Learning and Generalization in RNNs

Abhishek Panigrahi
Department of Computer Science
Princeton University
ap34@princeton.edu

Navin Goyal
Microsoft Research India
navingo@microsoft.com

Abstract

Simple recurrent neural networks (RNNs) and their more advanced cousins LSTMs etc. have been very successful in sequence modeling. Their theoretical understanding, however, is lacking and has not kept pace with the progress for feedforward networks, where a reasonably complete understanding in the special case of highly overparameterized one-hidden-layer networks has emerged. In this paper, we make progress towards remedying this situation by proving that RNNs can learn functions of sequences. In contrast to the previous work that could only deal with functions of sequences that are sums of functions of individual tokens in the sequence, we allow general functions. Conceptually and technically, we introduce new ideas which enable us to extract information from the hidden state of the RNN in our proofs—addressing a crucial weakness in previous work. We illustrate our results on some regular language recognition problems.

1 Introduction

Simple Recurrent Neural Networks [1] also known as Elman RNNs or vanilla RNNs (just RNNs henceforth) along with their more advanced versions such as LSTMs [2] and GRU [3] are among the most successful models for processing sequential data, finding wide-ranging applications including natural language processing, audio processing [4] and time series classification [5]. Feedforward networks (FFNs) model functions on inputs of fixed length, such as vectors in $\mathbb{R}^d$. In contrast, RNNs model functions whose input consists of sequences of tokens $x^{(1)}, x^{(2)}, \ldots$, where $x^{(i)} \in \mathbb{R}^d$ for each $i$. RNNs have a notion of memory; formally it is given by the hidden state vector which is denoted by $h^{(t)}$ after processing the $t$-th token. RNNs apply a fixed function to $h^{(t)}$ and $x^{(t+1)}$ to compute $h^{(t+1)}$ and the output. This fixed function is modeled by a neural networks with one hidden-layer. Compared to FFNs, new challenges arise in the analysis of RNNs: for example, the use of memory and the same function at each step introduces dependencies across time and RNN training suffers from vanishing and exploding gradients [6].

Studies aimed at understanding the effectiveness of RNNs have been conducted since their introduction; for some of the early work, see, e.g., [7, 8]. These works take the form of experimental probing of the inner workings of these models as well as theoretical studies. The theoretical studies are often focused on expressibility, training and generalization questions in isolation rather than all together—the latter needs to be addressed to approach full understanding of RNNs and appears to be far more challenging. While experimental probing has continued apace, e.g., [9, 10], progress on theoretical front has been slow. It is only recently that training and generalization are starting to be addressed in the wake of progress on the relatively easier case of FFNs as discussed next.

RNNs are closely related to deterministic finite automata [11] as well as to dynamical systems. With finite precision and ReLU activation, they are equivalent to finite automata [11] in computational power. In the last few years progress was made on theoretical analysis of overparameterized FFNs

*Work done as a research fellow in Microsoft Research India

with one-hidden-layer, e.g., \([12, 13, 14, 15, 16, 17, 18]\). Building upon these techniques, \([19]\) proved that RNNs trained with SGD (stochastic gradient descent) achieve small training loss if the number of neurons is sufficiently large polynomial in the number of training datapoints and the maximum sequence length.

But the gap between our understanding of RNNs and FFNs remains large. \([20, 21]\) provide generalization bounds on RNNs in terms of certain norms of the parameters. While interesting, these bounds shed light on only a part of the picture as they do not consider the training of the networks nor do not preclude the possibility that the norms of the parameters for the trained networks are large leading to poor generalization guarantees. RNNs can be viewed as dynamical systems and many works have used this viewpoint to study RNNs, e.g., \([22, 23, 24, 25]\). Other related work includes relation to kernel methods, e.g., \([26, 27, 28]\), linear RNNs \([29]\), saturated RNNs \([30, 31, 32]\), and echo state networks \([33, 34]\). Several other works talk about the expressive power of the novel sequence to sequence models Transformers \([35, 36]\). Due to a large number of works in this area it is not possible to be exhaustive: apart from the references directly relevant to our work we have only been able to include a small subset.

\([37]\) gave the first “end-to-end” result for RNNs. Very informally, their result is: if the concept class consists of functions that are sums of functions of tokens then overparameterized RNNs trained using SGD with sufficiently small learning rate can learn such a concept class. They introduce new technical ideas, most notably what they call re-randomization which allows one to tackle the dependencies that arise because the same weights are used in RNN across time. However, an important shortcoming of their result is limited expressive power of their concept class: while this class can be surprisingly useful as noted there, it cannot capture problems where the RNN needs to make use of the information in the past tokens when processing a token (in their terminolgy, their concept class can adapt to time but not to tokens). Indeed, a key step in their proof shows that RNNs can learn to ignore the hidden state \(h^{(t)}\). (The above concept class comes up because it can be learnt even if \(h^{(t)}\) is ignored.) But the hidden state \(h^{(t)}\) is the hallmark of RNNs and is the source of information about the past tokens—in general, not something to be ignored. Thus, it is an important question to theoretically analyze RNNs’ performance on general concept classes and it was also raised in \([37]\). This question is addressed in the present paper. As in previous work, we work with sequences of bounded length \(L\).

Without loss of generality, we work with token sequences \(x^{(1)}, \ldots, x^{(L)}\) of fixed length as opposed to sequences of length up to \(L\). Informally, our result is:

Overparameterized RNNs can efficiently learn concept classes consisting of one-hidden-layer neural networks that take the entire sequence of tokens as input. The training algorithm used is SGD with sufficiently small step size.

By the universality theorem for one-hidden-layer networks, such RNNs can approximate all continuous functions of \(x^{(1)}, \ldots, x^{(L)}\)—though naturally the more complex the functions in the class the larger the network size required. We note that the above result applies to all three aspects mentioned above: expressive power, training and generalization. We illustrate the power of our result by showing that some regular languages such as PARITY can be recognized efficiently by RNNs.

