Proceedings of Machine Learning Research 1–19

On the limit of gradient decent for Simple Recurrent Neural Networks with finite precision*

Rémi Eyraud REMI.EYRAUD@UNIV-ST-ETIENNE.FR
Univ Lyon, UJM-Saint-Etienne, CNRS, Laboratoire Hubert Curien UMR 5516, Saint-Etienne

Volodimir Mitarchuk VOLODIMIR.MITARCHUK@UNIV-ST-ETIENNE.FR
Univ Lyon, UJM-Saint-Etienne, CNRS, Laboratoire Hubert Curien UMR 5516, Saint-Etienne

Abstract

Despite their great practical successes, the understanding of neural network behavior is still a topical research issue. In particular, the class of functions learnable in the context of a finite precision configuration is an open question. In this paper, we propose to study the limits of gradient descent when such a configuration is set for the class of Simple Recurrent Networks (SRN). We exhibit conditions under with the gradient descend will provably fail and argue that these conditions correspond to classical experimental setup. We also design a class of SRN based on Deterministic finite State Automata (DFA) that fulfills the failure requirements. The definition of this class is constructive: we propose an algorithm that, from any DFA, constructs a SRN that computes exactly the same function, a result of interest by its own.

1. Introduction

One of the main challenges Machine Learning is facing is the lack of understanding of the reasons of the practical successes of deep learning. Indeed, whole research fields have been revolutionized by the boom of deep neural networks, from Signal (Xie et al., 2017) to Image Processing (Hemanth and Estrela, 2017) including finance (Zhang et al., 2021), for instance, but much remain to be done to fully understand their capabilities and limits.

One way of developing our comprehension of these models is to provide theoretical analyses of the processes at hand, a goal that has been shown particularly challenging. If we focus on deep models for sequential data, like Recurrent Neural Networks (RNN), the main theoretical result states that vanilla RNN, the Simple Recurrent Networks (SRN) (Elman, 1990), are able to compute any Turing Machine (Siegelmann and Sontag, 1992). Though of great interest, this theorem says few about the models at use nowadays: its proof relies on the use of unbounded time and necessitates neurons of infinite precision.

When finite precision is required, a more recent work (Chung and Siegelmann, 2021) shows that it is possible to maintain the Turing completeness for a RNN if the algorithm has access to a potentially unbounded amount of neurons. If the number of neurons is fixed, Weiss et al. (2018) practically shows that the expressivity, that is, the class of functions a class of models can compute, drastically decreases for all types of RNN, most of them falling in classes equivalent to the expressivity of finite state machines. Theoretically, the same type of results have been obtained for saturated RNN, a process that forces each neuron to have a binary output (Merrill et al., 2020).

* This work is supported in part by the ANR TAUDoS (ANR-20-CE23-0020).

In addition to expressivity, another limitation should be investigated: not all possibly computable functions in a finite precision configuration can be learned using currently used learning algorithms. Indeed, neural nets learning almost exclusively relies on the use of variants of a gradient descent, an approach whose limits is well-studied (see Netrapalli (2019) for a recent survey).

Regrouping both questions, this work studies the limitations of gradient descent for Simple Recurrent Networks in a finite precision configuration. In particular, we exhibit a sufficient condition for such networks to not be learnable using back propagation through time, the prevailing variant of gradient descent used to learn RNN from sequences. Though a bit technical, we argue that the conditions of our theorem correspond to frequently used experimental setups.

To show that a wide range of networks are not learnable in this context we exhibit a class of SRN that matches the conditions of our theorem. This class is obtained via an algorithm we provide that builds a SRN from any Deterministic Finite state Automata (DFA).

Section 2 of this paper is devoted for notations and the introduction of different necessary notions for the proofs. In Section 3, we develop the arguments towards the gradient descent failure in finite precision. In Section 4, we present a non learnable class of SRN and show its interest. We conclude this paper with a discussion of its interest in Section 5.

2. Preliminaries

In this section are enumerated all the notations and elementary operations that are used in this paper.

2.1. Elements of Linear Algebra

Numbers are noted by a lowercase italic letter, *e.g.*, $x \in \mathbb{R}$, the vectors of dimension higher than 2 by a bold lowercase letter, *e.g.*, $\mathbf{x} \in \mathbb{R}^d$ for $d > 1$, and the matrices by a bold uppercase letter, *e.g.*, $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$. For $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{x}[j]$ represents the j^{th} coordinate of the the vector $\mathbf{x}$. $Id_d = Diag(1, \ldots, 1) \in \mathbb{R}^{d \times d}$ is the identity matrix of dimension d. Given $x \in \mathbb{R}$ and $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$, $x * \mathbf{W}$ denotes multiplication of all values of $\mathbf{W}$ by x.

Given two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ we note $[\mathbf{u}\ \mathbf{v}]$ the matrix of $\mathbb{R}^{d \times 2}$ whose columns are the original vectors. We extend this definition to matrices in an obvious way.

For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, $\langle \mathbf{x}, \mathbf{y} \rangle$ is the usual scalar product in $\mathbb{R}^d$. If we fix $\mathbf{w} \in \mathbb{R}^d$ the function $\langle \mathbf{w}, \cdot \rangle : \mathbb{R}^d \ni \mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle \in \mathbb{R}$ is a linear form and belongs to the dual space of $\mathbb{R}^d$.

For $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ we define the transposition operation by $\mathbf{W}^T \in \mathbb{R}^{d_2 \times d_1}$ where the rows of $\mathbf{W}$ are columns of $\mathbf{W}^T$. For $\mathbf{x} = (x_1, \ldots, x_d)^T \in \mathbf{R}^d$, in this paper $\|\mathbf{x}\| = \sqrt{\sum_{s=1}^{d} (x_s)^2}$.

$\odot$ is the Hadamard product between vectors: if $\mathbf{x} = (x_1, \ldots, x_d)^T$ and $\mathbf{y} = (y_1, \ldots, y_d)^T$ we have $\mathbf{x} \odot \mathbf{y} = (x_1 * y_1, \ldots, x_d * y_d)^T$.

$\oplus$ is the concatenation operation on vectors. For $\mathbf{x} \in \mathbb{R}^{d_1}$ and $\mathbf{y} \in \mathbb{R}^{d_2}$ we have $\mathbf{x} \oplus \mathbf{y} \in \mathbb{R}^{d_1 + d_2}$ (where d_1 and d_2 might be different): if $\mathbf{x} = (x_1, \ldots, x_{d_1})$ and $\mathbf{y} = (y_1, \ldots, y_{d_2})$ then $\mathbf{x} \oplus \mathbf{y} = (x_1, \ldots, x_{d_1}, y_1, \ldots, y_{d_2})$

2.2. Elements of arithmetic

Definition 1 (Finite precision configuration) *A finite precision configuration is defined by a tuple of positive integers (b, M, Ex), b being the basis, M and Ex representing the number of digits allocated for the "significand" and the "exponent", respectively. In this configuration a number x is represented by:*

$$x = (sign) \times (significand) \times (b^{exponent})$$

For example: $x = -1 \times 12.35 \times 10^0$
The sign belongs to $\{-1, 1\}$ *which is encoded with one digit. The significand is of the form*

$$significand = x_1 \ \cdots \ x_j.x_{j+1} \ \cdots \ x_M$$

with $x_i \in \{0, \ldots, b-1\}$*, for* $1 \leq i \leq M$*.*

This notation allows multiple representations for the same number, indeed the float number from the example can be written as follows:

$$x = -1 \times 12.35 \times 10^0$$
$$x = -1 \times 1.235 \times 10^1$$

To overcome this drawback, a number is in normal form if the index j is equal to one and $x_1 \neq 0$. The total number of exponents allowed by Ex digits is $R := \sum_{s=0}^{Ex-1} b^s$, thus we can generate only a finite amount of float number in a such configuration.
The setting (b, M, Ex) generates the set $\mathbb{G}_{(b,M,Ex)}$ composed of all numbers representable following these restrictions. We will abusively use the notation $\mathbb{G}$ instead of $\mathbb{G}_{(b,M,Ex)}$ for the sake of simplicity when there is no ambiguity.
As an example, in the IEEE 754 norm, the simple floats are encoded in base $b = 2$ on 32 bits, with 1 bit for the sign, $M = 23$ for the significand and $Ex = 8$ for the exponent.

Arithmetic's operations in a finite precision configuration might sometimes be counter intuitive. For instance, suppose one wants to compute the addition of two numbers x and s. Let $x = (sign_x) * x_1.x_2 \cdots x_M * b^{exponent_x}$ and $s = (sign_s) * s_1.s_2 \cdots s_M * b^{exponent_s}$ such that $exponent_s < exponent_x$. In order to compute $x + s$, one first needs to match the exponents of x and s. Without lost of generality, we can fix the highest exponent and modify the magnitude of the other number

$$s = \underbrace{0.0 \cdots 0}_{D \ zeros} s_1 \cdots s_{M-D} * b^{exponent_x}$$

with $D = exponent_x - exponent_s$.
This illustrates the consequences of rounding in finite precision, since the D last digits of s are not taken into account in the addition. Worse, if $D > M$, adding s to x will return x.

