UNIVERSITÀ
DEGLI STUDI
FIRENZE

Optimization methods for Recurrent Neural Networks training

Giulio Galvan

20th April 2016

Università degli studi di Firenze

Introduction

The model

Definition of RNN

Given an input sequence $\{\boldsymbol{u}\}_{t=1,\dots,T}$, with $\boldsymbol{u}_t \in \mathbb{R}^p$, the output sequence of a RNN $\{\boldsymbol{y}\}_{t=1,\dots,T}$, with $\boldsymbol{y}_t \in \mathbb{R}^o$, is defined by the following:

$$\boldsymbol{y}^t \triangleq F(\boldsymbol{z}^t) \tag{1}$$

$$\boldsymbol{z}^t \triangleq W^{out} \cdot \boldsymbol{a}^t + \boldsymbol{b}^{out} \tag{2}$$

$$\boldsymbol{a}^t \triangleq W^{rec} \cdot \boldsymbol{h}^{t-1} + W^{in} \cdot \boldsymbol{u}^t + \boldsymbol{b}^{rec} \tag{3}$$

$$\boldsymbol{h}^t \triangleq \sigma(\boldsymbol{a}^t) \tag{4}$$

$$\boldsymbol{h}^0 \triangleq \mathbf{0}, \tag{5}$$

where $\sigma(\cdot) : \mathbb{R} \to \mathbb{R}$ is a non linear function applied element-wise called **activation function**, $F(\cdot)$ is called **output function**.

The parameters of the net are $\{W^{out}, W^{in}, W^{rec}, \boldsymbol{b}^{rec}, \boldsymbol{b}^{out}\}$.