In a concurrent work \([38]\), the authors also study related problems. The results there are not directly comparable: On the one hand, they do not normalize the input unlike our work. On the other hand, the concept classes treated in their work appear to be weak. The authors show that RNNs can learn concept classes that include functions of the form \(f(x^{(\ell_1)}, x^{(\ell_2)}, \ldots, x^{(\ell_N)})\), where \(\ell_1, \ldots, \ell_N \in [L]\). However, the number of SGD iterations and training samples necessary depend exponentially on the minimum of the two parameters, \(N\) and \(\ell_0 = \max(\ell_1, \ldots, \ell_N) - \min(\ell_1, \ldots, \ell_N)\).

2 Preliminaries

Let \(\mathbb{S}^{d-1} := \{x \in \mathbb{R}^d \mid \|x\|_2 = 1\}\) be the unit sphere in \(\mathbb{R}^d\). For positive integer \(n\) define \([n] := \{1, 2, \ldots, n\}\). Given a vector \(v\), by \(v_i\) we denote its \(i\)-th component. Given two vectors \(a \in \mathbb{R}^{d_1}\) and \(b \in \mathbb{R}^{d_2}\), \([a, b] \in \mathbb{R}^{d_1+d_2}\) denotes the concatenation of the two vectors. \(\langle \cdot, \cdot \rangle\) denotes the standard dot product. Given a matrix \(M\), we will denote its \(i\)-th row as \(m_i\) and the element in row \(i\) and column \(j\) as \(m_{ij}\). Given two matrices \(A \in \mathbb{R}^{d_1 \times d_2}\) and \(B \in \mathbb{R}^{d_3 \times d_4}\) with \(a_1 = b_1\) let
[A, B]_r ∈ ℝ^{a_1 × (a_2 + b_2)} denote the matrix whose rows are obtained by concatenating the respective rows of A and B. Similarly, [A, B]_e ∈ ℝ^{(a_1 + b_1) × a_2} (assuming a_2 = b_2) denotes the matrix whose columns are obtained by concatenating the columns of A and B.

O(·) and Ω(·) hide absolute constants. Similarly, poly(·) denotes a polynomial in its arguments with degree and coefficients bounded by absolute constants; different instances of poly(·) may refer to different polynomials. Writing out explicit constants would lead to unwieldy formulas without any new insights.

Let σ : ℝ → ℝ, given by σ(x) := max{0, x} = x I_{x≥0}, be ReLU activation function. ReLU can be extended to act on vectors by coordinate-wise application: σ((x_1, ..., x_d)) := (σ(x_1), ..., σ(x_d)). Note that ReLU is a positive homogenous function of degree 1, that is to say σ(λx) = λ σ(x) for all x and all λ ≥ 0.

To be learnable efficiently, the functions in the concept class need to be not too complex. We will quantify this with the following two complexity measures which are weighted norms of the Taylor expansion and intuitively can be thought of as quantifying network size and sample complexities, resp., needed to learn φ up to error ε.

**Definition 2.1** (Function complexity [15]). Suppose that function φ : ℝ → ℝ has Taylor expansion φ(z) = ∑_{i=0}^{∞} c_i z^i. For R, ε > 0, define

\[ \mathcal{C}_c(φ, R) := ∑_{i=0}^{∞} C^* R^i + \left( \frac{\sqrt{\log(1/ε)}}{\sqrt{i}} C^* R^i \right) |c_i|, \]

\[ \mathcal{C}_c(φ, R) := C^* ∑_{i=0}^{∞} (i + 1)^{1.75} R^i |c_i|, \]

where \( C^* = 10^4 \). As an example, if φ(z) = z^d for positive integer d, then \( \mathcal{C}_c(φ, R) = O(R^d) \) and \( \mathcal{C}_c(φ, R) = O(R^d \log^{7/2}(1/ε)) \). For φ(z) = sin z, cos z, e^z, we have \( \mathcal{C}_c(φ, R) = O(1) \) and \( \mathcal{C}_c(φ, R) = \text{poly}(1/ε) \). We have \( \mathcal{C}_c(φ, R) ≤ \mathcal{C}_c(φ, R) ≤ \mathcal{C}_c(φ, R) \times \text{poly}(1/ε) \) for all φ and for φ(z) = sin z, e^z or constant degree polynomials, they only differ by o(1/ε). See [15] for details. Note that φ itself is not a member of our concept class but functions like it will be used to construct members of our concept class.

# 3 Problem Formulation

In our set-up, RNNs output a label after processing the whole input sequence.\footnote{While our set-up has similarity to previous work [37], there are also important differences.} The data are generated from an unknown distribution \( D \) over \( ([\vec{x}^{(2)}, ..., \vec{x}^{(L-1)}], y^*) \) ∈ \( ([S^d]^{L-1}, Y) \), for some label set \( Y \subset \mathbb{R} \) and some positive integer \( d_{\text{out}} \). We call \( \vec{x} := ([\vec{x}^{(2)}, ..., \vec{x}^{(L-1)}]) \) the true sequence and \( y^* \) the true label. Denote by \( Z \) the training dataset containing \( N \) i.i.d. samples from \( D \). We preprocess the true sequence to normalize it:

**Definition 3.1** (Normalized Input sequence). Let \( \vec{x} := ([\vec{x}^{(2)}, ..., \vec{x}^{(L-1)}]) \) be a given true input sequence of length \( L - 2 \), s.t. \( \vec{x}^{(i)} \in S^d \) and \( \vec{x}^{(i)} = \frac{1}{2} \), for all \( i \in [2, L - 1] \). The normalized input sequence \( \vec{x} := ([\vec{x}^{(1)}, ..., \vec{x}^{(L)}]) \) is given by

\[ \vec{x}^{(1)} := (0^{d-1}, 1), \quad \vec{x}^{(\ell)} := (ε_x \vec{x}^{(\ell)}, 0), \quad ∀ \ell ∈ [2, L - 1], \quad \vec{x}^{(L)} := (0^{d-1}, 1), \]

where we set \( ε_x > 0 \) later in Theorem 3.1.

