Recurrent neural networks

Giulio Galvan

The model

RNN

Given an input sequences $\{\boldsymbol{u}\}_{t=1,\dots,T}$, 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:

$$\boldsymbol{y}^t \triangleq F(W^{out} \cdot \boldsymbol{a}^t + \boldsymbol{b}^{out}) \tag{1}$$

$$\boldsymbol{a}^t \triangleq W^{rec} \cdot \boldsymbol{h}^{t-1} + W^{in} \cdot \boldsymbol{u}^t + \boldsymbol{b}^{rec} \tag{2}$$

$$\boldsymbol{h}^t \triangleq \sigma(\boldsymbol{a}^t) \tag{3}$$

$$\boldsymbol{h}^0 \triangleq \overrightarrow{0}, \tag{4}$$

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

The optimization problem

Given a dataset D:

$$D \triangleq \{\{\overline{\boldsymbol{u}}^{(i)}\}_{t=1,\dots,T}, \overline{\boldsymbol{u}}_t^{(i)} \in \mathbb{R}^p, \{\overline{\boldsymbol{y}}^{(i)}\}_{t=1,\dots,T}, \overline{\boldsymbol{y}}_t^{(i)} \in \mathbb{R}^o; i = 1, \dots, N\} \tag{5}$$

we define a loss function $L_D : \mathbb{R}^N \to \mathbb{R}_{\geq 0}$ over D as

$$L_D(\boldsymbol{x}) \triangleq \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} L_t(\overline{\boldsymbol{y}}_t^{(i)}, \boldsymbol{y}_t^{(i)}(\boldsymbol{x})), \tag{6}$$

where $L_t(\cdot, \cdot)$ is an arbitrary loss function for the time step t. The problem is

$$\min_{\boldsymbol{x} \in \mathbb{R}^N} L_D(\boldsymbol{x}) \tag{7}$$

Stochastic gradient descent (SGD)

SGD is the standard framework in most of the applications.

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 minibatch

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

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 it's 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 it's long time dependencies.

A pathological problem example

An input sequence:

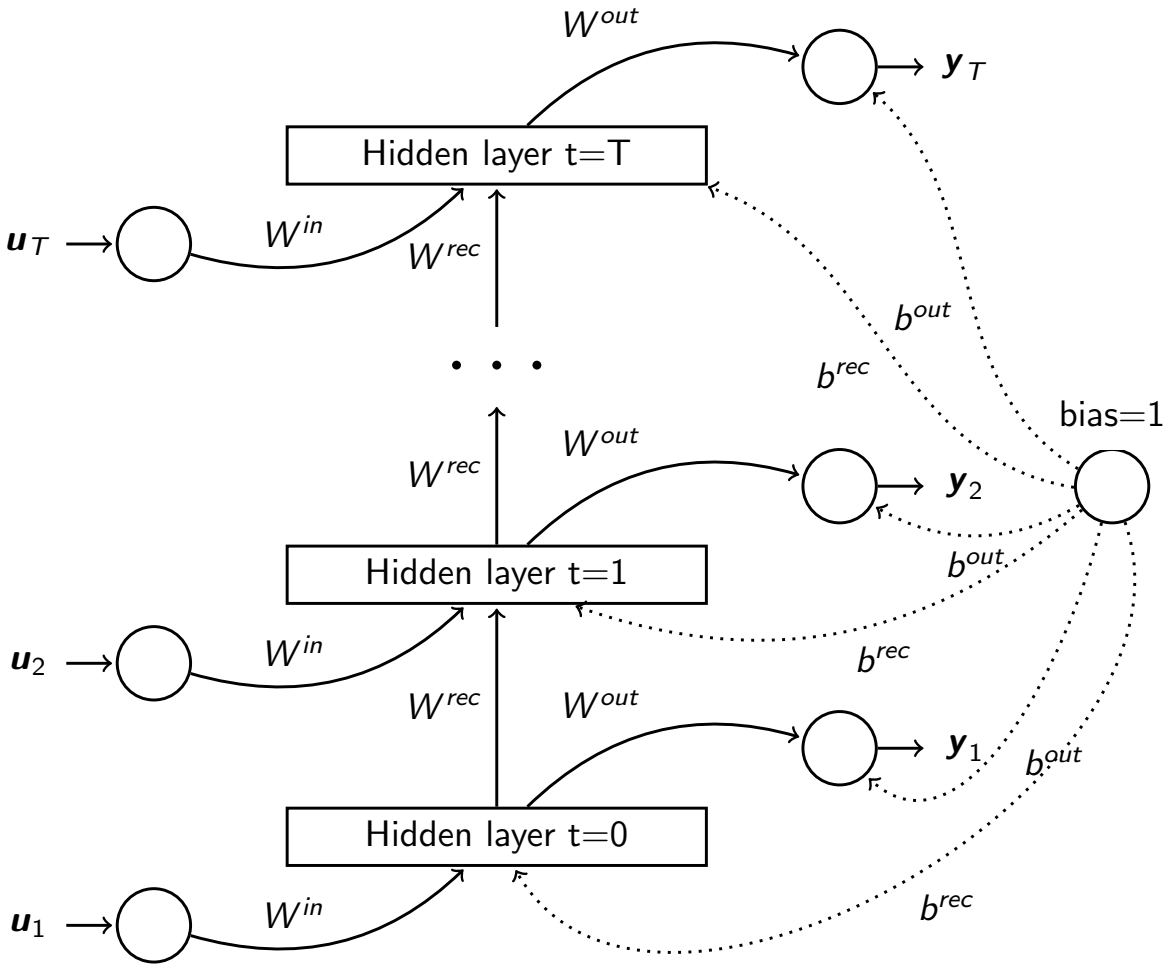
marker	0	1	0	...	0	1	0	0
value	0.3	**0.7**	0.1	...	0.2	**0.4**	0.6	0.9