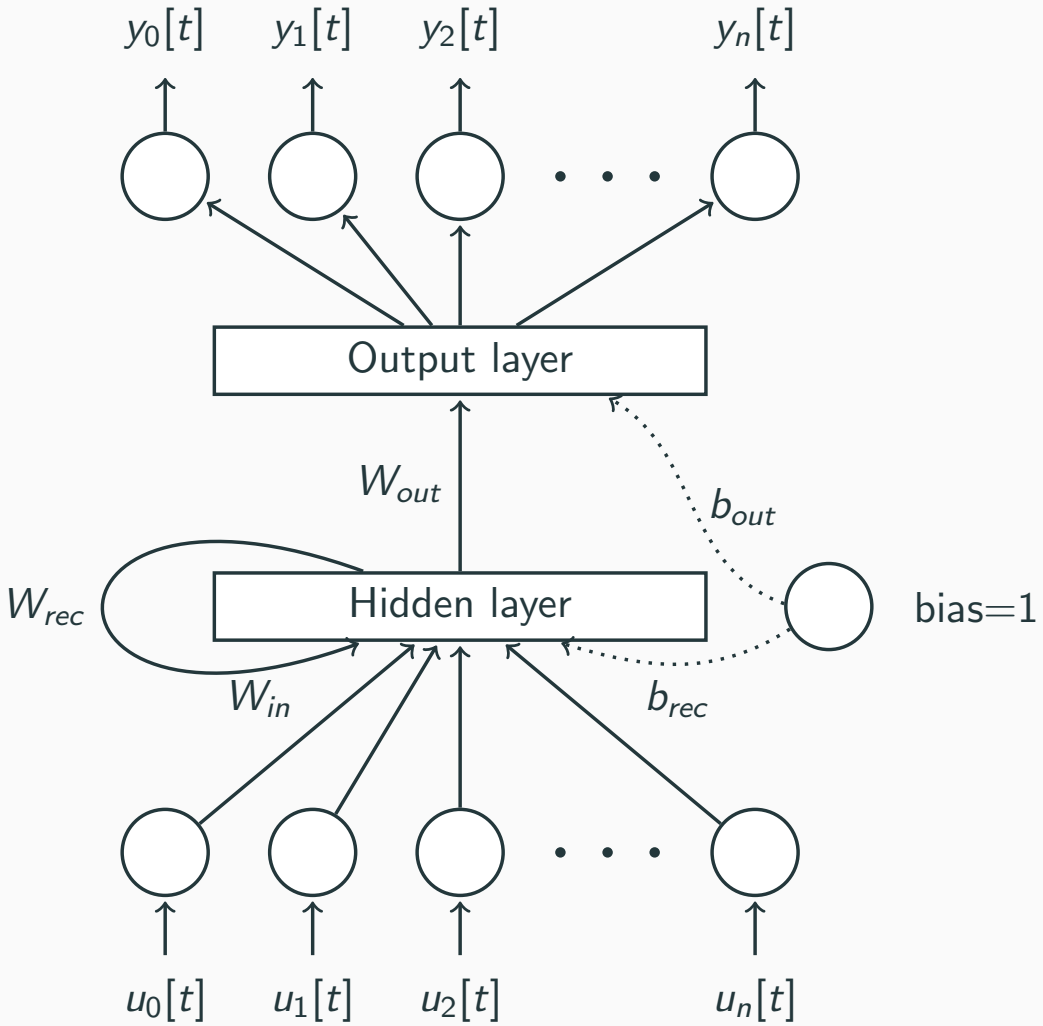


Figure 1: RNN model.

The optimization problem

Given a dataset D:

$$D \triangleq \{\{\overline{\boldsymbol{u}}^{(i)}\}_{t=1,\ldots,T}, \overline{\boldsymbol{u}}_t^{(i)} \in \mathbb{R}^p, \{\overline{\boldsymbol{y}}^{(i)}\}_{t=1,\ldots,T}, \overline{\boldsymbol{y}}_t^{(i)} \in \mathbb{R}^o; i = 1, \ldots, N\} \tag{6}$$

we define a loss function $L_D(\boldsymbol{x})$ over D as

$$L_D(\boldsymbol{x}) \triangleq \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} L_t(\overline{\boldsymbol{y}}_t^{(i)}, \boldsymbol{y}_t^{(i)}(\boldsymbol{x})), \tag{7}$$

where $L_t(\cdot, \cdot)$ is an arbitrary loss function for the time step t and $\boldsymbol{x}$ represents all the parameters of the network. The problem is

$$\min_{\boldsymbol{x}} L_D(\boldsymbol{x}) \tag{8}$$

Some learning examples

- Regression: mean squared error, linear output

$$L(\boldsymbol{y}, \boldsymbol{t}) = \frac{1}{M} \sum_{i=1}^{M} (y_i - t_i)^2, \quad F(\boldsymbol{y}) = \boldsymbol{y}. \tag{9}$$

- Binary classification: hinge loss, linear output

$$L(y, t) = \max(0, 1 - t \cdot y), \quad F(y) = y. \tag{10}$$

- Multi-way classification: cross entropy loss, softmax output

$$L(\boldsymbol{y}, \boldsymbol{t}) = -\frac{1}{M} \sum_{i=1}^{M} \log(y_i) \cdot t_i, \quad F(y_j) = \frac{e^{y_j}}{\sum_{i=1}^{M} e^{y_i}}. \tag{11}$$

Stochastic gradient descent (SGD)

Algorithm 1: Stochastic gradient descent

Data:

$D = \{\langle \boldsymbol{u}^{(i)}, \boldsymbol{y}^{(i)} \rangle\}$: training set

$\boldsymbol{x}_0$: candidate solution

m: size of each mini-batch

Result:

$\boldsymbol{x}$: solution

$\boldsymbol{x} \leftarrow \boldsymbol{x}_0$

while *stop criterion* **do**

  $I \leftarrow$ select m training example $\in D$

  $\alpha \leftarrow$ compute learning rate

  $\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \sum_{i \in I} \nabla_{\boldsymbol{x}} L(\boldsymbol{x}; \langle \boldsymbol{u}^{(i)}, \boldsymbol{y}^{(i)} \rangle)$