2.3. Elements of Calculus

As usual, $\circ$ defines the operation of composition $f \circ g(x) = f(g(x))$.
We will use this established result:

Theorem 2 *(Schwartz, 1997) Let $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ a linear function and $f : \mathbb{R}^q \to \mathbb{R}^d$ a function differentiable at $\mathbf{p}$ with its derivative at $\mathbf{p}$ noted f'. Then for all $\mathbf{k} \in \mathbb{R}^q$ we have:*

$$\frac{1}{\delta}\big(\mathcal{L}(f(\mathbf{p} - \delta\mathbf{k})) - \mathcal{L}(f(\mathbf{p})\big) = \mathcal{L}\big(\frac{1}{\delta}\Big(f(\mathbf{p} - \delta\mathbf{k})) - f(\mathbf{p})\Big)\big) \xrightarrow{\delta \to 0} \mathcal{L}\big(f'(\mathbf{p})\big)$$

This theorem can be found on page 94 of the cited book.
We also need the following lemma:

Lemma 3 *Let $\sigma : \mathbb{R} \ni x \mapsto \frac{1}{1+e^{-x}}$ and $c > 0$. If $|x| \geq \ln\left(\frac{1}{c}\right) + 1$ then: $\sigma'(x) \leq c$*

The proof of this lemma is given in the appendix.

Lemma 4 *(Underflow integer) Let (b, M, Ex) the finite precision configuration respecting the EEEI 754 norm. There exists J such that $\sigma(J) = 1$ and $\sigma(-J) = 0$ and J is defined by*

$$J = \lfloor \ln(b)l \rfloor + 1 \text{ with } l := \frac{b^{Ex} - 1}{b - 1} - \left\lfloor \frac{b^{Ex} - 1}{2(b - 1)} \right\rfloor$$

The proof of this lemma is given in the appendix.

2.4. Elements of language theory

Let Σ be a finite set of symbols called an alphabet. Σ^* represents all the finite sequences on Σ and ϵ is the empty sequence. Any subset L of Σ^* is called a language.
For $\omega \in \Sigma^*$, $|\omega|$ represents its length, $\omega \cdot \omega'$ is the concatenation of two sequences.

Definition 5 (DFA) *A Deterministic Finite State Automata (DFA) is entirely defined by $(\Sigma, Q, q_1, \delta, F)$ where: Σ is a finite alphabet; Q is the finite set of states; q_1 is the initial state; $\delta : Q \times \Sigma \to Q$ is the transition function; $F \subseteq Q$ is the set of accepting states*

The language represented by a DFA is the set $L = \{\omega \in \Sigma^* : \delta^*(q_0, \omega) \in F\}$, where δ^* is the transitive extension of the transition function: $\delta^*(q, \epsilon) = q$, $\delta^*(q, a \cdot \omega) = \delta^*(\delta(q, a), \omega)$ for $q \in Q, a \in \Sigma, \omega \in \Sigma^*$.

Definition 6 (One hot encoding) *Let $\Sigma = \{a_1, a_2, \ldots, a_m\}$ be a finite ordered alphabet of size m. Let $\{e_1, e_2, \ldots e_m\}$ be the canonical basis of the vector space $\mathbb{R}^m$, defined by: $e_i[j] = 1$ if $i = j$ and 0 if $i \neq j$. We define a one-hot encoding as a bijection:*

$$\phi : \ \Sigma \ni a_i \mapsto e_i \in \{e_1, e_2, \ldots e_m\}$$

The definition of a one-hot encoding is naturally extended to the elements of Σ^* by the bijection $\phi^* : \Sigma^* \to \{e_1, e_2, \ldots, e_m\}^*$ with $\phi^*(a_1 a_2 \cdots a_\kappa) \mapsto (\phi(a_1), \phi(a_2), \cdots, \phi(a_\kappa))$, where $\{e_1, e_2, \ldots, e_m\}^*$ is the set of all finite sequences of vectors of elements in $\{e_1, e_2, \ldots, e_m\}$. For simplicity, we might slightly abuse the notation and write $\phi(\omega)$ instead of $\phi^*(\omega)$.

2.5. Elements of Deep Learning

A neuron is a function $\sigma_{\mathbf{w}} : \mathbb{R}^d \ni \mathbf{x} \mapsto \sigma(\langle \mathbf{w}, \mathbf{x} \rangle) \in \mathbb{R}$. In this paper the activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is fixed and will be $\sigma : x \mapsto \frac{1}{1+e^{-x}}$ the logistic sigmoïdal function.
In Deep Learning the same notation is usually used for the activation function as a one variable scalar function and for a whole activation layer. To avoid any confusion, in this paper σ' will refer to the derivative of the function $\sigma : \mathbb{R} \to \mathbb{R}$ and $\sigma^\nabla : \mathbb{R}^d \to \mathbb{R}^d$ such that

$$\sigma^\nabla(x_1, \ldots, x_d) = (\sigma'(x_1), \ldots, \sigma'(x_d))$$

Definition 7 (FP-RNN) *A Finite Precision Recurrent Neural Network (FP-RNN) $\mathcal{R}$ is entirely defined by $(\mathcal{E}, \mathcal{D}, \mathbf{h}_1)$ where:*

$$\begin{aligned} &\mathbf{h}_1 \in \mathbb{G}^h \textit{is the initial hidden state} \\ &\mathcal{E} : \mathbb{G}^i \times \mathbb{G}^h \ni (\mathbf{x}, \mathbf{h}) \mapsto \mathcal{E}(\mathbf{x}, \mathbf{h}) \in \mathbb{G}^h && \textit{(the encoder)} \\ &\mathcal{D} : \mathbb{G}^h \ni \mathbf{h} \mapsto \mathcal{D}(\mathbf{h}) \in \mathbb{G}^o && \textit{(the decoder)} \end{aligned}$$

where i, h, and o are natural numbers.

Given a sequence $\omega = a_1 a_2 \ldots a_\kappa$ the execution of $\mathcal{R}$ on ω is totally expressed by the sequence $\big((\mathbf{y}_1, \mathbf{h}_1), (\mathbf{y}_2, \mathbf{h}_2), \ldots, (\mathbf{y}_\kappa, \mathbf{h}_\kappa)\big)$ recursively generated for $1 < s \leq \kappa$:

$$\mathbf{h}_s = \mathcal{E}(\phi(a_s), \mathbf{h}_{s-1}) \text{ (the hidden states) and } \mathbf{y}_s = \mathcal{D}(\mathbf{h}_s) \text{ (the output)}$$

In this paper, we focus on a particular class of FP-RNN:

Definition 8 (FP-SRN) *A Finite Precision Simple Recurrent Network (FP-SRN) is a RNN where the encoder $\mathcal{E}$ and the decoder $\mathcal{D}$ are defined by:*

$$\begin{aligned} &\mathcal{E} : \mathbb{G}^i \times \mathbb{G}^h \ni (\mathbf{x}, \mathbf{h}) \mapsto \sigma\left(\mathbf{W}_h \cdot (\mathbf{x} \oplus \mathbf{h} \oplus 1)\right) \in \mathbb{G}^h \\ &\mathcal{D} : \mathbb{G}^h \ni \mathbf{h} \mapsto \sigma\left(\mathbf{W}_o \cdot (\mathbf{h} \oplus 1)\right) \in \mathbb{G}^o \end{aligned}$$

where σ is the activation function, $\mathbf{W}_h$ is a matrix with h number of rows and $i + h + 1$ number of columns, and $\mathbf{W}_o$ is a matrix with o rows and $h + 1$ columns.

This definition used the usual trick that includes the bias in the weight matrix $\mathbf{W}_h$.

In supervised Machine Learning, the training of a neural network requires the use of a loss function:

Definition 9 *A loss function is a function $L : \mathbb{G}^o \times \mathbb{G}^o \to \mathbb{G}$ that compare computationally the output of a model on a given training data and the expected target value. Example the Binary Cross entropy for $p, q \in]0, 1[$:*

$$L(p, q) = -p \cdot \ln(q) - (1 - p) \cdot \ln(1 - q)$$

2.6. Elements of graph theory

For the purpose of our study, we need two basic definitions from graph theory:

Definition 10 (Oriented Graph) *An oriented graph is a tuple $G = (V, E)$ where V is a finite set of vertices and E is a set of edges on V: $E = \{(u, v) \in V \times V \mid u \neq v\}$.*

Definition 11 (Path) *Given a graph $G = (V, E)$, a path is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $\forall i \in \{1, \ldots, k-1\}$, $(v_i, v_{i+1}) \in E$ and $\forall i, j \in \{1, \ldots, k\}$, $v_i \neq v_j$.*

3. Finite precision SRN and learnability

Let L be a formal language. Solving the binary classification problem on L is answering the question: "for $\omega \in \Sigma^*$ do we have $\omega \in L$?". For this task, the whole sequence has to be processed before answering, therefor if we train SRNs for this problem we should apply the decoder only on the last hidden state. Thus the execution of a SRN $\mathcal{R}$ on a sequence ω, will be a sequence of the form $(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_\kappa, \hat{y}_\omega)$ where $\kappa = |\omega|$. For this setting we suppose that we are given a labeled training set $S = \{(\omega_1, y_1), \ldots (\omega_n, y_n)\}$ where $\omega_i \in \Sigma^*$ and $y_i = 1$ if $\omega_i \in L$ and $y_i = 0$ if $\omega_i \notin L$. In order to execute a SRN $\mathcal{R}$ on ω we have to one-hot encode ω via a ϕ. We will write $\mathcal{R}(\omega)$ instead of $\mathcal{R}(\phi(\omega))$ out of simplicity since ϕ is a bijection. Figure 1 visually illustrates an SRN of hidden dimension 4, suited for the binary classification task.