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

Why is this a difficult problem?

Because of it's long time dependencies.

Unfolding


W^{out}
y_T
Hidden layer t=T
u_T
W^{in}
W^{rec}
b^{out}
b^{rec}
bias=1
W^{rec}
W^{out}
y_2
Hidden layer t=1
b^{out}
u_2
W^{in}
b^{rec}
W^{rec}
W^{out}
y_1
b^{out}
Hidden layer t=0
u_1
W^{in}
b^{rec}

Understanding the gradient structure

Simply applying the chain rule it's easy to see that

$$\nabla L(\boldsymbol{x}) = \sum_{t=1}^{T} \nabla L_{|t}(\boldsymbol{x}), \tag{8}$$

where $\nabla L_{|t}$ is ...

Vanishing gradient: an upper bound

$$\frac{\partial \boldsymbol{a}^t}{\partial \boldsymbol{a}^k} = \prod_{i=t-1}^{k} diag(\sigma'(\boldsymbol{a}^i)) \cdot W^{rec}. \tag{9}$$

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

$$W^{rec} = S \cdot D \cdot V^T \tag{10}$$

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:

$$\frac{\partial \boldsymbol{a}^t}{\partial \boldsymbol{a}^k} = \prod_{i=t-1}^{k} diag(\sigma'(\boldsymbol{a}^i)) \cdot S \cdot D \cdot V^T \tag{11}$$

Since U and V are orthogonal matrix, hence

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

and

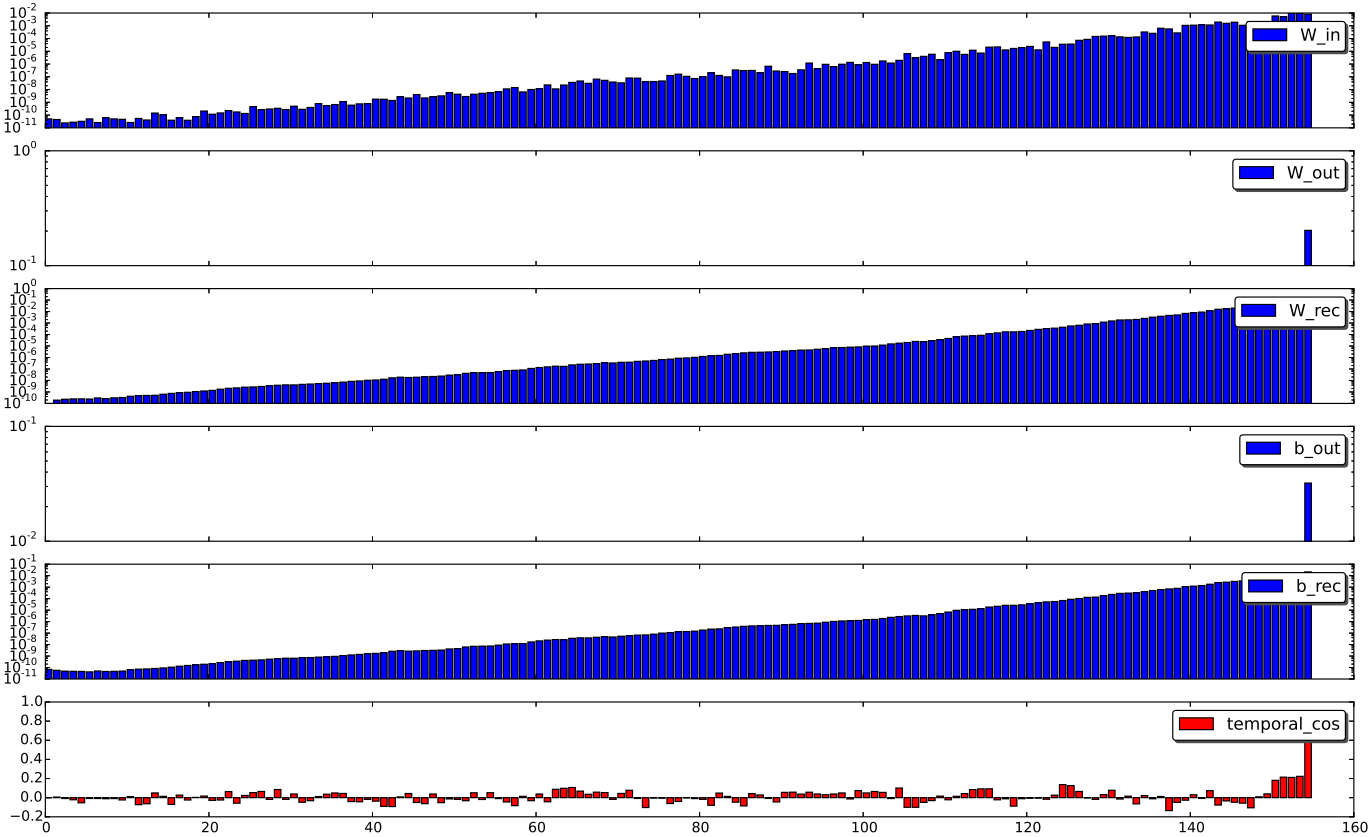
$$\|diag(\lambda_1, \lambda_2, ..., \lambda_r)\|_2 \leq \lambda_{max},$$