end

Gradient of a RNN

Gradient structure: unfolding

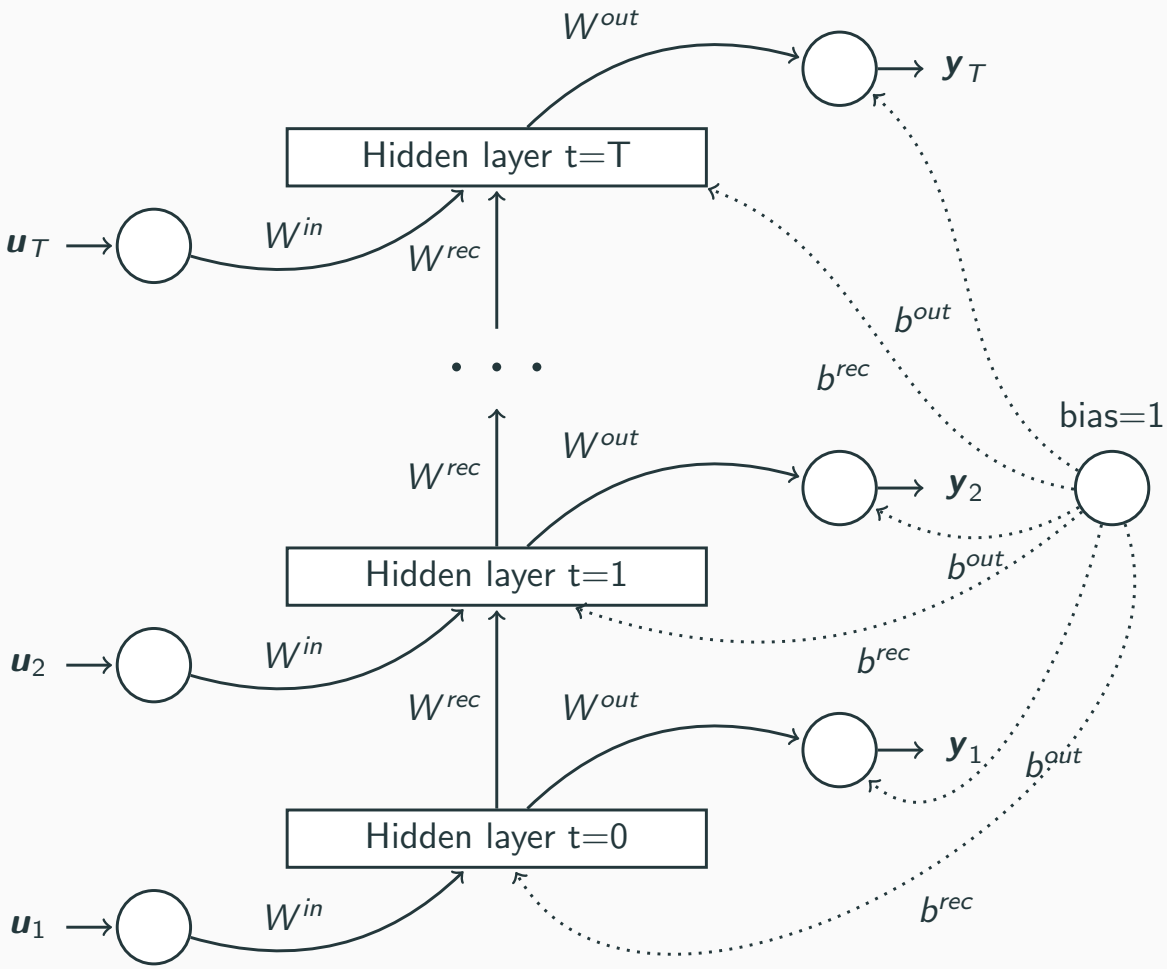


Figure 2: Unfolding of a RNN

Gradient structure: calculus

Consider, for ease of notation, the case where the loss function $L(\bar{\boldsymbol{u}}, \bar{\boldsymbol{y}})$ is defined only on the last step τ. Let $g(\boldsymbol{x}) : \mathbb{R}$ be the function defined by

$$g(\boldsymbol{x}) \triangleq L(F(\boldsymbol{z}^{\tau}(\bar{\boldsymbol{u}}; \boldsymbol{x}), \bar{\boldsymbol{y}}_{\tau})).$$