We use normalized sequence in place of the true sequence as input to RNNs, as it helps in proofs, e.g., with bounds on the changes in the activation patterns at each RNN cell, when the input sequences change and also with inversion of RNNs (defined later). Our method can be applied without normalization too, but in that case our error bound has exponential dependence on the length of the input sequence. The extra dimension in the normalized sequence serves as bias which we do not use explicitly to simplify notation.

## 3.1 RNNs

**Definition 3.2** (Recurrent Neural Networks). We assume that the input sequences are of length \( L \) for some given \( L > 0 \) and are of the form \( \vec{x}^{(1)}, ..., \vec{x}^{(L)} \) with \( \vec{x}^{(\ell)} \in \mathbb{R}^d \) for all \( \ell \in [L] \). An RNN is...
specified by three matrices $W_{\text{rnn}} \in \mathbb{R}^{m \times m}, A_{\text{rnn}} \in \mathbb{R}^{m \times d}$ and $B_{\text{rnn}} \in \mathbb{R}^{d_{\text{out}} \times m}$, where $m$ is the dimension of the hidden state and $d_{\text{out}}$ is the dimension of the output. The hidden states of the RNN are given by $h_{\text{rnn}}^{(0)} = 0 \in \mathbb{R}^m$ and

$$h_{\text{rnn}}^{(\ell)} := \sigma(A_{\text{rnn}} x^{(\ell)} + W_{\text{rnn}} h_{\text{rnn}}^{(\ell-1)}) \quad \text{for } \ell \in [L].$$

The output at each step $\ell \in [L]$ is given by $y_{\text{rnn}}^{(\ell)} = B_{\text{rnn}} h_{\text{rnn}}^{(\ell)}$. By RNN cell we mean the underlying FFN in $[1]$. The $m$ rows of $W_{\text{rnn}}$ and $A_{\text{rnn}}$ correspond to the $m$ neurons in the RNN.

Pick the matrices $W \in \mathbb{R}^{m \times m}$ and $A \in \mathbb{R}^{m \times d}$ by sampling entries i.i.d. from $N(0, \frac{2}{m})$, and pick $B$ by sampling entries i.i.d. from $N(0, \frac{2}{d_{\text{out}}})$. When $W_{\text{rnn}} = W$ and $A_{\text{rnn}} = A$, the RNN is said to be at random initialization. We will denote the parameters of an RNN at initialization by dropping the subscript “rnn”, thus the hidden states are $\{h^{(\ell)}\}_{\ell \in [L]}$. In the following theorems, we will keep $B_{\text{rnn}}$ at initialization $B$ and train only $A_{\text{rnn}}$ and $W_{\text{rnn}}$.

We write $F^{(\ell)}_{\text{rnn}}(x; W_{\text{rnn}}, A_{\text{rnn}}) = y_{\text{rnn}}^{(\ell)} = B h_{\text{rnn}}^{(\ell)}$ for the output of the $\ell$-th step. Our goal is to use $y_{\text{rnn}}^{(L)} \in \mathbb{R}^{d_{\text{out}}}$ to fit the true label $y^* \in \mathcal{Y}$ using some loss function $G : \mathbb{R}^{d_{\text{out}}} \times \mathcal{Y} \to \mathbb{R}$. We assume that for every $y^* \in \mathcal{Y}$, $G(0^k, y^*) \in [-1,1]$ is bounded, and $G(\cdot, y^*)$ is convex and 1-Lipschitz continuous in its first variable. This includes, for instance, the cross-entropy loss and $\ell_2$-regression loss (for bounded arguments).

### 3.2 Concept Class

We now define our target concept class, which we will show to be learnable by RNNs using SGD.

**Definition 3.3** (Concept Class). Our concept class consists of functions $F : \mathbb{R}^{(L-2)(d-1)} \to \mathbb{R}^{d_{\text{out}}}$ defined as follows. Let $\Phi$ denote a set of smooth functions with Taylor expansions with finite complexity as in Def. [21]. To define a function $F$, we choose a subset $\{\Phi_{r,s} : \mathbb{R} \to \mathbb{R}\}_{r \in [p], s \in [d_{\text{out}}]}$ from $\Phi$, a set of output coefficients with $|b_{r,s}^i| \leq 1$, and for each output dimension $s \in [d_{\text{out}}]$ we define the $s$-th coordinate $F_s$ of $F = (F_1, \ldots, F_{d_{\text{out}}})$ by

$$F_s(x) := \sum_{r \in [p]} b_{r,s}^i \Phi_{r,s} \left(\langle w_{r,s}^i, [x^{(2)}, \ldots, x^{(L-1)}]\rangle\right).$$

To simplify formulas, we assume $\Phi_{r,s}(0) = 0$ for all $r$ and $s$. We denote the complexity of the concept class by

$$c_c(\Phi, R) := \max_{\phi \in \Phi} \{c_c(\phi, R)\}, \quad c_s(\Phi, R) := \max_{\phi \in \Phi} \{c_s(\phi, R)\}.$$ 

Let $F^*$ be a function in the concept class with smallest possible population loss which we denote by OPT. Hence, we are in an agnostic learning setting where our aim is to learn a function with population objective $\text{OPT} + \varepsilon$. As one can observe, functions in the concept class are given by a one hidden layer network with $p$ neurons and smooth activations. We will show that the complexity of the functions $\Phi_{r,s}$ determines the number of neurons and the number of training samples necessary to train the recurrent neural network that has $\text{OPT} + \varepsilon$ population loss.

