

# Recurrent neural networks

A nice subtitle

---

Giulio Galvan

21st March 2016

Università degli studi di Firenze

# Introduction

---

# The model

## Def: RNN

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

$$\mathbf{y}^t \triangleq F(\mathbf{z}^t) \quad (1)$$

$$\mathbf{z}^t \triangleq W^{out} \cdot \mathbf{a}^t + \mathbf{b}^{out} \quad (2)$$

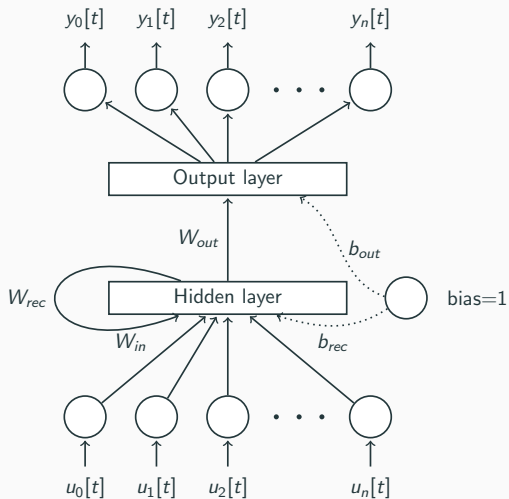
$$\mathbf{a}^t \triangleq W^{rec} \cdot \mathbf{h}^{t-1} + W^{in} \cdot \mathbf{u}^t + \mathbf{b}^{rec} \quad (3)$$

$$\mathbf{h}^t \triangleq \sigma(\mathbf{a}^t) \quad (4)$$

$$\mathbf{h}^0 \triangleq \mathbf{0}, \quad (5)$$

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

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



**Figure 1:** RNN model.

# The optimization problem

Given a dataset  $D$ :

$$D \triangleq \{\{\bar{\mathbf{u}}^{(i)}\}_{t=1,\dots,T}, \bar{\mathbf{u}}_t^{(i)} \in \mathbb{R}^p, \{\bar{\mathbf{y}}^{(i)}\}_{t=1,\dots,T}, \bar{\mathbf{y}}_t^{(i)} \in \mathbb{R}^o; i = 1, \dots, N\} \quad (6)$$

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

$$L_D(\mathbf{x}) \triangleq \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T L_t(\bar{\mathbf{y}}_t^{(i)}, \mathbf{y}_t^{(i)}(\mathbf{x})), \quad (7)$$

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

$$\min_{\mathbf{x}} L_D(\mathbf{x}) \quad (8)$$

## Some learning examples

- Regression: mean squared error, linear output

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

- Binary classification: hinge loss, linear output

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

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

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

# Stochastic gradient descent (SGD)

---

**Algorithm 1:** Stochastic gradient descent

---

**Data:**

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

$\mathbf{x}_0$ : candidate solution

$m$ : size of each mini-batch

**Result:**

$\mathbf{x}$ : solution

```
1  $\mathbf{x} \leftarrow \mathbf{x}_0$ 
2 while stop criterion do
3    $I \leftarrow$  select  $m$  training example  $\in D$ 
4    $\alpha \leftarrow$  compute learning rate
5    $\mathbf{x} \leftarrow \mathbf{x} - \alpha \sum_{i \in I} \nabla_{\mathbf{x}} L(\mathbf{x}; \langle \mathbf{u}^{(i)}, \mathbf{y}^{(i)} \rangle)$ 
6 end
```

---

# Convergence of SGD

- ▶ Nemirovski (2009)[4]: proof of convergence in the convex case
- ▶ there are no theoretical guarantees in the non-convex case
- ▶ in practice it always works: SGD is the standard framework in most of neural networks applications.



# The vanishing gradient problem

---

## A pathological problem example

An input sequence:

marker	0	1	0	...	0	1	0	0
value	0.3	<b>0.7</b>	0.1	...	0.2	<b>0.4</b>	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	<b>0.7</b>	0.1	...	0.2	<b>0.4</b>	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.