We compute the gradient as:

$$\frac{\partial g}{\partial W^{rec}} = \frac{\partial g}{\partial \boldsymbol{a}^{\tau}} \cdot \frac{\partial \boldsymbol{a}^{\tau}}{\partial W^{rec}} \tag{12}$$

$$= \nabla L^{T} \cdot J(F) \cdot \frac{\partial \boldsymbol{z}^{\tau}}{\partial \boldsymbol{a}^{\tau}} \cdot \frac{\partial \boldsymbol{a}^{\tau}}{\partial W^{rec}}. \tag{13}$$

In matrix notation we have:

$$\frac{\partial \boldsymbol{a}^t}{\partial W^{rec}} = \sum_{k=1}^{t} \frac{\partial \boldsymbol{a}^t}{\partial \boldsymbol{a}^k} \cdot \frac{\partial^+ \boldsymbol{a}^k}{\partial W^{rec}} \tag{14}$$

$$\frac{\partial^+ a^k}{\partial W_j^{rec}} = \begin{bmatrix} h_j^k & 0 & \cdots & \cdots & 0 \\ 0 & h_j^k & \cdots & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & \cdots & h_j^k \end{bmatrix} \tag{15}$$

$$\frac{\partial \boldsymbol{a}^t}{\partial \boldsymbol{a}^k} = \frac{\partial \boldsymbol{a}^t}{\partial \boldsymbol{a}^{k+1}} \cdot diag(\sigma'(\boldsymbol{a}^k)) \cdot W^{rec} \tag{16}$$

$$= \prod_{i=t-1}^{k} diag(\sigma'(\boldsymbol{a}^i)) \cdot W^{rec}. \tag{17}$$

The derivatives with respect to the other variables are computed in a similar fashion.

Gradient structure: temporal components

Putting all together we obtain:

$$\nabla_{W^{rec}} g = \sum_{k=1}^{\tau} \frac{\partial g}{\partial \boldsymbol{a}^{\tau}} \cdot \frac{\partial \boldsymbol{a}^{\tau}}{\partial \boldsymbol{a}^{k}} \cdot \frac{\partial^{+} \boldsymbol{a}^{k}}{\partial W^{rec}} \tag{18}$$

$$\triangleq \sum_{k=1}^{\tau} \nabla_{W^{rec}} L_{|k}. \tag{19}$$

We refer to $\nabla_{\boldsymbol{x}} g_{|k}$ as the **temporal gradient** for time step k w.r.t. the variable $\boldsymbol{x}$, and it is easy to see that it is the gradient we would compute if we replicated the variable $\boldsymbol{x}$ for each time step and took the derivatives w.r.t. to its k-th replicate.

The vanishing gradient problem

A pathological problem example

An input sequence:

marker	0	1	0	...	0	1	0	0
value	0.3	**0.7**	0.1	...	0.2	**0.4**	0.6	0.9

The predicted output should be the sum of the two one-marked positions (1.1).

Why is this a difficult problem?

Because of its long time dependencies.

A pathological problem example

An input sequence:

marker	0	1	0	...	0	1	0	0
value	0.3	**0.7**	0.1	...	0.2	**0.4**	0.6	0.9

The predicted output should be the sum of the two one-marked positions (1.1).

Why is this a difficult problem?

Because of its long time dependencies.

