:

$$\begin{aligned} W^r &:= [0, 0]; W^z := [0, a]; W^{\tilde{s}} := [b, 0] \\ U^r, U^z, U^{\tilde{s}} &:= 0; b^r, b^z, b^{\tilde{s}} := 0 \end{aligned}$$

Let $A := \sigma(a)$, $B := \tanh(b)$. Then the gated combination equation (13) becomes

$$S_t = AS_{t-1} + (1 - A)BX_t^I, \quad \text{if } t \neq L \quad (19)$$

$$S_t = (1 - A)S_{t-1} + ABX_t^I, \quad \text{if } t = L \quad (20)$$

Table 2. Working mechanisms of Vanilla RNN ($K = 2$) with constructed parameters

Mechanism	Condition	Effect	Comments
Load Emptying	$X_t^F = 0$	$S_t^2 = -1$	Loading state is set to clean background (-1)
Feature Loading	$X_t^F = 1$	$S_t^2 = W_{21}X_t^I$	Feature X_t^I is loaded into loading state
Memorization	$S_{t-1}^1 \in (-1, 1), S_{t-1}^2 = -1$	$S_t^1 = S_{t-1}^1$	Memorization state keeps its previous value
State Mixing	$h(S_{t-1}^1, S_{t-1}^2) \in (-1, 1), S_{t-1}^2 > -1$	$S_t^1 = h(S_{t-1}^1, S_{t-1}^2)$	Linear mixing of loading and memorization states and storing it in memorization state

In above, B is always bundled with X_t^I , serving merely as a global constant that scales the inputs and having no impact on the problem nature. We then take $B = 1$.

Observe that for both $t = L$ and $t \neq L$, the new state is a convex combination of the old state and the current input, however the weights are swapped between the two cases. This contrasts the case of Vanilla RNN (with $K = 1$), in which the same linear combination formula applies to all t .

This phenomenon of swapped weights can be precisely attributed to the gating mechanism in GRU, namely, that the gate variable z_t in (13) is computed dynamically from the previous state and the current input. To analyze the effect of gating, we discuss the consequence of (19) and (20) next.

Noise Suppression ($t < L$): In this phase, (19) applies and the network must suppress the content of the state, since it contains completely noise. This can be achieved by making A either close to 0 or close to 1. When A is near 0, it suppresses the noise by directly attenuating the state. When A is near 1, it suppresses the noise by attenuating the input.

Feature Loading ($t = L$): At this time, (20) applies. In this scenario, the network must amplify the incoming X_L^I and at the same time attenuate the old state. Only so will the desired information bit be superimposed on the noisy state with sufficient contrast. It is easy to see that both objectives can be achieved by choosing A close to 1.

Feature Memorization ($t > L$): In this phase, (19) applies again. At this time, the state contains the selected feature, and the network must sufficiently amplify the state and suppress the incoming noise. These two objectives can be achieved simultaneously by choosing A close to 1.

Summarizing the requirements of A in the three phases, we see that when choosing A close to 1, both feature selection and feature memorization can be achieved. It is in fact precisely due to gating that GRU is able to dynamically control what input to select and how to scale the state.

5. Experiments

5.1. Vanilla RNN: $K = 1$

We have performed extensive computer search in the parameter space of \mathcal{F}_{RNN} , and all checked parameter settings fail

the F1B test. Specifically, we apply each checked parameter setting to the Vanilla RNN model and feed 10,000 randomly generated positive sample paths of length 1000 to the model. This Monte-Carlo simulation then give rise to the final state distribution for the positive sample paths. Similarly, we obtain the final state distribution for the negative sample paths. For each examined parameter, we observe that the two distributions significantly overlap, suggesting a failure of the F1B test.

The results in Figure 2 are obtained by similar Monte-Carlo simulations as above. The figure shows some typical behaviours of Vanilla RNN with state dimension $K = 1$. The first row of the figure corresponds to a case where the value of U is relatively small. In this case, in the noise-suppression phase, the model is able to suppress the noise in the state, and we see the state values move towards the centre. This then allows the success in feature loading, where the positive distribution is separated from the negative distribution. However, such a parameter setting makes the network lack memorization capability. In the feature-memorization phase, we see the two distributions quickly overlap, and can not be separated in the end. That is, the memory of the feature is lost. The second row of the figure corresponds to a case where U is relatively large. In this case, the model is unable to compress the state values in the noise-suppression phase. As a consequence, when the feature is loaded into the state, the positive and the negative distributions can not be separated by a linear classifier. That is, feature selection fails. Then in the feature-memorization phase, the large value of U keeps magnifying the state values in order to keep its memory. As noise is continuously blended in, this effort of ‘‘memorization’’ makes the two distributions even more noisy and in the end completely overlap.