While we have defined $F^*$ as a function of $x$, since there’s a one-to-one correspondence between $x$ and $x$, it will occasionally be convenient to talk about $F^*$ as being a function of $x$—and this should cause no confusion. And similarly for other functions like $F^{(\ell)}_{\text{rnn}}(x; W, A)$.

### 3.3 Objective Function and the Learning Algorithm

We assume that there exists a function $F^*$ in the concept class that can achieve a population loss $\text{OPT}$, i.e. $\mathbb{E}_{(x, y^*) \sim \mathcal{D}} G(F^*(x), y^*) \leq \text{OPT}$. The following loss function is used for gradient descent:

$$\text{Obj}(W', A') = \mathbb{E}_{(x, y^*) \sim \mathcal{D}} \text{Obj}(x, y^*; W', A'),$$

where

$$\text{Obj}(x, y^*; W', A') = G(\lambda F^{(\ell)}_{\text{rnn}}(x; W + W', A + A'), y^*).$$
Parameter $\lambda$ whose value is set in the main Theorem 3.1 is a scaling factor needed for technical reasons discussed later. We consider vanilla stochastic gradient updates with $W_t, A_t$ denoting the matrices after $t$-steps of sgd. $W_t$ and $A_t$ are given by
\[
W_t = W_{t-1} - \eta \nabla W_{t-1} \text{Obj}(x, y^*; W_{t-1}, A_{t-1}),
\]
\[
A_t = A_{t-1} - \eta \nabla A_{t-1} \text{Obj}(x, y^*; W_{t-1}, A_{t-1}),
\]
where $(x, y^*)$ is a random sample from $Z$ and $x$ is its normalized form. It should be noted that $[37]$ train only $W$.

**Remark.** We made two assumptions in our set-up: (1) input sequences are of fixed length, and (2) the output is only considered at the last step. These assumptions are without loss of generality and allow us to keep already quite complex formulas manageable without affecting the essential ideas. The main change needed to drop these assumptions is a change in the objective function, which will now include terms not just for how well the output fits the target at step $L$ but also for the earlier steps. The objective function for each step behaves in the same way as that for step $L$, and so the sum can be analyzed similarly. Intuitively speaking, considering the output at the end is the “hardest” training regime for RNNs as it uses the “minimal” amount of label information.

### 3.4 RNNs learn the concept class

We are now ready to state our main theorem. We use $\rho := 100L d_{out} \log m$ in the following. Recall that a set $\Phi$ of smooth functions induces a concept class as in Def. 3.3.

**Theorem 3.1** (Main, restated in the appendix as Theorem D.5). Let $\Phi$ be a set of smooth functions. For $\epsilon := \frac{1}{\rho \Omega(p)}$ and $\epsilon \in \left(0, \frac{1}{p \rho \Omega(p)}\right)$, define complexity $C := \epsilon \rho \Omega(p, C, \epsilon^{-1})$ and $\lambda := \frac{\Omega(p, C, \epsilon^{-1})}{\rho \Omega(p)}$. Assume that the number of neurons $m \geq \text{poly}(C, p, L, d_{out}, \epsilon^{-1})$ and the number of samples $N \geq \text{poly}(C, p, L, d_{out}, \epsilon^{-1})$. Then with parameter choices $\eta := \Theta\left(\frac{1}{\epsilon \rho^2 m}\right)$ and $T := \Theta\left(p^2 C^2 \text{poly}(\rho) \epsilon^{-2}\right)$ with probability at least $1 - e^{-\Omega(\rho^2)}$ over the random initialization, SGD satisfies
\[
\mathbb{E}_{sGD} \left[ \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}_{(x, y^*) \sim D} \text{Obj}(x, y^*; W_t, A_t) \right] \leq \text{OPT} + \epsilon + 1/\text{poly}(\rho). \tag{3}
\]

Informally, the above theorem states that by SGD training of overparameterized RNNs with sufficiently small learning rate and appropriate preprocessing of the input sequence, we can efficiently find an RNN that has population objective nearly as small as $\text{OPT}$ as $\epsilon + 1/\text{poly}(\rho)$ is small. The required number of neurons and the number of training samples have polynomial dependence on the function complexity of the concept class, the length of the input sequence, the output dimension, and the additional prediction error $\epsilon$.

### 4 Proof Sketch

While the full proof is highly technical, in this section we will sketch the proof focusing on the conceptual aspects while minimizing the technical aspects to the essentials; full proofs are in the appendix. The high-level outline of our proof is as follows.

1. **Overparameterization simplifies the neural network behavior.** The function $F_{\text{rnn}}^{(L)}(x; W + W', A + A')$ computed by the RNN is a function of the parameters $W', A'$ as well as of the input $x$. It is a highly non-linear and non-convex function in both the parameters and in the input. The objective function inherits these properties and its direct analysis is difficult. However, it has been realized in the last few years—predominantly for the FFN setting—that when the network is overparameterized (i.e., as the number of neurons $m$ becomes large compared to other parameters of the problem such as the complexity of the concept class), the network behavior simplifies in a certain sense. The general idea carries over to RNNs as well: in (1) below we write the first-order Taylor approximation of $F_{\text{rnn}}^{(L)}(x; W + W', A + A')$ at $W$ and $A$ as a linear function of $W'$ and $A'$; it is still a non-linear function of the input.
sequence. As in [37] we call this function pseudo-network, though our notion is more general as we vary both the parameters $W'$ and $A'$. Pseudo-network is a good approximation of the target network as a function of $\bar{x}$ for all $\bar{x}$.