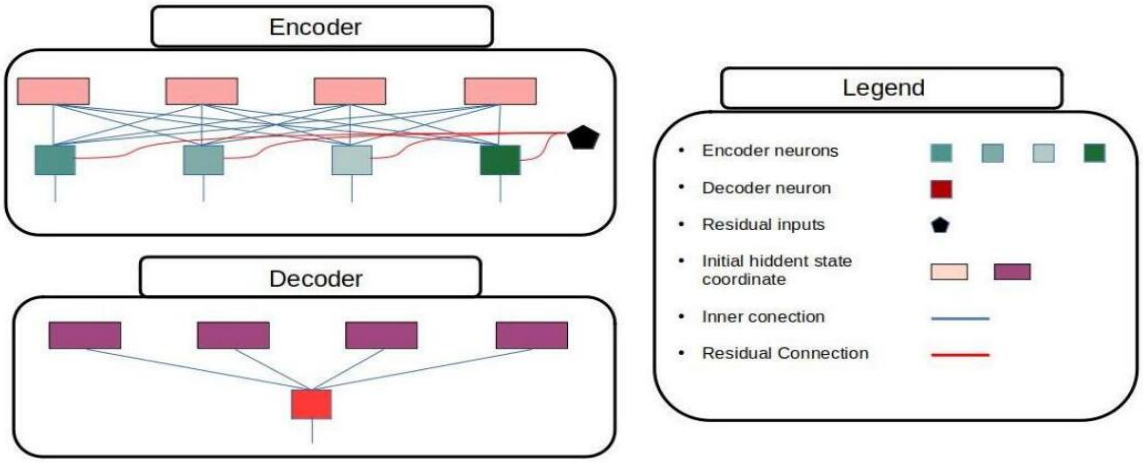

Figure 1: Illustration of the Encoder Decoder architecture

3.1. Back Propagation Trough Time

To train neural networks the usual approach is the one of Gradient Descent (GD). Its classical implementation for RNN is called Back Propagation Trough Time (BPTT) that we detail in this section.

Let $\mathcal{R}$ be an FP-SRN with hidden dimension h and $S = \{(\omega_1, y_1), \ldots (\omega_n, y_n)\}$ a labeled training set. Let $L : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ be a differentiable loss function. For a $(\omega, y) \in S$ we update the parameters of $\mathcal{R}$ using (ω, y) by computing

$$\frac{\partial L(\mathcal{R}(\omega), y)}{\partial(\mathbf{W}_h, \mathbf{W}_o)}$$

BPTT is the technique that allows this computation. Let $\omega = a_1 \ldots a_\kappa$, with $a_t \in \Sigma$ for all t. In order to obtain the output of the FP-SRN on ω, $\mathcal{R}(\omega)$, we have to compute recursively $\mathbf{h}_\kappa$ and then $\mathcal{D}(\mathbf{h}_\kappa)$, with $\mathbf{h}_{t+1} = \mathcal{E}(\phi(a_t), \mathbf{h}_t) \quad 1 \leq t < \kappa$. A way of seeing the recursive computation of the RNN is to unroll the calculation so that we obtain a pseudo feed-forward neural network $\mathcal{R}_\kappa$ evaluated at ω with $(\kappa + 1) + 1$ layers including the loss of the output

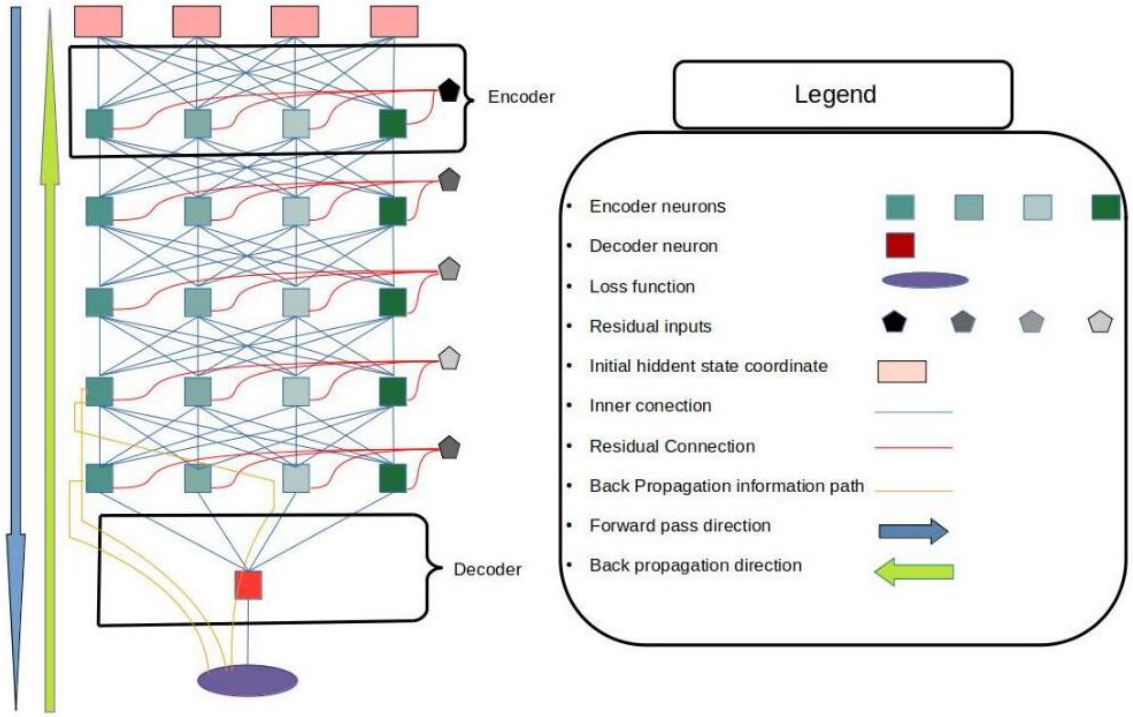

Figure 2: Illustration of the unrolling of the RNN for the BPTT

computation $L\left(\mathcal{D}(\mathbf{h}_\kappa), y\right)$. Every parameter of this pseudo feed-forward network $\mathcal{R}_\kappa$ is then updated by back propagating the gradient. The difference between $\mathcal{R}_\kappa$ and a real feed-forward network is that the encoder linear layers share the same parameters $\mathbf{W}_h$. As each parameter needs to be updated, we focus on a given one, $w_{i,j}$, in the matrix $\mathbf{W}_h$ where $1 \leq i \leq h$ and $1 \leq j \leq h + m$ specify its position, m being the size of the alphabet. The unfold network is used to back propagate the gradient for the parameter update. Figure 2 illustrates the unrolling procedure of an SRN of hidden dimension 4 evaluated on a word of length $\kappa = 5$. Following the spirit of the representation of the gradient back propagation used in Ancona et al. (2017), the pseudo feed-forward network $\mathcal{R}_\kappa(\omega)$ is seen as a graph. The vertices of this graph $\mathcal{R}_\kappa(\omega)$ are tuples $(v, G_v(\omega))$ where v is a function used in the neural network (i.e. a neuron or the loss function), and $G_v(\omega)$ represents the variables that the function v is fed with while computing $\mathcal{R}(\omega)$. This graph representation of the computation $\mathcal{R}(\omega)$ allows us to use the previously cited framework for the gradient back propagation which is a sum of products over all paths $P_{i,j}$ in the graph between the node of the loss function $(L, G_L(\omega))$ and the neuron node $(\sigma_{\mathbf{w}_i}, G_{\sigma_{\mathbf{w}_i}}(\omega))$ hosting the parameter of interest $w_{i,j}$:

$$\frac{\partial L(\mathcal{R}_\kappa(\omega), y)}{\partial w_{i,j}} = \sum_{p \in P_{i,j}} \prod_{(v, G_v) \in p} \frac{\partial v}{\partial w_{i,j}}(G_v(\omega))$$

It is important to note that due to the fact that weight matrix $\mathbf{W}_h$ is reused κ times in the computation, the set $P_{i,j}$ contains paths that link the loss node with the neuron node

containing $w_{i,j}$ present in the first encoder layer, second encoder layer, etc., up until the κ encoder layer present in $\mathcal{R}_\kappa(\omega)$. In Figure 2, we exhibit some back propagation paths that are represented in yellow. These paths transmit information for the update of one parameter in the neuron ■. The difference between the framework in Ancona et al. (2017) and the representation here is that here the gradient is computed with respect to the parameters of the neural network and in the cited article the gradient is computed with respect to the input of the neural net. However the framework is valid due to the symmetry of the scalar product.