5.2. GRU: $K = 1$

Figure 3 shows some Monte-Carlo simulation results, obtained similarly as those in Figure 2, that validate Theorem 4. The top row of the figure corresponds to a case where A is chosen not close to 1. In this case, we see that in the noise-suppression phase, the state values move away from the centre, suggesting an increased noise power in the state. This noise is however not too strong, which allows the feature to be loaded in the state reasonably well. This

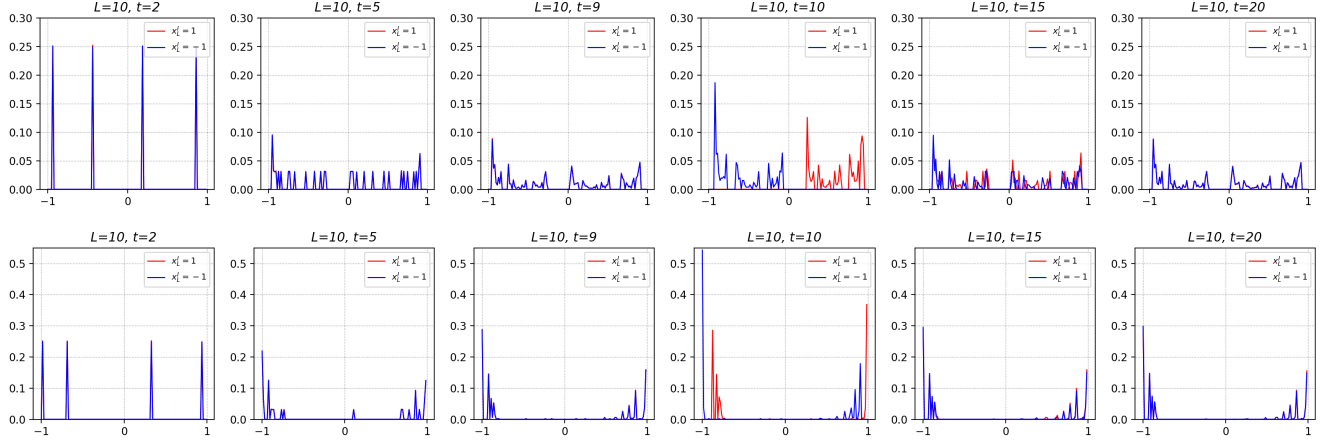


Figure 2. Distribution of state S_t , for various t , in Vanilla RNN with state dimension 1. Input sequence length $n = 20$. Flag location $L = 10$. Red curves: state distributions of positive sample paths. Blue curves: state distributions of negative sample paths. Top row: $U = 0.8$, $W_1 = 0.9$, and $W_2 = 0.1$. Bottom row: $U = 2.0$, $W_1 = 0.7$, $W_2 = 0.1$. The parameters are chosen in a way such that the evolution of the distributions is not too fast, allowing for better visualization.