Vanishing gradient: an illustration

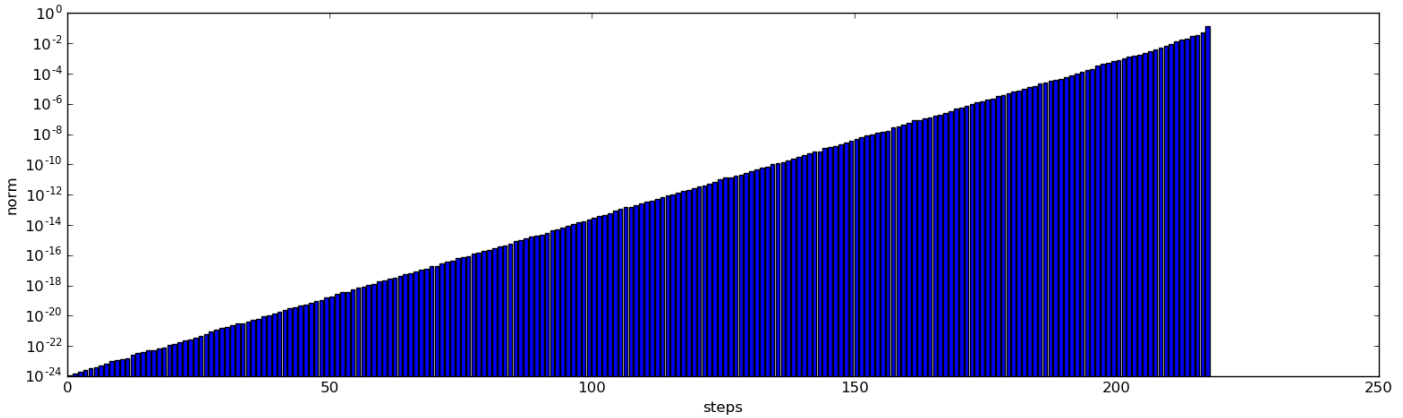


Figure 3: Norm of the temporal gradients at different time steps.

Vanishing gradient: a sufficient condition

$$\frac{\partial \boldsymbol{a}^t}{\partial \boldsymbol{a}^k} = \prod_{i=t-1}^{k} diag(\sigma'(\boldsymbol{a}^i)) \cdot W^{rec}. \tag{20}$$

Taking the singular value decomposition of W^{rec}:

$$W^{rec} = S \cdot D \cdot V^T \tag{21}$$

where S, V^T are squared orthogonal matrices and $D \triangleq diag(\mu_1, \mu_2, ..., \mu_r)$ is the diagonal matrix containing the singular values of W^{rec}. Hence:

$$\frac{\partial \boldsymbol{a}^t}{\partial \boldsymbol{a}^k} = \prod_{i=t-1}^{k} diag(\sigma'(\boldsymbol{a}^i)) \cdot S \cdot D \cdot V^T \tag{22}$$

Since U and V are orthogonal matrix, hence

$$\|U\|_2 = \left\|V^T\right\|_2 = 1,$$

and

$$\|diag(\lambda_1, \lambda_2, ..., \lambda_r)\|_2 = \lambda_{max},$$

we get

$$\left\|\frac{\partial \boldsymbol{a}^t}{\partial \boldsymbol{a}^k}\right\|_2 = \left\|\left(\prod_{i=t-1}^{k} diag(\sigma'(\boldsymbol{a}^i)) \cdot S \cdot D \cdot V^T\right)\right\|_2 \tag{23}$$

$$\leq \left(\sigma'_{max} \cdot \mu_{max}\right)^{t-k-1} \tag{24}$$

A new SGD approach for training RNNs

Initialization

Motivated by the bounds for the vanishing gradient on the singular values of the recurrent matrix we explored an initialization scheme which **scales the spectral radius** of such matrix.

Algorithm 2: Recurrent weight matrix initialization scheme

Data:

$\rho =$ desired spectral radius

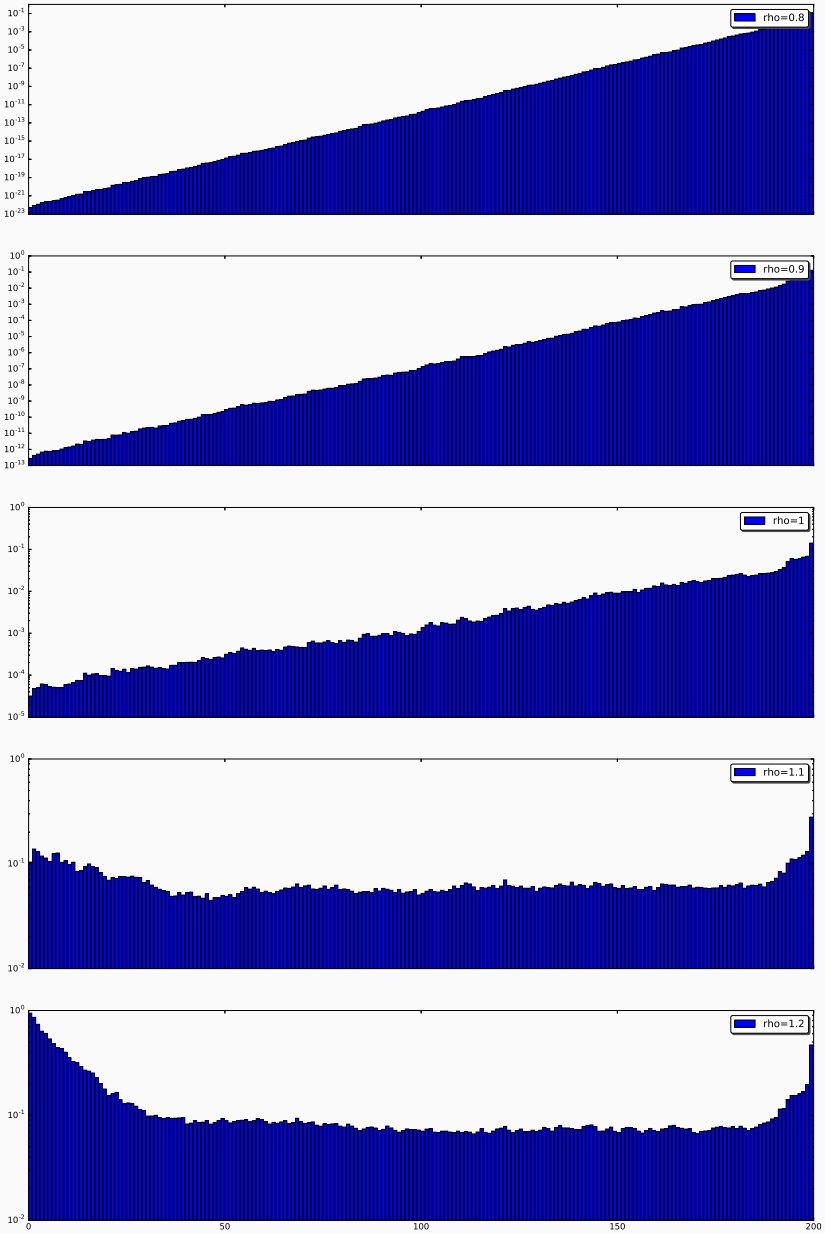
1 $W_{rec} \sim \mathcal{N}(0, \sigma^2)$