3.2. A condition for BPTT failure

In this section we develop our argument of BPTT failure in a finite precision framework. Let $\mathbb{G}$ be the set of all float numbers representable in a given finite precision configuration (b, M, Ex). Let Σ be an alphabet with m elements, let $\mathcal{R}$ be an FP-SRN with hidden dimension h. We suppose that $\mathcal{R}$ is calibrated for processing sequences $\omega \in \Sigma^*$ for a binary classification task, and we dispose a labeled training set S of size n. That means that the encoder weight matrix $\mathbf{W}_h$ of $\mathcal{R}$ has h rows and $h+m$ columns, the decoder weight matrix $\mathbf{W}_o$ has one row and h columns. Recall that the rows of the matrix $\mathbf{W}_h$ are denoted $\mathbf{w}_i$ for $1 \leq i \leq n$. We set

$$\|\mathbf{w}^*\| := max\{\|\mathbf{w}\| \ : \ \mathbf{w} \in \{\mathbf{w}_1, \ldots, \mathbf{w}_n\}\}; \ \mathring{w} = min\{|w| \ : w \text{ is a parameter of } [\mathbf{W}_h \mathbf{W}_o]\}$$

In finite precision and normal form, we have $\mathring{w} = sign * \beta_1.\beta_2 \cdots \beta_M * b^\lambda$.

Theorem 12 *With the notations just above, we set $\varepsilon = b^D$ with $D = \lambda - (M+1)$ and we suppose that $\exists N$ such that :*

1. *$\forall\ 1 \leq i \leq n$, $\forall\ t \geq 1$ and for all $e \in \phi(\Sigma)$ we have:*

$$|\langle \mathbf{w}_i : (e \oplus \mathbf{h}_t) \rangle| \geq N$$

2. *$\forall\ t \geq 1$ by setting $\mathbf{p} := \langle \mathbf{W}_o, \mathbf{h}_t \rangle$*

$$\left| L'\big(\sigma(\mathbf{p})\big)\sigma'(\mathbf{p}) * \langle \mathbf{W}_o, \mathbf{h}_t \odot (1 - \mathbf{h}_t) \rangle \right| \leq 1$$

3. *$hC < 1$ and $\frac{C}{1-hC} \leq \varepsilon$ where $C := \left[\sigma'(N) \times \left(\|\mathbf{w}^*\|\sigma'(N)\sqrt{h} + 1\right)\right]$*

Then for each sample (ω, y) of the training set we have:

$$\frac{\partial L(\mathcal{R}(\omega), y)}{\partial\,(\mathbf{W}_h, \mathbf{W}_o)} = 0$$

Note that N can be specified analytically using the other parameters of the theorem, as stated in Lemma 16 in the appendix.
The first hypothesis insures a certain degree of saturation of the activation function σ, i.e. the output of σ is close to 0 or 1. This hypothesis is important to have a small gradient, indeed the sigmoidal activation function admits small variation when the input is far from zero. The second hypothesis is a bound on the decoder gradient. The decoder gradient is

a common factor in the gradient of all parameters, so we need to have a control over this part. Finally the third hypotheses will allow us to bound the gradient.
For readability reasons, the complete proof of the theorem is given at the end of this paper. The idea of this theorem is: if the parameter that we try to reach by GD have the properties stated by the theorem then the Gradient will start to be rounded to zero even before reaching the target. The following example shows how these hypotheses are reasonable restrictions toward a classical practical framework.

3.3. A practical example of Theorem 12

We fix the alphabet $\Sigma = \{a, b\}$ and a FP-RNN $\mathcal{R}$. We treat at first the hidden dimension h as a variable in order to observe its impact on the requirements and consider the finite precision configuration $(b, M, Ex) = (2, 23, 8)$ what refers to the simple float number in the EEEI 754 norm. By the lemma 4 we obtain that $\sigma(-89) = \frac{1}{1+e^{89}} = 0$. We set $N := 89$, $\|\mathbf{w}^*\| = 267\sqrt{h}$ where $267 = 3 \times 89$ and $\mathring{w} = 89$. What we ask is that the parameter of the FP-SRN are in $[-267, -89] \cup [89, 267]$. Since we are doing the computations in basis $\tilde{b} = 10$ we obtain $\tilde{M} = 15$. In this setting $\lambda = 2$ therefor $D = -16$. We compute:

$$\begin{aligned}\sigma'(N) \cdot \left(\|\mathbf{w}^*\|\sigma'(N)\sqrt{h+2}+1\right) &= (h+2) \cdot 1.3246266325 \cdot 10^{-75} + 2.2273635618 \cdot 10^{-39} \\ &\leq h \cdot 10^{-74} + 10^{-38}\end{aligned}$$

We observe that we have a degree of freedom with the dimension h. Note that the above computations are doable in pytorch. If we set $h = 10^{36}$ this way we have $\sigma'(N) \cdot \left(\|\mathbf{w}^*\|\sigma'(N)\sqrt{h+2}+1\right) \leq 10^{-37}$. This leads us to the fact that with $N := 89$, $M = 15$, $\mathring{w} = 89$ and $\|\mathbf{w}^*\| = 267\sqrt{h}$ we would need a SRN with more than 10^{36} neurons, corresponding roughly to 10^{72} parameters to fall in a case Theorem 12 does not apply. Actually, one of the largest neural net, GPT-3, has less than 10^{10} parameters.

4. A non learnable class

In this section we present a class of non learnable FP-SRN. According to Theorem 12, in a setting tuned for a binary classification task on formal language, there exist combinations of parameters that cannot be reached by GD. We provide an example of such event based on the simplest class of the language theory, the DFA.

4.1. An algorithm to construct a SRN from any DFA

In this section we design an algorithm, DFA2SRN, that outputs a FP-SRN from any DFA. Let $(\Sigma, Q, q_1, \delta, F)$ be a DFA. In order to simulate the DFA, we have to simulate the transition function, to encode every state such that it can be read by the FP-SRN, and then the model has to simulate the function discriminating between elements of F and $Q \setminus F$. The encoding of the states of the DFA follows the spirit of one-hot encoding. The transition function is simulated by the encoder and the discrimination function will be simulated by the decoder.
An example of the run of the algorithm is given in Section 4.2 below.

The main idea is to make the hidden states h_t, $t \geq 1$, be an encoding of the state q reached in the DFA while parsing a sequence up to its t^{th} element. Thus $\mathbf{W}_h$ has to encode the transition function, and $\mathbf{W}_o$ has to encode the $q \in F$ relation.

We set $\mathfrak{Q} := Card(Q)$, $m := Card(\Sigma)$. We assume that $Q = \{q_1, q_2, \ldots, q_{\mathfrak{Q}}\}$ and we define $I_F \subset \{1, \ldots, \mathfrak{Q}^2 m\}$ with

$$(\{i, i+1, \ldots, (i+\mathfrak{Q}m)-1\} \subset I_F \iff q_i \in F)$$

The linear parts of the SRN are defined by:

$$\mathbf{W}_h \in \mathbb{G}^{z_1 \times z_2} \ ; \ \mathbf{h}_1 \in \mathbb{G}^{z_1} \text{ with } \mathbf{h}_1[1] = 1, \mathbf{h}_1[j] = 0 \text{ if } j \neq 1;$$
$$\mathbf{W}_o \in \mathbb{G}^{1 \times z_1} \text{ with } \mathbf{W}_o[j] = J \text{ if } j \in I_F \text{ and } -J \text{ if } j \notin I_F$$

with $z_1 = \mathfrak{Q}^2 m$, $z_2 = \mathfrak{Q}^2 m + m$, and J the smallest integer saturating the activation function as defined in Lemma 4.

For more clarity the matrix $\mathbf{W}_h$ is divided into two sub-matrices for its definition. $\mathbf{W}_h^i \in \mathbb{G}^{z_1 \times m}$ and $\mathbf{W}_h^h \in \mathbb{G}^{z_1 \times z_1}$. We define $\mathbf{W}_h^i := 2 * \left.\begin{bmatrix} Id_m \\ \vdots \\ Id_m \end{bmatrix}\right\} \mathfrak{Q}^2$ identity matrices piled up.