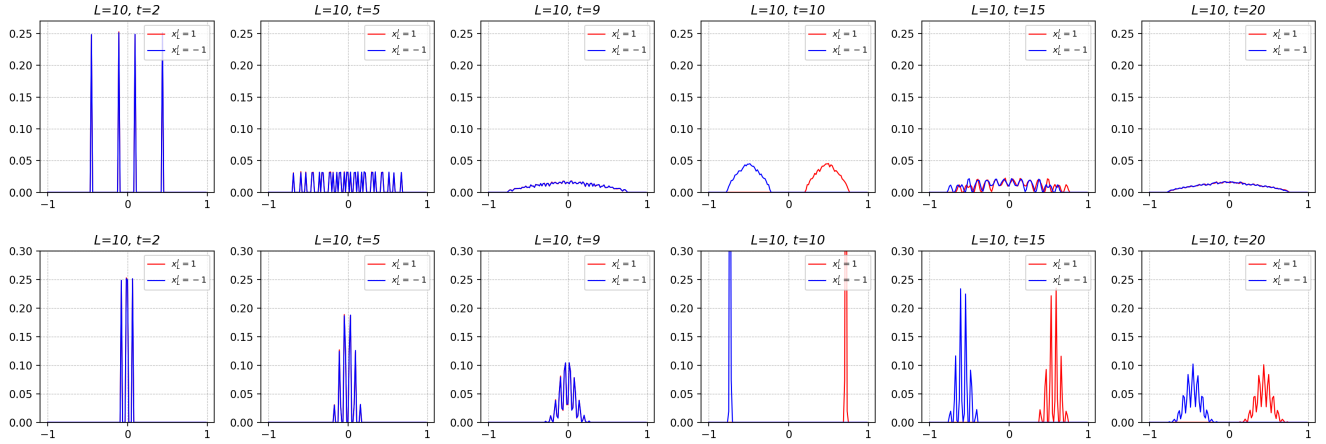


Figure 3. Positive (red) and negative (blue) distribution of state S_t , at various t , for GRU with state dimension 1 (the same set up as that in Figure 2). Top row: $a = 0.5$, $b = 1.0$ ($A = 0.62$, $B = 0.76$). Bottom row: $a = 4.0$, $b = 1.0$ ($A = 0.98$, $B = 0.76$).

is indicated by the well separated positive and negative distributions in the feature-loading phase. But this value of A does not support feature memorization. After the feature is loaded, the two distributions quickly smear into each other and become completely overlapping in the end. The bottom row of the figure corresponds to a case where A is chosen close to 1 (as suggested by the construction). In this case, the state is well suppressed in the noise-suppression phase, allowing for perfect feature loading. The positive and negative distributions are kept well separated in the feature memorization phase, allowing for perfect linear classification. Hence GRU passes the test.

Previous explanations on the effectiveness of gating are mostly based on gradient arguments. For example, in (Hochreiter & Schmidhuber, 1997), introducing gates is for “trapping” the gradient signal in back propagation. Recently the effectiveness of gating is also justified using a

signal-propagation model based on dynamical isometry and mean field theory (Chen et al., 2018). This paper demonstrates the power of gating from a different perspective, namely, that regardless of the training algorithm, gating intrinsically allows the network to dynamically adapt to the input and state configuration. This perspective, to the best of our knowledge, is made precise for the first time.

5.3. Vanilla RNN: $K = 2$

For various sequence lengths n , we train Vanilla RNN with state dimension $K = 2$ where the activation function in the state-update equation is \tanh (some results of Vanilla RNN with $K = 2$ and the piece-wise linear activation function g are given in Supplementary Materials). During training, the final state S_n is sent to a learnable logistic regression classifier. Mini-batched back-propagation is used to train the

