Recurrent neural networks

Giulio Galvan

Introduction

The model

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

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

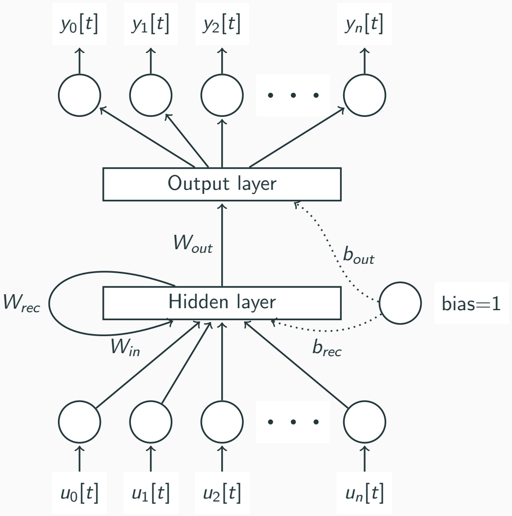


Figure 1: RNN model.

The optimization problem

Given a dataset D:

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

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

```
x ← x_0
while stop criterion do
    I ← select m training example ∈ D
    α ← compute learning rate
    x ← x − α Σ_{i∈I} ∇_x L(x; ⟨u^(i), y^(i)⟩)
end
```

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

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

Convergence of SDG

- 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	**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.

Gradient

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

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

and

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

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

$$= \sum_{t=1}^{T} \frac{\partial g_t}{\partial \boldsymbol{a}^t} \cdot \frac{\partial \boldsymbol{a}^t}{\partial W^{rec}} \tag{13}$$

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

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

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

where

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

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

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

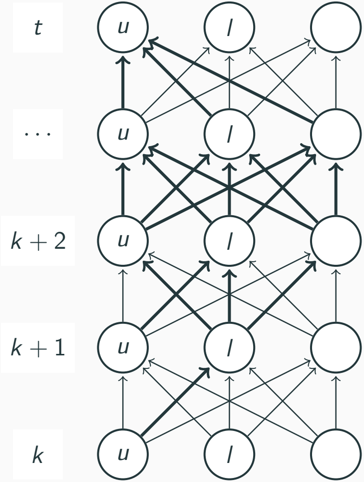


Figure 2: Nodes involved in $\frac{\partial a_u^t}{\partial a_l^k}$.

In matrix notation we have:

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

$$\frac{\partial^+ a^k}{\partial W_j^{rec}} = \begin{bmatrix} \phi_j^k & 0 & \cdots & \cdots & 0 \\ 0 & \phi_j^k & \cdots & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & \cdots & \phi_j^k \end{bmatrix} \tag{19}$$

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

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

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

Vanishing gradient: a sufficient condition

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

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

$$W^{rec} = S \cdot D \cdot V^T \tag{23}$$

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:

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

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

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

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

Solutions

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

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

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

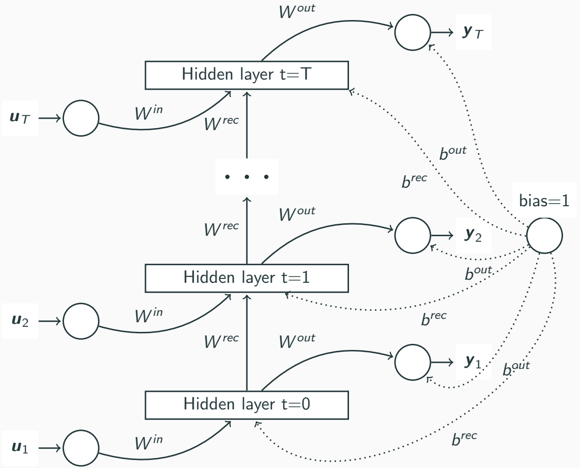


Figure 3: Unfolding of a RNN

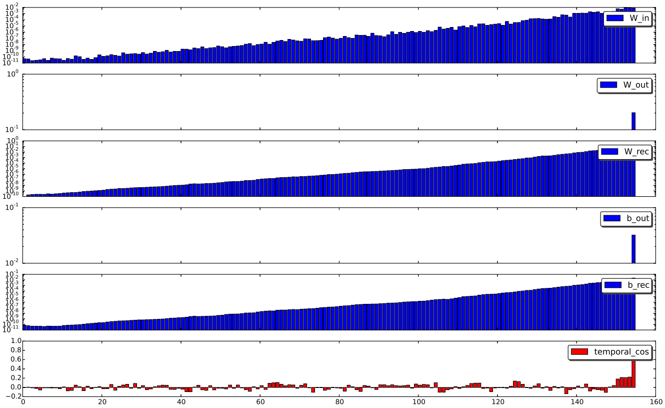
Understanding the gradient structure

It's easy to see that

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

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:

$$g_t(\boldsymbol{x}) = \frac{\nabla L_{|t}(\boldsymbol{x})}{\left\|\nabla L_{|t}(\boldsymbol{x})\right\|}. \tag{28}$$

- combine the normalized gradients in a simplex:

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

  with $\sum_{t=1}^{T} \beta_t = 1, \beta_t > 0$ (randomly picked at each iteration).
- exploit the gradient norm:

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

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

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{x}} L$

if $\|\boldsymbol{g}\| \geq \mathit{threshold}$ **then**

$\quad \boldsymbol{g} \leftarrow \frac{\mathit{threshold}}{\|\boldsymbol{g}\|} \boldsymbol{g}$

end

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

References I

- 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.

References II

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.