For the construction of $\mathbf{W}_h^h$, we start by defining for every state q_k, $1 \leq k \leq \mathfrak{Q}$, a column vector $\mathbf{u}_k \in \mathbb{G}^{\mathfrak{Q}^2 m}$ by the following rules:

R1: For $1 \leq s \leq m$ if $\delta(q_k, a_s) = q_j$ then $\mathbf{u}_k[r] = -1$ with $r = (j-1)\mathfrak{Q}m + (k-1)m + s$

R2: All the other coordinates of $\mathbf{u}_k$ are set to -3

Note that by definition of a DFA every vector $\mathbf{u}_k$ has exactly $\mathfrak{Q}$ coordinates set to -1 and $\mathfrak{Q}^2 m - \mathfrak{Q}$ coordinates set to -3. Rule R1 defines an encoding for $(q_k, a_s, \delta(q_k, a_s))$ therefor the vector $\mathbf{u}_k$ contains all the transitions starting at q_k.

Afterward we generate, for every $q_k \in Q$, the matrix:

$$\mathbf{U}_k = \underbrace{\begin{bmatrix} \mathbf{u}_k & \mathbf{u}_k & \cdots & \mathbf{u}_k \end{bmatrix}}_{\mathfrak{Q}m \text{ times}} \in \mathbb{G}^{\mathfrak{Q}^2 m \times \mathfrak{Q}m}$$

Then we define the matrix $\mathbf{W}_h^h = \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 & , \cdots & \mathbf{U}_{\mathfrak{Q}} \end{bmatrix} \in \mathbb{G}^{\mathfrak{Q}^2 m \times \mathfrak{Q}^2 m}$.

Finally we define the matrix $\mathbf{W}_h = J * \begin{bmatrix} \mathbf{W}_h^i & \mathbf{W}_h^h \end{bmatrix}$.

In the following paragraph we prove that the constructed FP-SRN is simulating the functioning of the fixed DFA. For this purpose, in first place, we have to prove that, for any word $\omega = a_1 \cdots a_\kappa \in \Sigma^*$ of length κ, there exists a well defined correspondence between

$$\Big[\sigma\Big(\mathbf{W}_h\big(\phi(a_1) \oplus h_1\big)\Big), \ldots, \sigma\Big(\mathbf{W}_h\big(\phi(a_\kappa) \oplus h_\kappa\big)\Big)\Big] \text{ and } [\delta(q_0, a_1), \ldots, \delta(q_{\kappa-1}, a_\kappa)] \quad (1)$$

Secondly we will prove that we have:

$$\Big[\delta(q_{\kappa-1}, a_\kappa) \in F\Big] \iff \Big[\sigma\big(\mathbf{W}_o h_{\kappa+1}\big) = 1\Big] \quad (2)$$

Where $h_{\kappa+1} = \sigma\Big(\mathbf{W}_h\big(\phi(a_\kappa) \oplus h_\kappa\big)\Big)$

Let $e_s \in \phi\big(\Sigma\big)$ and $\mathbf{h} \in \mathbf{G}^{\mathfrak{Q}^2 m}$ an element of the canonical basis having a 1 at the coordinate $1 \leq g \leq \mathfrak{Q}^2 m$. This random choice of $\mathbf{h}$ has two purposes 1) show that the encoder operates stably no mater the starting vector as long as the vector is from the canonical basis, 2) demonstrate that $\mathfrak{Q}m$ canonical basis vectors are corresponding with one state in Q, but these vectors have the same effect on the encoder. The computation $\mathbf{W}_h(e_s \oplus \mathbf{h}) = J * \big(\mathbf{W}_h^i e_s + \mathbf{W}_h^h \mathbf{h}\big)$ gives us on one side the vector $\mathbf{W}_h^i e_s$ which is of the form:

$$\mathbf{W}_h^i e_s[k] = 2 \text{ if } (k \equiv s \text{ modulo } m) \text{ and } 0 \text{ otherwise}$$

For the other part of the equation, by the euclidean division of g by $\mathfrak{Q}m$, there exists two integers c and $\mathfrak{r}$ such that

$$g = c\mathfrak{Q}m + \mathfrak{r}$$

The matrix $\mathbf{W}_h^h$ is very redundant (we repeat $\mathfrak{Q}m$ times the same vector) therefor dividing by $\mathfrak{Q}m$ allows to find to which state of the DFA the vector $\mathbf{W}_h^h \mathbf{h}$ is related to (recalling the fact that applying a matrix on a canonical basis vector is equivalent to selecting a column of the matrix an returning it). By definition of the matrix $\mathbf{W}_h^h$ and the euclidean division, it follows that $\mathbf{W}_h^h \mathbf{h} = \mathbf{u}_c$. By construction of the vector $\mathbf{u}_c$, there is a unique $p \in \{1, \ldots, \mathfrak{Q}\}$ such that $\delta(q_c, a_s) = q_p$, implying that the for $r = (p-1)\mathfrak{Q}m + (c-1)m + s$ we have $\mathbf{u}_c[r] = -1$. Note that:

$$(p-1)\mathfrak{Q}m + (c-1)m + s \equiv s \text{ modulo } m$$

And no other coordinate set to -1 is equal to s modulo m. Suppose that we have: $(p'-1)\mathfrak{Q}m + (c'-1)m + s' \equiv s$ modulo m then we have $s' \equiv s$ modulo m But since $1 \leq s, s' \leq m$ we must have $s' = s$. That implies that the vector $\mathbf{W}_h^i e_s + \mathbf{u}_c$ has only one positive coordinate $(\mathbf{W}_h^i e_s)[r] + \mathbf{u}_c[r] = 2 - 1 = 1$. For all coordinates $\theta \neq r$ we have $(\mathbf{W}_h^i e_s)[\theta] + \mathbf{u}_c[\theta] = 2 - 3$ or $= 0 - 3$ or $= 0 - 1$. Subsequently by multiplying the vector $\mathbf{W}_h^i e_s + \mathbf{u}_c$ by J and applying the sigmoïdal function, in the finite precision configuration, the output vector will have 1 at the coordinate r and zeros elsewhere. Finally by Lemma 15 we conclude that the vector $\mathbf{h}_p = \sigma(J * \big(\mathbf{W}_h^i e_s + \mathbf{u}_c\big))$ is related to the state p of the DFA: The only coordinate of $\mathbf{h}_p$ that is not null is $\mathbf{h}_p[r]$.

Given the definition of $\mathbf{W_o}$ we have $\mathbf{W_0} h_p = J$ if $p \in F$ and $\mathbf{W_0} h_p = -J$ otherwise. Therefor, the output of the FP-RNN on a sequence is 1 if the parsing of the sequence in the DFA ends in a final state, and 0 otherwise.

4.2. An example of the DFA2SRN algorithm

Figure 3 is a representation of an automaton computing the language of all words ending by an a on the alphabet $\Sigma = \{a, b\}$ (the initial state is q_1, the sole accepting state is q_2). Let (b, M, Ex) be the finite precision configuration. We set J the smallest integer such that $\sigma(-J) = 0$ and $\sigma(J) = 1$. The existence and expression of this integer is given by Lemma 4 that is proven in Appendix.

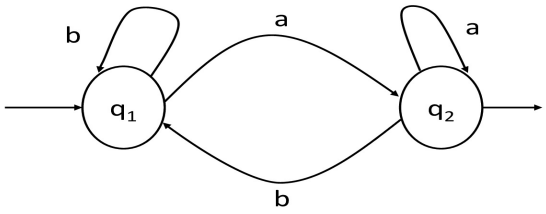

Figure 3: An example of a DFA

The algorithm gives us: $\mathbf{h}_1 = \begin{pmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{pmatrix}, \mathbf{W}_o = \begin{pmatrix}-1\\-1\\-1\\-1\\1\\1\\1\\1\end{pmatrix}$ $\mathbf{u_1} = \begin{pmatrix}-3\\-1\\-3\\-3\\-1\\-3\\-3\\-3\end{pmatrix}, \mathbf{u_2} = \begin{pmatrix}-3\\-3\\-3\\-1\\-3\\-3\\-1\\-3\end{pmatrix}$ and:

$$\mathbf{W}_h = J * \left[\begin{array}{cc|cccccccc} 2 & 0 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 \\ 0 & 2 & -1 & -1 & -1 & -1 & -3 & -3 & -3 & -3 \\ 2 & 0 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 \\ 0 & 2 & -3 & -3 & -3 & -3 & -1 & -1 & -1 & -1 \\ 2 & 0 & -1 & -1 & -1 & -1 & -3 & -3 & -3 & -3 \\ 0 & 2 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 \\ 2 & 0 & -3 & -3 & -3 & -3 & -1 & -1 & -1 & -1 \\ 0 & 2 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 \end{array}\right]$$

$\mathbf{W}_\mathbf{h}^\mathbf{i}$ $\mathbf{W}_\mathbf{h}^\mathbf{h}$ $\mathbf{u}_1$ $\mathbf{u}_2$ $\mathbf{U}_1$ $\mathbf{U}_2$

We consider the computation of the FP-SRN on the word a and the word b. Their one-hot encoding is $\phi(a) = \begin{pmatrix}1\\0\end{pmatrix}$ and $\phi(b) = \begin{pmatrix}0\\1\end{pmatrix}$.