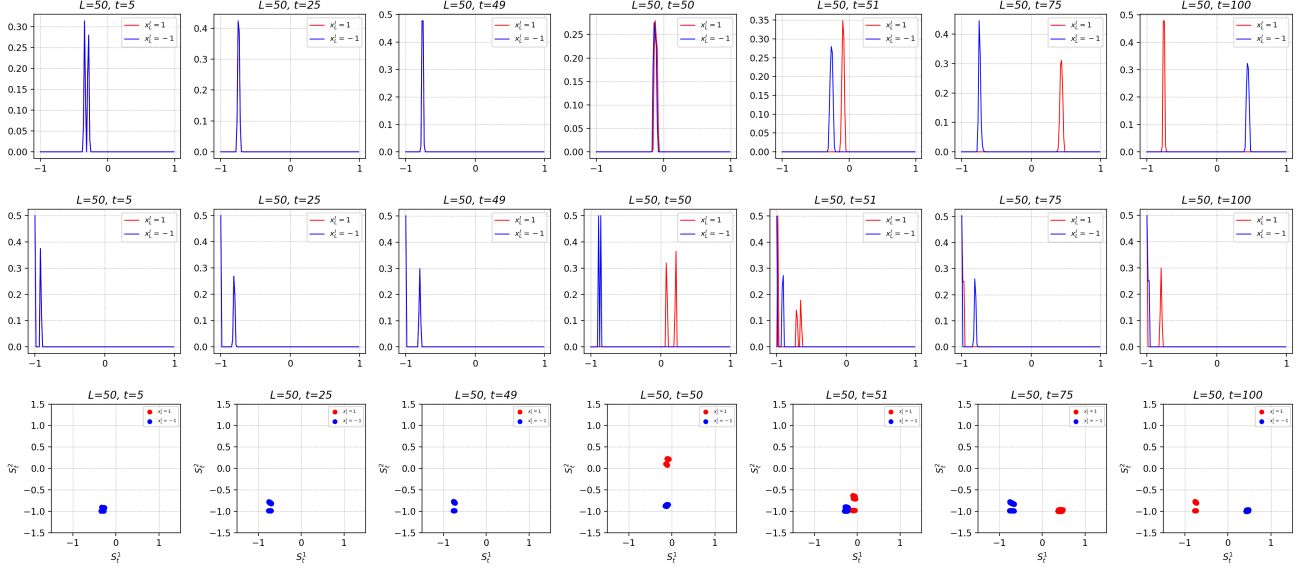


Figure 4. Distributions of state S_t^1 (top row), state S_t^2 (middle row) and the scatter plots of (S_t^1, S_t^2) (bottom row), at various t , in tanh-activated Vanilla RNN with state dimension 2. Sequence length $n = 100$. Flag location $L = 50$. Distributions/scattering patterns for positive (resp., negative) sample paths are shown in red (resp., blue). Results are obtained using 100000 positive sample paths and 100000 negative sample paths. Parameters: $U_{11} = -1.2$, $U_{12} = 0.1938$, $U_{21} = 0.8660$, $U_{22} = 0.6481$, $W_{11} = 0.0073$, $W_{12} = -0.3010$, $W_{21} = 0.7336$, $W_{22} = 1.1052$.

model. With increasing n , the chance of successful training (namely, training loss converging to near zero) decreases significantly, due to the well-known gradient vanishing problem of Vanilla RNN. Figure 4 shows some Monte-Carlo simulation results obtained for a Vanilla RNN parameter setting learned successfully, the shown behaviour is typical in all successfully learned models.

The learned model in Figure 4 is a case where the S_t^1 serves as the memorization state and S_t^2 as the loading state. When $t < L = 50$, S_t^2 remains at a low value, providing a clean background for the loading of the upcoming feature. When $t = L$, the feature X_L^1 is loaded into the loading state S_L^2 , where the positive and negative sample paths exhibit themselves as two separable distributions. When $t = L + 1$, the feature is transferred to the memorization state S_{L+1}^1 , where positive sample paths are separable from the negative ones. Note that at this point, information about the feature is in fact lost in the loading state S_{L+1}^2 and the two distributions on the loading state are no longer separable. But the feature in the memorization state remains memorized for $t > L$, which allows the eventual classification of the positive and negative paths.

6. Concluding Remarks

We propose to use the “Flagged-1-Bit” (F1B) test to RNN’s behaviour in selecting and memorizing temporal features. This test strips off the influence of any training algorithm and reveals the models’ intrinsic capability. Using this test, we articulate that there is in general a conflict between feature selection and feature memorization. Although the

F1B test is a simplified special case, it reveals some insights concerning how RNN deals with this conflict, which might be extrapolated to a broader context of sequence learning.

Specifically with limited state dimension, with or without gate appears to be “the great divide” in the models’ ability and behaviour to resolve such a conflict. Gating allows the network to dynamically adjust its state-update equation, thereby resolving this conflict. Without gates, Vanilla RNN can only exploit the freedom in its state space to accommodate these conflicting objectives. In particular, we point out that Vanilla RNN may resolve this conflict by assigning each state dimension a different function; in the special case of the F1B test and with 2-dimensional state space, one dimension serves to select feature, and the other serves to memorize it. Such an insight is expected to extend to more complex sequence learning problems, although much study is required in those settings. One potential direction is to consider a more general “Flagged- k -Bit” test, in which k randomly spread flags specify the locations of k desired information bits.

This research also implies that if not subject to gradient-based training methods, Vanilla RNN, with adequate state dimensions, is not necessarily inferior to gated models. Thus fixing the gradient problem and developing new optimization techniques for Vanilla RNN remain important.