2. **Existence of a good RNN.** In order to show that the RNN training successfully learns, we first show that there are parameters values for RNN so that as a function of $\bar{x}$ it is a good approximation of $F^*$. Instead of doing this directly, we show that the pseudo-network can approximate $F^*$; this suffices as we know that the RNN and the pseudo-network remain close. This is done by constructing parameters $W^*$ and $A^*$ so that the resulting pseudo-network approximates the target function in the concept class (Section 4.2) for all $\bar{x}$.

3. **Optimization.** SGD makes progress because the loss function is convex in terms of the pseudo-network which stays close to the RNN as a function of $x$. Thus, SGD finds parameters with training loss close to that achieved by $W^*$, $A^*$.

4. **Generalization.** Apply a Rademacher complexity-based argument to show that SGD has low population loss.

Step 2 is the main novel contribution of our paper and we will give more details of this step in the rest of this section.

### 4.1 Pseudo-network

We define the pseudo-network here. Suppose $W$, $A$, $B$ are at random initialization. The linear term in the first-order Taylor approximation is given by the pseudo-network

$$F^{(L)}(x; W', A') := \sum_{i=1}^{L} \text{Back}_{i \rightarrow L} D^{(i)} \left( W' h^{(i-1)} + A' x^{(i)} \right)$$  \hspace{1cm} (4)

This function approximates the change in the output of the RNN, when $W'$ changes to $(W + W', A + A')$. The parameter $\lambda$, that we defined in the objective function, will be used to make the contribution of $F^{(L)}_{\text{rnn}}$ at initialization small thus making pseudo-network a good approximation of RNN. Hence, we can observe that the pseudo-network is a good approximation of the RNN, provided the weights stay close to the initialization.

To complete the above definition of pseudo-network we define the two new notations in the above formula. For each $\ell \in [L]$, define $D^{(\ell)} \in \mathbb{R}^{m \times m}$ as a diagonal matrix, with diagonal entries

$$d_{r}^{(\ell)} := \| w_{r}^\top h^{(\ell-1)} + a_{r}^\top x^{(\ell)} \|_1, \quad \forall r \in [m].$$  \hspace{1cm} (5)

In words, the diagonal of matrix $D^{(\ell)}$ represents the activation pattern for the RNN cell at step $\ell$ at initialization.

Define $\text{Back}_{i \rightarrow j} \in \mathbb{R}^{d_{\text{out}} \times m}$ for each $1 \leq i \leq j \leq L$ by

$$\text{Back}_{i \rightarrow j} := BD^{(j)}W \ldots D^{(i+1)}W,$$

with $\text{Back}_{i \rightarrow j} := B$ for each $i \in [L]$. Matrices $\text{Back}_{i \rightarrow j}$ in Eq. (1) arise naturally in the computation of the first-order Taylor approximation (equivalently, gradients w.r.t. the parameters) using standard matrix calculus. Very roughly, one can think of $\text{Back}_{i \rightarrow j}$ as related to the backpropagation signal from the output at step $j$ to the parameters at step $i$.

### 4.2 Existence of good pseudo-network

Our goal is to construct $W^*$ and $A^*$ such that for any true input sequence $x = (x^{(2)}, \ldots, x^{(L-1)})$, if we define the normalized sequence $\bar{x} = (x^{(1)}, \ldots, x^{(L)})$, then with high probability we have

$$F^{(L)}(x; W^*, A^*) \approx F^*(\bar{x}).$$  \hspace{1cm} (6)

\[\text{The above outline is similar to prior work, e.g., [37]. Details can be quite different though, e.g., they only train $W$ and keep $A$ fixed to its initial value. Their contribution was also mainly in Step 2 and the other steps were similar to prior work.}\]
To simplify the presentation, in this sketch we will assume that \( p \), the number of neurons in the concept class, and the output dimension \( d_{\text{out}} \) are both equal to 1. Also, let the output weight \( b^\dagger \) := 1. These assumptions retain the main proof ideas while simplifying equations. Overall, we assume that the target function \( F^* : \mathbb{R}^{(L-2)\cdot(d-1)} \to \mathbb{R} \) on a given sequence is given by
\[
F^*(\mathbf{x}) = \Phi^*(\langle w^\dagger, [\mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(L-1)}]\rangle),
\]
where \( \Phi^* : \mathbb{R} \to \mathbb{R} \) is a smooth function and \( w^\dagger \in \mathbb{S}^{(L-2)\cdot(d-1)-1} \).

First, we state Lemma 6.2 in [13], which is useful for our construction of the matrices \( W^* \) and \( A^* \). Consider a smooth function \( \phi : [-1, 1] \to \mathbb{R} \). It can be approximated as a linear combination of step functions (derivatives of ReLU) for all \( u \in (-1, 1) \), i.e., there exists a “weight function” \( H : \mathbb{R}^2 \to \mathbb{R} \) such that \( \phi(u) \approx \mathbb{E}_{\alpha_1, \beta_1, b_0}[H(\alpha_1, b_0) I_{\alpha_1 u + \beta_1 \sqrt{1-u^2} + b_0 \geq 0}] \) where \( \alpha_1, \beta_1 \sim \mathcal{N}(0, 1) \) and \( b_0 \sim \mathcal{N}(0, 1) \) are independent random variables (we omitted some technical details).