We have: $\mathbf{W}_h(\phi(a) \oplus \mathbf{h}_1) = J * \begin{pmatrix}-1 & -1 & -1 & -3 & 1 & -3 & -1 & -3\end{pmatrix}^T$

After applying the activation function we obtain $\mathbf{h}_2 = \begin{pmatrix}0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\end{pmatrix}^T$ due to the overflow and underflow. The vector $\mathbf{h}_2$ is an accepting vector because $\sigma(\mathbf{W}_o\mathbf{h}_2) = \sigma(J) = 1$ due to the overflow. If we do the same computation for b we obtain $\mathbf{h}'_2 = \begin{pmatrix}0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\end{pmatrix}^T$. The vector $\mathbf{h}'_2$ is not an accepting vector because $\sigma(\mathbf{W}_o\mathbf{h}'_2) = \sigma(-J) = 0$ due to the underflow.

4.3. On the learnability of this SRN class

In this section we apply Theorem 12 to the class of FP-SRN constructed just above. In fact, Example 3.3 is based on that type of architecture. Let $\mathcal{R}$ be such a SRN. We suppose that the hidden dimension is $h \geq 1$ and that $\mathcal{R}$ is operating on a finite alphabet Σ of size m. We set the loss function to be the Cross Entropy. For a finite precision configuration (b, M, Ex), there exists a J such that $\|\mathbf{w}^*\| \leq 3 * J\sqrt{h+m}$, $\mathring{w} = J$ and $N = J$. Following Example 3.3, in a real situation it is clear that it is impossible to have a SRN constructed by DFA2SRN that does not fall into the requirements of Theorem 12.
The last thing to verify is the second requirement of the theorem. The formula of the Cross Entropy is $L(\mathcal{R}(\omega), y) = -y\ln(\mathcal{R}(\omega)) - (1-y)\ln(1-\mathcal{R}(\omega))$. Since $y \in \{0, 1\}$ one of the terms in the right hand-side of the above equation is equal to zero. Without loss of generality we suppose that $1 - y = 0$. Then $L(\mathcal{R}(\omega), y) = -y\ln(\mathcal{R}(\omega))$. We obtain that $|L'(\mathcal{R}(\omega))| = \frac{y}{\mathcal{R}(\omega)} = \frac{1}{\sigma(\mathbf{W}_o\mathbf{h}_\kappa)}$.
In finite precision this is not well defined, but if we consider the common part of the production resulting from the computation of the partial derivative $\frac{\partial L}{\partial w_{i,j}}(\mathcal{R}(\omega))$, that is the decoder part, noted F in the proof:

$$F = \frac{1}{\sigma(\mathbf{W}_o\mathbf{h}_\kappa)}\sigma(\mathbf{W}_o\mathbf{h}_\kappa)(1-\sigma(\mathbf{W}_o\mathbf{h}_\kappa))\langle\mathbf{W}_o, \mathbf{h}_\kappa \odot (1-\mathbf{h}_\kappa)\rangle = \alpha \cdot \beta$$

where $\alpha = (1-\sigma(\mathbf{W}_o\mathbf{h}_\kappa))$ and $\beta = \langle\mathbf{W}_o, \mathbf{h}_\kappa \odot (1-\mathbf{h}_\kappa)\rangle$. We have $\alpha \leq 1$ and $\beta = 0$ by construction since every hidden vector produced by the SRN (including the initial hidden state vector $\mathbf{h}_1$) has all its coordinates set to zero except one coordinate set to 1, therefor $\mathbf{h}_\kappa \odot (1-\mathbf{h}_\kappa) = \mathbf{0}_{\mathbb{G}^h}$. Thus $F = 0 \leq 1$, which allows us to state that the class of SRN obtained by DFA2SRN is not learnable by back propagation of the gradient.

5. Discussion

A first element of discussion is that our results, though stated for a classical gradient descent, is applicable to the widely used stochastic gradient descent: as all gradients are null under the assumptions of Theorem 12, so is its mean on any batch.

Another straightforward extension of our result concern other activation functions. Indeed, a similar argument can be made concerning any bounded function, like the hyperbolic tangent: the only difference being the value that saturates the activation.

One can wonder the interest of the class of SRN we exhibit as an example of a non-learnable class. Notice first that any RNN behaves like a finite state machine in a finite precision configuration (Pozarlik, 1998): its hidden state can only store a (huge) finite number of configurations. It is thus relevant to use the simplest finite state machines, the DFA, as starting point. Then, even if its construction is not trivial, it covers a large class of functions thus assessing the interest of the theorem. Finally, the algorithm constructing a SRN from any DFA is of interest by itself: it is, as far as we know, the first one allowing such construction. Indeed, if it is a widely accepted fact that SRN can compute any regular language, we surprisingly did not find any published work stating the result: the closest paper on the subject (Siegelmann, 1996) shows that for a given DFA with $\mathfrak{Q}$ states operating on an alphabet of size m there exists an SRN with $3\mathfrak{Q}m$ neurons that simulates the

computation of the DFA, however its proof is not constructive and thus does not provide an algorithm to build such models.

References

M. Ancona, E. Ceolini, C. Öztireli, and M. Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. *arXiv:1711.06104*, 2017.

S. Chung and H. Siegelmann. Turing completeness of bounded-precision recurrent neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

J. Hemanth and V. Estrela. *Deep learning for image processing applications*. IOS Press, 2017.

W. Merrill, G. Weiss, Y. Goldberg, R. Schwartz, N. A Smith, and E. Yahav. A formal hierarchy of rnn architectures. *CoRR*, 2020.

P. Netrapalli. Stochastic gradient descent and its variants in machine learning. *Journal of the Indian Institute of Science*, 99:201–213, 2019.

R. Pozarlik. What type of finite computations do recurrent neural networks perform? In *ICANN*, 1998.

L. Schwartz. *Analyse Volume 2 -Calcul différentiel et équations différentielles-*. broché, 1997.

H. Siegelmann. Recurrent neural networks and finite automata. *Computational intelligence*, 12(4):567–574, 1996.

H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. In *COLT*, pages 440–449, 1992.

G. Weiss, Y. Goldberg, and E. Yahav. On the practical computational power of finite precision RNNs for language recognition. In *ACL*, 2018.

D. Xie, L. Zhang, and L. Bai. Deep learning in visual computing and signal processing. *Appl. Comp. Intell. Soft Comput.*, 2017.

J. Zhang, J. Zhai, and H. Wang. A survey on deep learning in financial markets. In *Proc. of the First International Forum on Financial Mathematics and Financial Technology*. Springer, 2021.

6. Appendix

6.1. Proof of Theorem 12

This subsection is dedicated to the proof of the criterion for BPTT failure. To do this we are going to prove that, under the hypotheses of Theorem 12, the update generated by GD is rounded to zero for every parameter in the SRN. We fix a parameter $w_{i,j}$ in the encoder weight matrix $\mathbf{W}_h$, and compute recursively the gradient on the word ω with $|\omega| = \kappa$. We already established that $\frac{\partial L(\mathcal{R}_\kappa(\omega),y)}{\partial w_{i,j}} = \sum_{p\in P_{i,j}} \prod_{(v,T_v)\in p} \frac{\partial v}{\partial w_{i,j}}(G_v)$. By the BPTT technique $\sum_{p\in P_{i,j}} \prod_{(v,T_v)\in p} \frac{\partial v}{\partial w_{i,j}}(G_v)$ contains the update for a $w_{i,j}$ in every layer of the unrolled neural network $\mathcal{R}_\kappa$. The following lemma deals with the length of the paths in $P_{i,j}$.