we get

$$\left\|\frac{\partial \boldsymbol{a}^t}{\partial \boldsymbol{a}^k}\right\|_2 = \left\|\left(\prod_{i=t-1}^{k} diag(\sigma'(\boldsymbol{a}^i)) \cdot S \cdot D \cdot V^T\right)\right\|_2 \tag{12}$$

$$\leq \left(\sigma'_{max} \cdot \mu_{max}\right)^{t-k-1} \tag{13}$$

Temporal gradient norms: an illustration



Existent solutions

- Long short-term memory (LSTM). Hochreiter, Schmidhuber (1997)
  - the network structure is modified with specialized "memory cells"
  - a truncated version of back-propagation is employed
- Hessian-Free optimization (HF). Martens (2010)
  - 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)
  - a first order method
  - uses a penalty to deal with the vanishing gradient problem

Existent solutions

- Long short-term memory (LSTM). Hochreiter, Schmidhuber (1997)
  - the network structure is modified with specialized "memory cells"
  - a truncated version of back-propagation is employed
- Hessian-Free optimization (HF). Martens (2010)
  - 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)
  - a first order method
  - uses a penalty to deal with the vanishing gradient problem

Existent solutions

- Long short-term memory (LSTM). Hochreiter, Schmidhuber (1997)
  - the network structure is modified with specialized "memory cells"
  - a truncated version of back-propagation is employed
- Hessian-Free optimization (HF). Martens (2010)
  - 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)
  - a first order method
  - uses a penalty to deal with the vanishing gradient problem

Existent solutions

- Long short-term memory (LSTM). Hochreiter, Schmidhuber (1997)
  - the network structure is modified with specialized "memory cells"
  - a truncated version of back-propagation is employed
- Hessian-Free optimization (HF). Martens (2010)
  - 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)
  - a first order method
  - uses a penalty to deal with the vanishing gradient problem

A new proposal

- use the structure of the gradient to compute a descent direction which does not suffer from the vanishing gradient problem
- normalize the temporal components

$$d(\boldsymbol{x}) = \sum_{t=1}^{T} \frac{\nabla L_{|t}(\boldsymbol{x})}{\left\|\nabla L_{|t}(\boldsymbol{x})\right\|} \tag{14}$$

- add some randomness for robustness:

$$d(\boldsymbol{x}) = \sum_{t=1}^{T} \beta_t \frac{\nabla L_{|t}(\boldsymbol{x})}{\left\|\nabla L_{|t}(\boldsymbol{x})\right\|}, \tag{15}$$

with $\sum_{t=1}^{T} \beta_t = 1, \beta_t > 0$

Open Issues: Initialization

- Some tasks, like the XOR one, are still "unresolved" (even for the other approaches). They cannot be solved with high probability (varying the seed)
- it seems to be an **initialization** matter

Popular strategies for initialization are:

- "small random weights", usually drawn from Gaussian distribution with zero mean.
- "reservoir initialization"
- sparse gaussian initialization, only some weights are actually sampled from a Gaussian the other are zero. (Used by HF)

Open Issues: Learning rate

- the **learning rate** is usually tuned by hand, there is no convergence theory for SGD in the non convex case
- some **momentum** or **averaging** technique often yield better convergence time, again tuned by hand

The end