The above statement can be straightforwardly extended to the following slightly more general version:

**Lemma 4.1.** For every smooth function \( \phi \), any \( \mathbf{w} \in \mathbb{S}^{d-1} \), and any \( \varepsilon \in \left(0, \frac{1}{\mathcal{C}_\varepsilon(\phi, \mathcal{O}(1))}\right) \) there exists a \( H : \mathbb{R}^2 \to (-\mathcal{C}_\varepsilon(\phi, 1), \mathcal{C}_\varepsilon(\phi, 1)) \), which is \( \mathcal{C}_\varepsilon(\phi, 1) \)-Lipschitz continuous and for all \( u \in \mathbb{S}^{d-1} \), we have
\[
\left| \phi(\mathbf{w}^\top u) - \mathbb{E}_{\mathbf{w} \sim \mathcal{N}(0,1), b_0 \sim \mathcal{N}(0,1)}[H(\mathbf{w}^\top \mathbf{w}, b_0) I_{\mathbf{w}^\top u + b_0 \geq 0}] \right| \leq \varepsilon.
\]

Very informally, this lemma states that the activation pattern of a one-layer ReLU network (given by \( \mathbf{I}_{\mathbf{w}^\top u \geq 0} \) at initialization) can be used to express a smooth function of the dot product of the input vector with a fixed vector. While the above statement involves an expectation, one can easily replace it by an empirical average with slight increase in error. This statement formed the basis for FFN and RNN results in [13, 32]. Can we use it for RNNs for our general concept class? An attempt to do so is the following lemma showing that the pseudo-network can express any smooth function of the hidden state \( \mathbf{h}^{(L-1)} \) and \( \mathbf{x}^{(L)} \).

**Lemma 4.2 (Informal).** For a given smooth function \( \phi \), a vector \( \mathbf{w} \in \mathbb{S}^{(m+d-1)} \), and any \( \varepsilon \in \left(0, \frac{1}{\mathcal{C}_\varepsilon(\phi, \mathcal{O}(1))}\right) \), there exist matrices \( W^* \) and \( A^* \) such that for every normalized input sequence \( \mathbf{x} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}) \) formed from a sequence \( \mathbf{x} \), we have with high probability,
\[
\left| F^{(L)}(\mathbf{x}; W^*, A^*) - \phi(\langle \mathbf{w}, [\mathbf{h}^{(L-1)}, \mathbf{x}^{(L)}_{d-1}]\rangle) \right| \leq \varepsilon,
\]
provided \( m = \text{poly}(\frac{1}{\varepsilon}, L, \mathcal{C}_\varepsilon(\phi, \mathcal{O}(1))) \). Vector \( \mathbf{x}^{(L)}_{d-1} \) is \( \mathbf{x}^{(L)} \) without the last coordinate, the bias term appended to each input.

The reason \( \mathbf{h}^{(L-1)} \) and \( \mathbf{x}^{(L)} \) come up is because they serve as inputs to the RNN cell when processing the \( L \)-th input. The proof sketched below uses the fact that RNNs are one-layer FFNs unrolled over time. Hence, we could try to apply the result of Lemma 4.1 to the RNN cell at step \( L \). However, a difficulty arises in carrying out this plan because the contributions of previous times steps also come up (as seen in the equations below) and it can be difficult to disentangle the contribution of step \( L \). This is addressed in the proof:

**Proof.** Recall that \( W, A \sim \mathcal{N}(0, \frac{2}{m} I) \). Also, recall that we have assumed for simplicity \( d_{\text{out}} = 1 \). Hence, \( B \) and \( \text{Back}_{i \rightarrow L} \) are row and column vectors respectively. For typographical simplicity, denote by \( b_r \) and \( \text{Back}_{i \rightarrow L, r} \) the respective \( r \)-th components of these vectors.

We set \( W^* := 0 \) and for every \( r \in [m] \), \( a_r^* := \frac{1}{m} b_r H(\sqrt{m/2}([w_r, a_r; d-1], \mathbf{w})) \), \( \sqrt{m/2} a_r d \) for a function \( H \) that we will describe below. With these choices we have
\[
F^{(L)}(\mathbf{x}; W^*, A^*) = \sum_{i=1}^{L} \text{Back}_{i \rightarrow L} D^{(i)} \left( W^* \mathbf{h}^{(i-1)} + A^* \mathbf{x}^{(i)} \right)
\]
\[
= \frac{1}{m} \sum_{i=1}^{L} \sum_{r \in [m]} b_r \text{Back}_{i \rightarrow L, r} H(\sqrt{m/2}([w_r, a_r; d-1], \mathbf{w})) \cdot \mathbf{I}_{\mathbf{w}^\top \mathbf{h}^{(i-1)} + a_r^* \mathbf{x}^{(i)} \geq 0}.
\]
In the last step, we have simplified the formula using sum over neurons. The first $L - 1$ summands in the outer sum above nearly vanish due to small correlation between $B$ and $\text{Back}_{i \rightarrow L}$ for $i < L$ (see Lemma 4.1). Recall that $\text{Back}_{L \rightarrow L} = B$ and thus the correlation is not small for $i = L$. This gives

$$F^{(L)}(x; W^*, A^*) \approx \frac{1}{m} \sum_{r \in [m]} b_r^2 H(\sqrt{m/2}((w_r, a_{r:d-1}), W)), \sqrt{m/2}a_{r,d} \cdot \mathbb{I}_{w_r} h^{(i-1)} + a_r^\top x^{(i)} \geq 0.$$ 

Now, this resembles a discretized version of Lemma 4.1. We can substitute $u$ as $[h^{(L-1)}, x^{(L)}]$ in Lemma 4.1 and use concentration bounds with respect to the randomness of weights $W$ and $A$ to complete the proof. 

More generally, with much more technical work, it might be possible to prove an extension of the above lemma asserting the existence of a pseudo-network approximating a sum of functions of type $\sum_{i \in [L]} \mathbb{P}(w_i, [h^{(i-1)}, x^{(i)}])$. However, even so it is not at all clear what class of functions of $x$ this represents because of the presence of the hidden state vectors.