Let  $L_t(\bar{\mathbf{u}}, \bar{\mathbf{y}})$  the loss function for the time step  $t$  and  $g_t(\mathbf{x}) : \mathbb{R}$  be the function defined by

$$g_t(\mathbf{x}) \triangleq L_t(F(\mathbf{z}^t(\bar{\mathbf{u}}; \mathbf{x}), \bar{\mathbf{y}}_t))$$

and

$$g(\mathbf{x}) \triangleq \sum_{t=1}^T g_t(\mathbf{x})$$

$$\frac{\partial g}{\partial W_{rec}} = \sum_{t=1}^T \nabla L_t^T \cdot J(F) \cdot \frac{\partial \mathbf{z}^t}{\partial \mathbf{a}^t} \cdot \frac{\partial \mathbf{a}^t}{\partial W_{rec}} \quad (12)$$

$$= \sum_{t=1}^T \frac{\partial g_t}{\partial \mathbf{a}^t} \cdot \frac{\partial \mathbf{a}^t}{\partial W_{rec}} \quad (13)$$

Let's consider a single output unit  $u$ , and a weight  $w_{lj}$ , we have

$$\frac{\partial a_u^t}{\partial w_{lj}} = \sum_{k=1}^t \frac{\partial a_u^t}{\partial a_l^k} \cdot \frac{\partial a_l^k}{\partial w_{lj}} \quad (14)$$

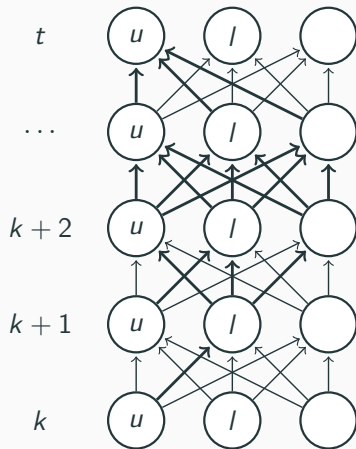
$$= \sum_{k=1}^t \delta_{lu}^{tk} \cdot \phi_j^{t-1} \quad (15)$$

where

$$\delta_{lu}^{tk} \triangleq \frac{\partial a_u^t}{\partial a_l^k}. \quad (16)$$

Let  $P(l)$  be the set of parents of neuron  $l$ , defined as the set of parents in the unfolded network.

$$\delta_{lu}^{tk} = \sum_{h \in P(l)} \delta_{hu}^{tk} \cdot \sigma'(a_h^{t-1}) \cdot w_{hl} \quad (17)$$



**Figure 2:** Nodes involved in  $\frac{\partial a_u^t}{\partial a_l^k}$ .

In matrix notation we have:

$$\frac{\partial \mathbf{a}^t}{\partial W^{rec}} = \sum_{k=1}^t \frac{\partial \mathbf{a}^t}{\partial \mathbf{a}^k} \cdot \frac{\partial^+ \mathbf{a}^k}{\partial W^{rec}} \quad (18)$$

$$\frac{\partial^+ \mathbf{a}^k}{\partial W_j^{rec}} = \begin{bmatrix} \phi_j^k & 0 & \dots & \dots & 0 \\ 0 & \phi_j^k & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & \dots & \dots & \phi_j^k \end{bmatrix} \quad (19)$$

$$\frac{\partial \mathbf{a}^t}{\partial \mathbf{a}^k} = \frac{\partial \mathbf{a}^t}{\partial \mathbf{a}^{k+1}} \cdot \text{diag}(\sigma'(\mathbf{a}^k)) \cdot W^{rec} \quad (20)$$

$$= \prod_{i=t-1}^k \text{diag}(\sigma'(\mathbf{a}^i)) \cdot W^{rec}. \quad (21)$$