Lemma 13 *With previous notations, the length of the paths in $P_{i,j}$ is always in $\{3, 4, \dots, \kappa + 2\}$ and the number of paths ϖ of each length is $Card(\{p \in P_{i,j}/|p| = \varpi\}) = h^{\varpi-3}$*

Proof We will reason layer by layer, starting with the last encoder layer in $\mathcal{R}_\kappa(\omega)$. This layer corresponds to the computation of the hidden state $\mathbf{h}_\kappa$. The occurrence of $(\sigma_{\mathbf{w}_i}, G_{\sigma_{\mathbf{w}_i}})$ in this layer is the closest to the root $(L, G_L(\omega))$, and is connected by a unique path of length 3. This last statement is clear considering the bottleneck shape of the graph $\mathcal{R}_\kappa(\omega)$ and the only vertex separating $(\sigma_{\mathbf{w}_i}, G_{\sigma_{\mathbf{w}_i}})$ and $(L, G_L(\omega))$ which is $(\sigma_{\mathbf{W}_o}, G_{\sigma_{\mathbf{W}_o}})$ corresponding to the decoder neuron. Therefor $Card(\{p \in P_{i,j}/|p| = 3\}) = h^{3-3} = 1$. Note that this reasoning is applicable for any $(\sigma_{\mathbf{w}_s}, G_{\sigma_{\mathbf{w}_s}})$, $1 \leq s \leq h$. Moving one layer backward in the encoder *i.e.* to the layer corresponding to the computation of $\mathbf{h}_{\kappa-1}$, the vertex $(\sigma_{\mathbf{w}_i}, G_{\sigma_{\mathbf{w}_i}})$ in this layer has h connections with the layer computing $\mathbf{h}_\kappa$, in other words $(\sigma_{\mathbf{w}_i}, G_{\sigma_{\mathbf{w}_i}})$ is connected to every node in the last encoder layer what represents h connections. This is due to the fully connected nature of the encoder. Then from the last layer each vertex admits one unique path to the loss vertex. Therefor $Card(\{p \in P_{i,j}/|p| = 4\}) = h^{4-3} = h$. Like for the previous layer, this reasoning is applicable for any $(\sigma_{\mathbf{w}_s}, G_{\sigma_{\mathbf{w}_s}})$, $1 \leq s \leq h$ in the layer. By applying this arguments recursively we extract the desired formula. ■

Lemma 4 (recall) (Underflow integer) Let (b, M, Ex) the finite precision configuration respecting the EEEI 754 norm. There exists J such that $\sigma(J) = 1$ and $\sigma(-J) = 0$ and J is defined by

$$J = \lfloor \ln(b)l \rfloor + 1 \text{ with } l := \frac{b^{Ex} - 1}{b - 1} - \left\lfloor \frac{b^{Ex} - 1}{2(b - 1)} \right\rfloor$$

.

Proof The total number of exponents obtainable in this configuration is: $\frac{b^{Ex}-1}{b-1} = \sum_{s=0}^{Ex-1} b^s$. The configuration (b, M, Ex) allows to have $u := \lfloor \frac{b^{Ex}-1}{2(b-1)} \rfloor$ positive exponents and $l := \frac{b^{Ex}-1}{b-1} - \lfloor \frac{b^{Ex}-1}{2(b-1)} \rfloor$ negative exponents. We make the assumption that $M < min\{u, l\}$ Let $K := \ln(b)l$ we have on one side: $\sigma(K) = \frac{1}{1+e^{-\ln(b)l}} \leq \frac{1}{1-b^l} = 1$ due to the rounding, and $\sigma(-K) = \frac{1}{1+e^{\ln(b)l}} = \frac{1}{1+b^l} = \frac{1}{b^l} = b^{-l}$ the smallest number we can have. Consequently if we set $J := \lfloor K \rfloor + 1$ we will still have $\sigma(J) = 1$ and $\sigma(-J) < b^{-l}$. Therefor $\sigma(-J)$ will be rounded to zero.
Note that in the EEEI 754 norm the exponent of the 32 bit float numbers we have $l = 126$, $u = 127$, and the assumption $M < min\{u, l\}$ is satisfied. ■

Lemma 14 *Using the previous notations, for all* $p \in P_{i,j}$ *where* $p = \big((v, G_v(\omega))_1, \ldots, (v, G_v(\omega))_\gamma\big)$ *we have:*

$$(v, G_v(\omega))_1 = (L, G_L(\omega)) \text{ and } (v, G_v(\omega))_2 = (\sigma_{\mathbf{W}_o}, G_{\sigma_{\mathbf{W}_o}}(\omega))$$

The proof of this lemma is straightforward: this fact is due to the characteristic of the decoder layer.

By Lemma 14 all the production in the expression $\sum_{p \in P_{i,j}} \prod_{(v,T_v)\in p} \frac{\partial v}{\partial w_{i,j}}(G_v)$ share a common factor $F := \left| L'\big(\sigma(\langle \mathbf{w}, \mathbf{h}_\kappa \rangle)\big) \frac{\partial \sigma(\langle \mathbf{W}_o, \cdot \rangle)}{\partial w_{i,j}}(\mathbf{h}_\kappa) \right|$ where $\frac{\partial \sigma(\langle \mathbf{W}_o, \cdot \rangle)}{\partial w_{i,j}}(\mathbf{h}_\kappa) = \sigma'(\langle \mathbf{w}, \mathbf{h}_\kappa \rangle) * \langle \mathbf{w}, \sigma^\nabla(\mathbf{h}_\kappa)) \rangle$

Thus by requirement 2 of Theorem 12, F is smaller than one. This allows us to bound the product referring to the path of the smallest length:

$$F * \underbrace{\sigma'(\langle \mathbf{w}_i, \mathbf{z}_{\kappa-1} \rangle)}_{\clubsuit} * \underbrace{\big[\langle \mathbf{w}_i, \mathbf{0}_{\mathbb{R}^m} \oplus \sigma^\nabla(\mathbf{W}_h \mathbf{z}_{\kappa-2})) \rangle + \tau\big]}_{\spadesuit} \tag{3}$$

The product $\clubsuit * \spadesuit$ is the application of the chain rule on $\frac{\partial \sigma \circ \langle \mathbf{w}_i, \cdot \rangle}{\partial w_{i,j}}(\mathbf{z}_{\kappa-1}) = \sigma'(\langle \mathbf{w}_i, \mathbf{z}_{\kappa-1} \rangle) * \frac{\partial \langle \mathbf{w}_i, \cdot \rangle}{\partial w_{i,j}}(\mathbf{z}_{\kappa-1})$. We have $\frac{\partial \langle \mathbf{w}_i, \cdot \rangle}{\partial w_{i,j}}(\mathbf{z}_{\kappa-1}) = \spadesuit$ by Theorem 2.

The value $\tau = \mathbf{z}_{\kappa-1}[j] \in \mathbb{G} \cap [0, 1]$ appears in the computation because $w_{i,j}$ is a parameter of the function $\langle \mathbf{w}_i : \cdot \rangle : \mathbb{G}^{h+m} \to \mathbb{G}$. The final point to clarify is the appearance of $\mathbf{0}_{\mathbb{R}^m}$ instead of $\phi(\omega_{\kappa-1})$. This is also explained by Theorem 2 simply by considering the function f being constant here on m first coordinates, thus the zeros on these coordinates.

Returning to Expression (3). We can bound it by:

$$F * \sigma'(\langle \mathbf{w}_i, \mathbf{z}_{\kappa-1} \rangle) * \big[\langle \mathbf{w}_i, \mathbf{0}_{\mathbb{R}^m} \oplus \sigma^\nabla(\mathbf{W}_h \mathbf{z}_{\kappa-2})) \rangle + \tau\big] \leq \sigma'(N)\Big(\|\mathbf{w}^*\|(\sigma'(N))\sqrt{h} + 1\Big)$$

We have $F \leq 1$ and $\sigma'(\langle \mathbf{w}_i, \mathbf{z}_{\kappa-1} \rangle) \leq \sigma'(N)$ by requirement 1. Then by setting $C := \big[\langle \mathbf{w}_i, \mathbf{0}_{\mathbb{R}^m} \oplus \sigma^\nabla(\mathbf{W}_h \mathbf{z}_{\kappa-2})) \rangle + \tau\big]$ we obtain:

$$\begin{aligned} C \leq & \|\mathbf{w}_i\| \cdot \|\mathbf{0}_{\mathbb{R}^m} \oplus \sigma^\nabla(\mathbf{W}_h \mathbf{z}_{\kappa-2}))\| + 1 \\ = & \|\mathbf{w}_i\| \cdot \sqrt{\sum_{s=1}^{h} \sigma'(\langle \mathbf{w}_s : \mathbf{z}_{\kappa-2} \rangle)^2} \quad + 1 \end{aligned}$$

and then

$$\begin{aligned} C \leq & \|\mathbf{w}^*\| \cdot \sqrt{\sum_{s=1}^{h} \sigma'(N)^2} + 1 \\ = & \|\mathbf{w}^*\|(\sigma'(N))\sqrt{h} + 1 \end{aligned}$$

Following the spirit of the previous reasoning and Lemma 13, we obtain:

$$\begin{aligned} & \sum_{p \in P_{i,j}, |p|=4} \prod_{\big(v, G_v(\omega)\big) \in p} \frac{\partial v}{\partial w_{i,j}}(G_v(\omega)) \\ \leq & h * F\Big(\sigma'(N)(\|\mathbf{w}^*\|(\sigma'(N))\sqrt{h} + 1)\Big)^2 \leq h * C^2 \end{aligned}$$

Where $C = \sigma'(N)(\|\mathbf{w}^*\|(\sigma'(N))\sqrt{h}+1)$
More generally:

$$\begin{aligned}
&\sum_{p\in P_{i,j},|p|=\varpi} \prod_{\left(v,G_v(\omega)\right)\in p} \left|\frac{\partial v}{\partial w_{i,j}}(G_v(\omega))\right| \\
\leq& h^{\varpi-3} * F\Big(\sigma'(N)(\|\mathbf{w}^*\|(\sigma'(N))\sqrt{h}+1)\Big)^{\varpi-2} \\
\leq& h^{\varpi-3} * C^{\varpi-2} \text{ by hypothesis 2}
\end{aligned}$$