2 $r \leftarrow \text{spectral\_radius}(W_{rec})$

3 $W_{rec} \leftarrow \frac{\rho}{r} \cdot W_{rec}$

4 **return** W_{rec}

Effect of initialization on the temporal gradients



Effect of initialization on the rate of success

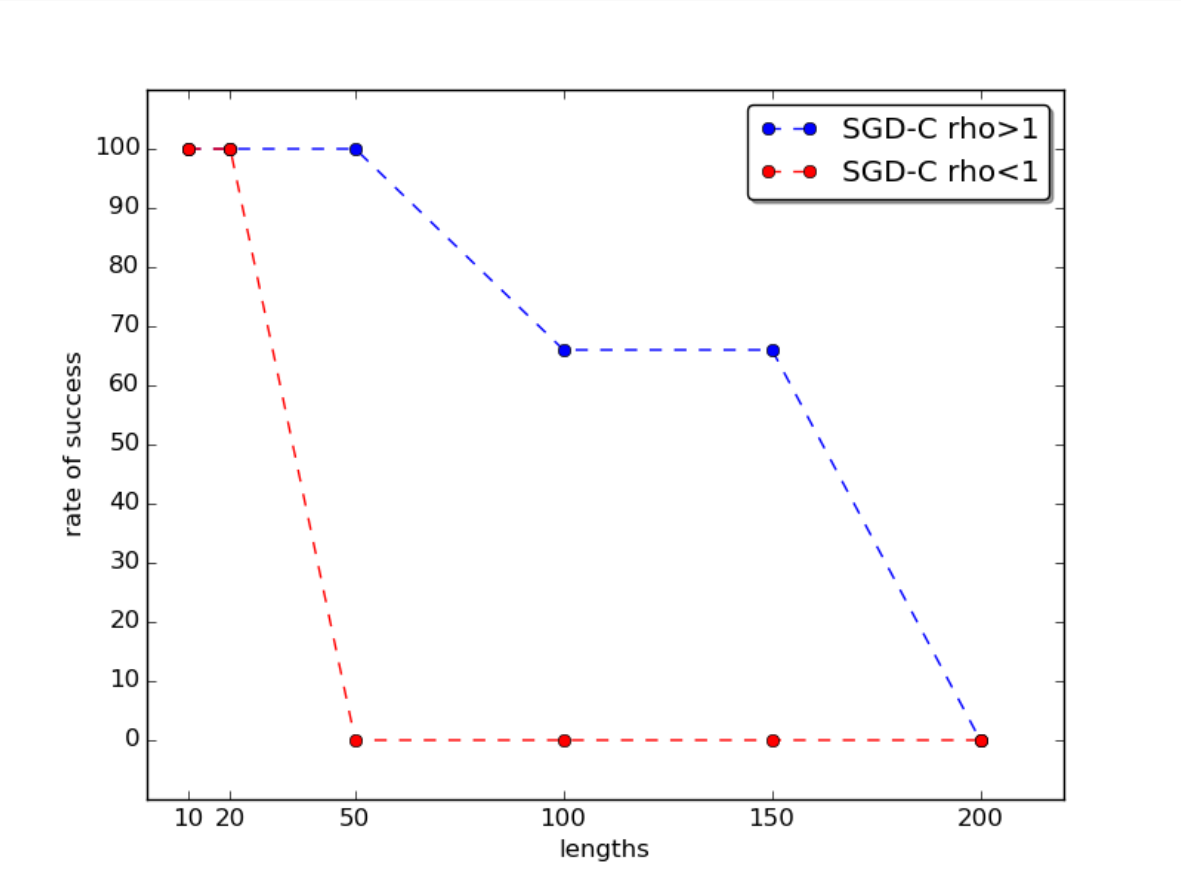


Figure 5: Rate of success (mean of 3 runs) for the temporal order task for various lengths with SGD modified with gradient clipping. In blue and red the results when W_{rec} is initialized with spectral radius bigger and smaller than one respectively.

A different descent direction

The idea is to use the structure of the gradient to compute a "descent" direction which does not suffer from the vanishing problem.

- Normalize the temporal gradients:

$$s_t(\boldsymbol{x}) = \frac{\nabla L_{|t}(\boldsymbol{x})}{\left\|\nabla L_{|t}(\boldsymbol{x})\right\|}. \tag{25}$$

- Combine the normalized gradients in a convex way:

$$s(\boldsymbol{x}) = \sum_{t=1}^{T} \beta_t \cdot s_t(\boldsymbol{x}). \tag{26}$$

  with $\sum_{t=1}^{T} \beta_t = 1, \beta_t > 0$ (randomly picked at each iteration).

- Introduce the gradient norm:

$$d(\boldsymbol{x}) = -\left\|\nabla L(\boldsymbol{x})\right\| \frac{s(\boldsymbol{x})}{\|s(\boldsymbol{x})\|}. \tag{27}$$

Algorithm 3: RNN training

Data:

$D = \{\langle \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \rangle\}$: training set

m: size of each mini-batch

μ: constant learning rate

τ: gradient clipping threshold

ρ: initial spectral radius

ψ threshold for the direction norm

$W_{rec}, W_{in}, W_{out} \sim \mathcal{N}(0, \sigma^2)$

$\boldsymbol{b}_{out}, \boldsymbol{b}_{rec} \leftarrow 0$

$r \leftarrow \text{spectral\_radius}(W_{rec})$

$W_{rec} \leftarrow \frac{\rho}{r} \cdot W_{rec}$

$\theta_0 = [W_{rec}, W_{in}, W_{out}, \boldsymbol{b}_{out}, \boldsymbol{b}_{rec}]$

while *stop criterion* **do**

  $I \leftarrow$ sample m training example $\in D$

  $\{\nabla_\theta L_{|t}\}_{t=1}^{T} \leftarrow \text{compute\_temporal\_gradients}(\theta_k, I)$