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

## Vanishing gradient: a sufficient condition

$$\frac{\partial \mathbf{a}^t}{\partial \mathbf{a}^k} = \prod_{i=t-1}^k \text{diag}(\sigma'(\mathbf{a}^i)) \cdot W^{\text{rec}}. \quad (22)$$

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

$$W^{\text{rec}} = S \cdot D \cdot V^T \quad (23)$$

where  $S, V^T$  are squared orthogonal matrices and

$D \triangleq \text{diag}(\mu_1, \mu_2, \dots, \mu_r)$  is the diagonal matrix containing the singular values of  $W^{\text{rec}}$ . Hence:

$$\frac{\partial \mathbf{a}^t}{\partial \mathbf{a}^k} = \prod_{i=t-1}^k \text{diag}(\sigma'(\mathbf{a}^i)) \cdot S \cdot D \cdot V^T \quad (24)$$



Since  $U$  and  $V$  are orthogonal matrix, hence

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

and

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

we get

$$\left\| \frac{\partial \mathbf{a}^t}{\partial \mathbf{a}^k} \right\|_2 = \left\| \left( \prod_{i=t-1}^k diag(\sigma'(\mathbf{a}^i)) \cdot S \cdot D \cdot V^T \right) \right\|_2 \quad (25)$$

$$\leq (\sigma'_{max} \cdot \mu_{max})^{t-k-1} \quad (26)$$

# Solutions

---

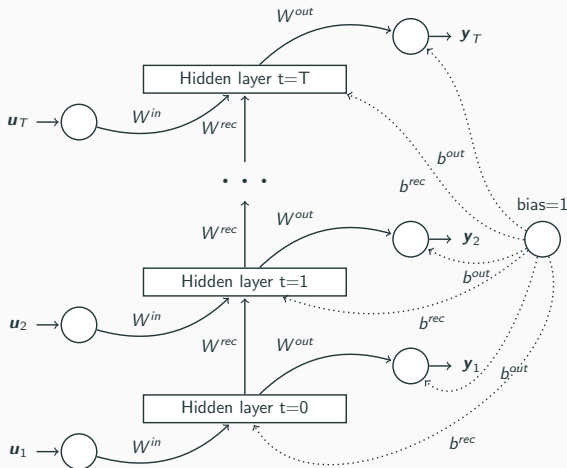
- ▶ Long short-term memory (LSTM). Hochreiter, Schmidhuber (1997)[1]
  - ▶ the network structure is modified with specialized "memory cells"
  - ▶ a truncated version of back-propagation is employed (but works also without it)
- ▶ Hessian-Free optimization (HF). Martens (2010) [3]
  - ▶ a second order method
  - ▶ a "cheap" approximation of the Hessian is employed
  - ▶ the quadratic sub-problem is solved through conjugate gradient + structural damping
- ▶ Pascanu, Bengio (2013) [5]
  - ▶ a first order method
  - ▶ uses a penalty to deal with the vanishing gradient problem

- ▶ Long short-term memory (LSTM). Hochreiter, Schmidhuber (1997)[1]
  - ▶ the network structure is modified with specialized "memory cells"
  - ▶ a truncated version of back-propagation is employed (but works also without it)
- ▶ Hessian-Free optimization (HF). Martens (2010) [3]
  - ▶ a second order method
  - ▶ a "cheap" approximation of the Hessian is employed
  - ▶ the quadratic sub-problem is solved through conjugate gradient + structural damping
- ▶ Pascanu, Bengio (2013) [5]
  - ▶ a first order method
  - ▶ uses a penalty to deal with the vanishing gradient problem