Thus, the major challenge in constructing $W^*$ and $A^*$ to express the functions from the desired concept class is to use the information contained in $h^{(i)}$. The construction of $W^*$ in [27] is not able to use this information and ignores it by treating it as noise (which is also non-trivial). The idea underlying our construction is that $h^{(i)}$ in fact contains information about all the inputs $x^{(1)}, \ldots, x^{(L)}$ up until step $\ell$. Furthermore and crucially, this information can be recovered approximately by a linear transformation (Theorem 4.5 below). This enables us to show:

**Theorem 4.3 (Existence of pseudo-network approximation for target function; abridged statement of Theorem D.2 in the appendix).** For every target function $F^*$ of the form Eq. (1), there exist matrices $W^*$ and $A^*$ such that with probability at least $1 - e^{-\Omega(\rho^2)}$ over $W, A, B$, we have for every normalized input sequence $x = (x^{(1)}, \ldots, x^{(L)})$ formed from a true sequence $\bar{x}$,

$$|F^{(L)}(x; W^*, A^*) - F^*(x^{(1)}, [x^{(2)}, \ldots, x^{(L-2)}])| \leq \epsilon + \frac{1}{\text{poly}(\rho)},$$

provided $m \geq \text{poly}(\rho, L, \epsilon^{-1}, C_\epsilon(\Phi, O(\epsilon^{-1})))$ and $\epsilon \geq \frac{1}{\text{poly}(\rho)}$.

**Proof sketch.** By Theorem 4.5, there exists a matrix $\overline{W}^{[L]}$ such that $\overline{W}^{[L]} h^{(L-1)} \approx [x^{(1)}, \ldots, x^{(L)}]$ for all input sequences $[x^{(1)}, \ldots, x^{(L)}]$. We can modify $\overline{W}^{[L]}$ to get $[x^{(2)}, \ldots, x^{(L-1)}]$. Hence, by using $[\overline{W}^{[L]} w^i, 0]$ as $\overline{W}$ and $\Phi^*$ as $\phi$ in Lemma 4.2, we can have $F^{(L)}(x; W^*, A^*) \approx \Phi^*([w^i, \overline{W}^{[L]} h^{(L-1)})] \approx F^*(x)$, implying $F^{(L)}$ and $F^*$ are close. Accounting for all the errors in inversion and approximation of function, we get the final bound.

**Re-randomization.** In the proof sketches of Lemmas 4.2 and Theorem 4.3 above we swept a technical but critical consideration under the rug: the random variables $\{w_r, a_r\}_{r \in [m]}$, $\overline{W}^{[L]}$, $\{\text{Back}_{i \rightarrow L}\}_{i \in [L]}$ and $\{h^{(i)}\}_{i \in [L]}$ are not independent. This invalidates application of standard concentration inequalities w.r.t. the randomness of $W$ and $A$—this application is required in the proofs. Here our new variation of the re-randomization technique from [27] comes in handy. The basic idea is the following: whenever we want to apply concentration bounds w.r.t. the randomness of $W$ and $A$, we divide the set of rows into disjoint sets of equal sizes. For each set, we will re-randomize the rows of the matrix $[W, A]_r$, show that the matrices $\overline{W}^{[L]}$, $\{\text{Back}_{i \rightarrow L}\}_{i \in [L]}$ and $\{h^{(i)}\}_{i \in [L]}$ don’t change a lot and then apply concentration bounds w.r.t. the new weights in the set. Finally, we account for the error from each set.

### 4.3 The rest of the proof

Having shown that there exists a pseudo-network approximation of the RNN that can also approximate the concept class, we will complete the proof by showing that SGD can find matrices with performance similar to $W^*$ and $A^*$ on the population objective $\text{Obj}(\cdot)$. Lemma D.3 shows that the training loss decreases with time. The basic idea is to use the fact that within small radius of perturbation, overparameterized RNNs behave as a linear network and hence the training can be
analyzed via convex optimization. Then, we show using Lemma 4.4 that the Rademacher complexity for overparameterized RNNs is bounded. Again, the main idea here is that overparameterized RNNs behave as pseudo-networks in their overparameterized regime and hence their Rademacher complexity can be approximated by the Rademacher complexity of pseudo-networks. Finally, using generalization bounds on the Rademacher complexity, we get the final population-level objective in Theorem D.5.

4.4 Invertibility of RNNs at initialization

In this section, we describe how to get back \( x^{(1)}, \ldots, x^{(L)} \) from the hidden state \( h^{(L)} \). The following lemma states that any linear function can be represented by a one-hidden layer FFN with activation function ReLU, with a small approximation error of the order \( \frac{1}{\sqrt{m}} \).

**Lemma 4.4.** [A simpler continuous version can be found in Lemma C.7 in the appendix] For any \( v \in \mathbb{R}^d \), the linear function taking \( x \) to \( v^\top x \) for \( x \in \mathbb{R}^d \), can be represented as

\[
\left| v^\top x - p^\top \sigma(Tx) \right| \leq \frac{\|v\| \cdot \|x\|}{\sqrt{m}},
\]

with \( p = 2Tv \), where \( T \in \mathbb{R}^{m \times d} \) is a matrix with elements i.i.d. sampled from \( \mathcal{N}(0, 1) \).

Using the above lemma, we will show that the hidden state \( h^{(L)} \) can be inverted using a matrix \( \tilde{W}^{(L)} \) to get back the input sequence \( x^{(1)}, \ldots, x^{(L)} \).

**Theorem 4.5.** [Informal version of Theorem D.1] There exists a set of matrices \( \{\tilde{W}^{(\ell)}\}_{\ell \in [L]} \), which can possibly depend on \( W \) and \( A \), such that for any \( \varepsilon_x < \frac{1}{L} \) and any given normalized sequence \( x^{(1)}, \ldots, x^{(L)} \), with probability at least \( 1 - e^{-\Omega(\rho^2)} \) we have

\[
\|\|x^{(1)}, \ldots, x^{(L)}\| - \tilde{W}^{(L)}^\top h^{(L)}\|_\infty \leq \text{poly}(L, \rho, m^{-1}, \varepsilon_x).
\]

Very roughly, the above result is obtained by repeated application of Lemma 4.4 to go from \( h^{(\ell)} \) to \( (h^{(\ell-1)}, x^{(\ell)}) \) starting with \( \ell = L \). This uses the fact that the RNN cell is a one-hidden layer neural network and hence Lemma 4.4 is applicable. Several difficulties need to be overcome to carry out this plan.