  $\boldsymbol{d}_k \leftarrow \text{simplex\_combination}(\{\nabla_\theta L_{|t}\})$

  if $\|\nabla_\theta L(\theta_k)\|_2 > \psi$ **then**

    $\boldsymbol{d}_k \leftarrow \nabla_\theta L(\theta_k)$

  end

  $$\alpha_k = \begin{cases} \mu & \text{if } \|\boldsymbol{d}_k\|_2 \leq \tau \\ \frac{\mu \cdot \tau}{\|\boldsymbol{d}_k\|_2} & \text{otherwise} \end{cases}$$

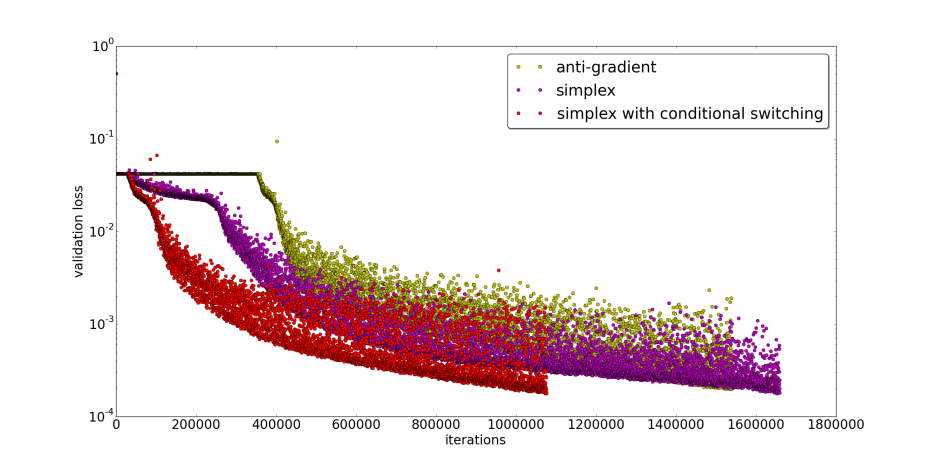
  $\theta_{k+1} \leftarrow \theta_k + \alpha_k \boldsymbol{d}_k$

  $k \leftarrow k + 1$

end

return θ_k

Effect of the simplex direction



(a) Loss (in log scale) for the addition task during training

	anti-gradient	simplex with conditional switching
addition	1807466	**1630666**
temporal order	2164800	**1010000**

(b) Number of iterations

Figure 6: Comparison between SGD using as descent direction the anti-gradient, the simplex direction and the simplex direction with conditional switching.

A real case: Lupus disease prediction

An example of patience record

	Visit 0	Visit 1	Visit 2	Visit 3	Visit 4
age	44.23	44.63	44.77	44.98	45.58
MyasteniaGravis	0	0	0	0	0
arthritis	1	0	1	1	0
c3level	119	96	85.42	76	76
c4level	9	7	6	6	6
hematological	0	0	6	6	6
skinrash	0	0	0	0	0
sledai2kInferred	12	2	2	2	0
...					
SDI	0	0	0	0	**1**

References I


S. Hochreiter and J. Schmidhuber.
Long short-term memory.
Neural Comput., 9(8):1735–1780, Nov. 1997.


J. Martens and I. Sutskever.
Training deep and recurrent networks with hessian-free optimization.
In G. Montavon, G. B. Orr, and K. Müller, editors, *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*, pages 479–535. Springer, 2012.


A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro.
Robust stochastic approximation approach to stochastic programming.
SIAM Journal on Optimization, 19(4):1574–1609, 2009.

References II


R. Pascanu, T. Mikolov, and Y. Bengio.
On the difficulty of training recurrent neural networks.
In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1310–1318, 2013.