# Existent solutions

- ▶ Long short-term memory (LSTM). Hochreiter, Schmidhuber (1997)[1]
  - ▶ the network structure is modified with specialized "memory cells"
  - ▶ a truncated version of back-propagation is employed (but works also without it)
- ▶ Hessian-Free optimization (HF). Martens (2010) [3]
  - ▶ a second order method
  - ▶ a "cheap" approximation of the Hessian is employed
  - ▶ the quadratic sub-problem is solved through conjugate gradient + structural damping
- ▶ Pascanu, Bengio (2013) [5]
  - ▶ a first order method
  - ▶ uses a penalty to deal with the vanishing gradient problem

# Understanding the gradient structure: unfolding



**Figure 3:** Unfolding of a RNN

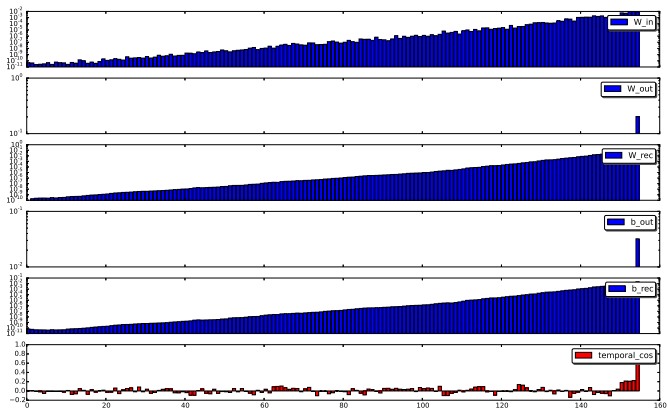
# Understanding the gradient structure

It's easy to see that

$$\nabla L(\mathbf{x}) = \sum_{t=1}^T \nabla L_{|t}(\mathbf{x}), \quad (27)$$

where  $\nabla L_{|t}$  is the gradient computed imagining to replicate all the variables for each time step and taking the derivatives w.r.t. the variables for step  $t$

# Temporal gradient norms: an illustration





## A new proposal

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 components:

$$g_t(\mathbf{x}) = \frac{\nabla L_t(\mathbf{x})}{\|\nabla L_t(\mathbf{x})\|}. \quad (28)$$

- combine the normalized gradients in a simplex:

$$g(\mathbf{x}) = \sum_{t=1}^T \beta_t \cdot g_t(\mathbf{x}). \quad (29)$$

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

- exploit the gradient norm:

$$d(\mathbf{x}) = \|\nabla L(\mathbf{x})\| \frac{g(\mathbf{x})}{\|g(\mathbf{x})\|}. \quad (30)$$

## Open issues

---

# Open Issues: Initialization

- ▶ Some tasks, like the XOR one, are still "unresolved" (even for the other approaches). They cannot be solved with most of the seeds used for the initialization
- ▶ it seems to be an **initialization** matter

Popular strategies for initialization are:

- ▶ "small random weights", usually drawn from gaussian or uniform distribution with zero mean (Pascanu).
- ▶ sparse initialization: only some weights are actually sampled from a distribution, the other are set to zero (HF).
- ▶ ESN-like initialization[2].

# Open Issues: Learning rate

- ▶ the **learning rate** is usually tuned by hand, there is no convergence theory for SGD in the non convex case
- ▶ a **gradient clipping** technique is often employed:

---

**Algorithm 2:** Gradient clipping

---

```
1  $\mathbf{g} \leftarrow \nabla_{\mathbf{x}} L$   
2 if  $\|\mathbf{g}\| \geq threshold$  then  
3    $\mathbf{g} \leftarrow \frac{threshold}{\|\mathbf{g}\|} \mathbf{g}$   
4 end
```

---

- ▶ some **momentum** or **averaging** technique often yield better convergence time, again tuned by hand





S. Hochreiter and J. Schmidhuber.

**Long short-term memory.**

*Neural Comput.*, 9(8):1735–1780, Nov. 1997.



M. Lukosevicius and H. Jaeger.

**Reservoir computing approaches to recurrent neural network training.**

*Computer Science Review*, 3(3):127–149, 2009.



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.



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.