One difficulty is that a naive application of Lemma 4.4 results in exponential blowup of error with \( L \). The reason is as follows: in Lemma 4.4, the factor 2 in the linear transformation \( p \) appears because of the activation function ReLU. A naive application will result in defining \( \tilde{W}^{(\ell)} \) inductively as \( 2|\tilde{W}^{(\ell-1)}| A \), for \( 2 \leq \ell \leq L \), with \( W^{[1]} = A \). This will lead to the approximation error exploding exponentially with \( L \) in induction, since the approximation error at each step will depend on the norm of \( W^{(\ell-1)} \).

However, one can observe from the proof of Lemma 4.4 that it suffices to use the following linear transformation: \( p(w) = w^\top v \|ww^\top x\| \geq 0 \), which helps in removing the factor of 2. However, this requires knowing the indicator function \( \mathbb{I}_{w^\top x = 0} \). This implies for RNNs, we need to know the indicator matrix \( D^{(\ell)} \) at each step.

To do so, we define a base sequence \( x_{(0)} \), whose indicator matrices \( \{D^{(\ell)}_{(0)}\}_{\ell \in [L]} \) will be used to approximate the indicator matrix \( D^{(\ell)} \) at each step \( \ell \) for any sequence \( x \). That is, \( \tilde{W}^{(\ell)} \) will be defined inductively as \( [D^{(\ell)}_{(0)} \tilde{W}^{(\ell-1)}], D^{(\ell)}_{(0)} A \) for \( 2 \leq \ell \leq L \), with \( W^{[1]} = D^{(1)}_{(0)} A \). We then show that the approximation error of the indicator matrix builds up to give an error with polynomial dependence on \( \varepsilon_x \) and \( L \). We defer the technical details of this resolution to the full proof in the appendix.

Secondly, we apply re-randomization to tackle the dependence between \( W, A, \{h^{(\ell)}\}_{\ell \in [L]} \) and \( \{\tilde{W}^{(\ell)}\}_{\ell \in [L]} \). We performed few toy experiments on the ability of invertibility for RNNs at initializa-
tion (Sec. I). We observed, as predicted by our theorem above, that the error involved in inversion decreases with the number of neurons and increases with the length of the sequence (Fig. 4).

5 On concept classes

It is apparent that our concept class is very general as it allows arbitrary dependencies across tokens. To concretely illustrate the generality of our concept class, and to compare with previous work, we show that our result implies that RNNs can recognize a simple formal language \( D_{L_1} \). Here we are working in the discrete setting where each input token comes from \( \{0, 1\} \) possibly represented as a vector when fed to the RNN. For a sequence \( z \in \{0, 1\}^L \), we define \( D_{L_1}(z) \) to be 1 if the number of 1’s in \( z \) is exactly 1 and define it to be 0 otherwise. We can show that \( D_{L_1} \) is not representable in the concept class of \([37]\) (see Theorem [H.1] in the appendix). However, we can show that the language \( D_{L_1} \) can be recognized with a one-layer FFN with one neuron and quadratic activation. The idea is that we can simply calculate the number of 1’s in the input string, which is doable using a single neuron. This implies that our concept class can represent language \( D_{L_1} \) with low complexity.

More generally, we can show that our concept class can efficiently represent pattern matching problems, where strings belong to a language only if they contain given strings as substrings. In general, we can show that our concept class can express general regular languages. However, the complexity of the concept class may depend super-polynomially on the length of the input sequence, depending on the regular language (more discussion in sec. [H]). Some regular languages allow special treatment though. For example, consider the language PARITY. PARITY is the language over alphabet \( \{0, 1\} \) with a string \( w = (w_1, \ldots, w_j) \in \text{PARITY} \) iff \( w_1 + \ldots + w_j = 1 \mod 2 \), for \( j \geq 1 \). We can show in sec. [H] that PARITY is easily expressible by our concept class with small complexity. RNNs perform well on regular language recognition task in our experiments in Sec. I. Figuring out which regular languages can be efficiently expressed by our concept class remains an interesting open problem.

6 Limitations and Conclusions

We proved the first result on the training and generalization of RNNs when the functions in the concept class are allowed to be essentially arbitrary continuous functions of the token sequence. Conceptually the main new idea was to show that the hidden state of the RNN contains information about the whole input sequence and this can be recovered via a linear transformation. We believe our techniques can be used to prove similar results for echo state networks.

Two main limitations of the present work are: (1) Our overparameterized setting requires the number of neurons to be large in terms of the problem parameters including the sequence length—and it is often qualitatively different from the practical setting. Theoretical analysis of practical parameter setting remains an outstanding challenge—even for one-hidden layer FFNs. (2) We did not consider generalization to sequences longer than those in the training data. Such a result would be very interesting but it appears that it would require stronger assumptions than our very general assumptions about the data distribution. Our techniques might be a useful starting point to that end: for example, if we knew that the distributions of the hidden states are similar at different times steps and the output is the same as the hidden state (i.e. \( B \) is the identity) then our results might easily generalize to higher lengths. We note that to our knowledge the limitation noted here holds for all works dealing with generalization for RNNs. (3) Understanding LSTMs remains open.

Our work addresses theoretical analysis of RNNs and is not directly concerned with applications. We do not anticipate any immediate societal impact of our work. In the long run, it may help improve RNNs and translate to impact on the applications.

7 Acknowledgements

We thank Sanjeev Arora, Raghav Somani and Abhishek Shetty for their valuable suggestions on improving the presentation of the paper. We also thank the reviewers, the area chair and the senior area chair for their valuable suggestions regarding this work.
References