Finally, consider the spectrum of models from Vanilla RNN, to PRU, to GRU, and to LSTM. At one end of the spectrum, Vanilla RNN uses no gate and relies on increasing state

dimension to resolve the feature selection-memorization conflict. At the other end, LSTM uses a complicated gate structure to resolve the same conflict. This observation makes one wonder whether a better compromise may exploit both worlds, i.e., using a simpler gate structure and slightly increased state dimensions. Indeed, experimental evidences have shown that with simpler gate structures, GRU and PRU may outperform LSTM([Chung et al., 2014](#); [Long et al., 2018](#)).

References

- Arjovsky, M., Shah, A., and Bengio, Y. Unitary evolution recurrent neural networks. In *ICML*, pp. 1120–1128, 2016.
- Bengio, Y., Simard, P. Y., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5 2:157–66, 1994.
- Cao, Z., Wei, F., Dong, L., Li, S., and Zhou, M. Ranking with recursive neural networks and its application to multi-document summarization. In *AAAI*, pp. 2153–2159, 2015.
- Chen, M., Pennington, J., and Schoenholz, S. S. Dynamical isometry and a mean field theory of rnns: Gating enables signal propagation in recurrent neural networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 872–881, 2018.
- Cheng, J. and Lapata, M. Neural summarization by extracting sentences and words. In *ACL*, pp. 484–494, Berlin, Germany, 2016.
- Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. In *EMNLP*, 2014a.
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, pp. 1724–1735, 2014b.
- Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
- Collins, J., Sohl-Dickstein, J., and Sussillo, D. Capacity and trainability in recurrent neural networks. *CoRR*, abs/1611.09913, 2016.
- Dorobantu, V., Stromhaug, P. A., and Renteria, J. Dizzyrnn: Reparameterizing recurrent neural networks for norm-preserving backpropagation. *arXiv preprint arXiv:1612.04035*, 2016.
- Elman, J. L. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.
- Feller, W. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, 2008.
- Geiger, J. T., Zhang, Z., Weninger, F., Schuller, B., and Rigoll, G. Robust speech recognition using long short-term memory recurrent neural networks for hybrid acoustic modelling. In *INTERSPEECH*, 2014.
- Gers, F. A. and Schmidhuber, J. Recurrent nets that time and count. In *IJCNN*, volume 3, pp. 189–194, 2000.
- Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- Kalchbrenner, N., Espeholt, L., Simonyan, K., van den Oord, A., Graves, A., and Kavukcuoglu, K. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.
- LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436, 2015.
- Lewis, M., Yarats, D., Dauphin, Y. N., Parikh, D., and Batra, D. Deal or no deal? end-to-end learning for negotiation dialogues. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2017.
- Long, D., Zhang, R., and Mao, Y. Prototypical recurrent unit. *Neurocomputing*, 311:146–154, 2018. doi: 10.1016/j.neucom.2018.05.048.
- Mikolov, T., Karafiát, M., Burget, L., Černocký, J., and Khudanpur, S. Recurrent neural network based language model. In *INTERSPEECH*, 2010.
- Nallapati, R., Zhou, B., dos Santos, C., Gulcehre, C., and Xiang, B. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *CoNLL*, pp. 280–290, 2016.
- Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. *ICML*, 28:1310–1318, 2013.
- Shwartz-Ziv, R. and Tishby, N. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.
- Siegelmann, H. T. and Sontag, E. D. On the computational power of neural nets. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92*, pp. 440–449, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X. doi: 10.1145/130385.130432.
- Sundermeyer, M., Schlüter, R., and Ney, H. Lstm neural networks for language modeling. In *INTERSPEECH*, 2012.
- Wen, T., Gasic, M., Mrksic, N., Su, P., Vandyke, D., and Young, S. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *EMNLP*, 2015.

Wisdom, S., Powers, T., Hershey, J. R., Roux, J. L., and Atlas, L. E. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 4880–4888, 2016.

Wu, Z. and King, S. Investigating gated recurrent networks for speech synthesis. In *ICASSP*, pp. 5140–5144. IEEE, 2016.

Zeyer, A., Doetsch, P., Voigtlaender, P., Schlüter, R., and Ney, H. A comprehensive study of deep bidirectional lstm rnns for acoustic modeling in speech recognition. In *ICASSP*, pp. 2462–2466. IEEE, 2017.