At this point we can derive a bound on the partial derivative $\frac{\partial L(\mathcal{R}_\kappa(\omega),y)}{\partial w_{i,j}}$ which is:

$$\begin{aligned}
\left|\frac{\partial L(\mathcal{R}_\kappa(\omega),y)}{\partial w_{i,j}}\right| &= \left|\sum_{p\in P_{i,j}} \prod_{(v,G_v)\in p} \frac{\partial v}{\partial w_{i,j}}(G_v(\omega))\right| \\
&\leq \sum_{p\in P_{i,j}} \prod_{(v,G_v)\in p} \left|\frac{\partial v}{\partial w_{i,j}}(G_v(\omega))\right| \\
&\leq \sum_{s=1}^{\kappa} \sum_{\substack{p\in P_{i,j} \\ |p|=s+2}} \prod_{(v,G_v)\in p} \left|\frac{\partial v}{\partial w_{i,j}}(G_v(\omega))\right| \\
&\leq \sum_{s=1}^{\kappa} h^{s-1}C^s \\
&= C\cdot\sum_{s=0}^{\kappa-1} h^sC^s \leq C\cdot\sum_{s=0}^{\infty} h^sC^s = \frac{C}{1-hC}
\end{aligned}$$

By hypotheses number 3 we have $\frac{C}{1-hC} < \varepsilon$ Thus

$$\left|\frac{\partial L(\mathcal{R}_\kappa(\omega),y)}{\partial w_{i,j}}\right| \leq= \varepsilon$$

It follows, by requirement on ε, that if we add ε to any parameter of the weight matrix $\mathbf{W}_h$, it will have no effect. ■

6.2. Proofs of other lemmas

Lemma 15 *In DFA2SRN, a hidden vector $\mathbf{h}_p$ encodes the corresponding DFA state q_p.*

Proof Let examine the computation $\mathbf{W}_h^h\mathbf{h}_p$. We recall that $r = (p-1)\mathfrak{Q}m + (c-1)m + s$. Let $\mathbf{w}_r := \mathbf{W}_h^h\mathbf{h}_p$ the r^{th} column vector of the matrix $\mathbf{W}_h^h$ since the vector $\mathbf{h}_p$ has zeros everywhere except for a 1 at the position r. Recall that $1 \leq p, c \leq \mathfrak{Q}$ and $1 \leq s \leq m$ thus $\underbrace{(p-1)\mathfrak{Q}m+1}_{\alpha} \leq r \leq \underbrace{(p-1)\mathfrak{Q}m + (\mathfrak{Q}-1)m + m}_{\beta} = p\mathfrak{Q}m$ Therefor by applying the euclidean division $\div$ by $\mathfrak{Q}m$ on α, r, and β, we obtain $(p-1) + rest_\alpha \leq r \div \mathfrak{Q}m \leq p$ Thus the vector $\mathbf{w}_r$ is a copy of the vector $\mathbf{u}_p$ by the construction of the matrix $\mathbf{W}_h^h$. This allows to conclude that the constructed FP-SRN is simulating the execution of the DFA. ■

Lemma 3 (recall) *Let* $\sigma : \mathbb{R} \ni x \mapsto \frac{1}{1+e^{-x}}$ *and* $c > 0$. *If* $|x| \geq \ln\left(\frac{1}{c}\right) + 1$ *then:* $\sigma'(x) \leq c$

Proof Of the lemma
At first we recall some properties of the functions σ and σ'. We have:

$$\forall\ x \in \mathbb{R}\ \sigma(x) = 1 - \sigma(-x)$$

σ' is symmetric:

$$\forall\ x \in \mathbb{R}\ \sigma'(x) = \sigma'(-x)$$

Suppose that we want

$$\sigma'(N) \leq c$$

We have

$$\sigma'(N) = \frac{e^N}{(1+e^N)^2} \leq c$$

$$\begin{aligned}
\frac{e^N}{(1+e^N)^2} \leq c &\iff e^N \leq c * (1 + 2e^N + e^{2N}) \\
&\iff e^N - 2ce^N - ce^{2N} - c \leq 0 \\
&\iff -ce^{2N} + (1-2c)e^N - c \leq 0
\end{aligned}$$

Now we set $y := e^N$ and we obtain:

$$-cy^2 + (1-2c)y - c \leq 0$$

There for by solving the equation :

$$-cy^2 + (1-2c)y - c = 0$$

we will obtain the desired statement. The roots of the quadratic equation are given by

$$\begin{aligned}
y_1 &= \frac{1 - 2c + \sqrt{1-4c}}{2c} \\
y_2 &= \frac{1 - 2c - \sqrt{1-4c}}{2c}
\end{aligned}$$

In our problem c is quit small so $y_2 \approx c$ because

$$\begin{aligned}
y_2 = \frac{1 - 2c - \sqrt{1-4c}}{2c} &= \frac{1 - 2c - (1 - 2c - 2c^2 + o(c^2))}{2c} \\
&= c + o(c)
\end{aligned}$$

By using the limited development of $\sqrt{1+x}$. For the other root we have,

$$\begin{aligned}
y_1 = \frac{1 - 2c + \sqrt{1-4c}}{2c} &= \frac{1 - 2c + (1 - 2c - 2c^2 + o(c^2))}{2c} \\
&= \frac{1 - 2c - c^2 + o(c^2)}{c}
\end{aligned}$$

By applying the natural logarithm on y_1 and y_2 we obtain:

$$\ln(y_2) = \ln(c + o(c)) < 0$$

That means that this root is not going to interest us, because N has to be positive. On the other hand we have

$$\begin{aligned}\ln(y_1) &= \ln\left(\frac{1}{c}\right) + \ln(1 - (2c + c^2 + o(c^2))) \\ &= \ln\left(\frac{1}{c}\right) - (2c + c^2) + O(c^2)\end{aligned}$$

Thus by setting

$$N \geq \ln\left(\frac{1}{c}\right) + 1$$

We are sure to have to desired inequality. ■

The following lemma gives a sufficient condition on the number N in Theorem 12 such that the hypothesis 3 is satisfied.

Lemma 16 *Any number* $N \geq \ln\left(\frac{2\sqrt{h}\|\mathbf{w}^*\|}{\sqrt{1+4\sqrt{h}\|\mathbf{w}^*\|\eta}-1}\right) + 1)$ *fulfill the hypothesis of Theorem 12*

Proof We recall that $\mathbf{w}^*$ is the biggest raw vector in the encoder weight matrix $\mathbf{W}_h$ is the sense of the euclidean norm. The natural number h is the dimension of the FP-SRN. Let $\eta > 0$, under the notations from above we need:

$$\sigma'(N) \times \left(\|\mathbf{w}^*\|\sigma'(N)\sqrt{h} + 1\right) \leq \eta \tag{4}$$

We have $\sigma'(N) \times \left(\|\mathbf{w}^*\|\sigma'(N)\sqrt{h} + 1\right) = \sqrt{h}\|\mathbf{w}^*\|\sigma'(N)^2 + \sigma'(N)$
In order to solve (4), we solve the equation $\sqrt{h}\|\mathbf{w}^*\|\sigma'(N)^2 + \sigma'(N) = \eta$
By substituting $\sigma'(N)$ by y in the former equation we obtain a quadratic equation $ay^2 + by + c = 0$ where $a = \sqrt{h}\|\mathbf{w}^*\|$, $b = 1$ and $c = -\eta$. This equation admits 2 real roots $y_1 = \frac{-1-\sqrt{1-4ac}}{2a} < 0$ and $y_2 = \frac{-1+\sqrt{1-4ac}}{2a} > 0$. Thus for all $y \in [y_1, y_2]$ we have $ay^2 + by \leq \eta$. But since we fixed $y = \sigma'(N) > 0$ the solutions that interest us are $y \in [0, y_2]$. Following from that: $\left(\sigma'(N) \leq \frac{-1+\sqrt{1+4\sqrt{h}\|\mathbf{w}^*\|\eta}}{2\sqrt{h}\|\mathbf{w}^*\|}\right) \Rightarrow \left(\sigma'(N) \times \left(\|\mathbf{w}^*\|\sigma'(N)\sqrt{h} + 1\right) \leq \eta\right)$

Finally by summoning Lemma 3 we get the succeeding assertion:

$$\left(N \geq \ln\left(\frac{2\sqrt{h}\|\mathbf{w}^*\|}{\sqrt{1+4\sqrt{h}\|\mathbf{w}^*\|\eta}-1}\right) + 1\right) \Rightarrow \left(\sigma'(N) \times \left(\|\mathbf{w}^*\|\sigma'(N)\sqrt{h} + 1\right) \leq \eta\right)$$

■