

92586 Computational Linguistics

Lesson 18. Recurrent Neural Networks

Alberto Barrón-Cedeño

Alma Mater Studiorum-Università di Bologna
a.barron@unibo.it @albarron_

30/04/2021



Previously

► CNNs for text

Table of Contents

Introduction

Keeping the past in mind

RNNs in Keras

Chapter 8 of Lane et al. (2019)

Introduction

Introduction

CNNs

- ▶ Good for analysing *full* texts (~sentences)
- ▶ Words tending to appear close to each other are spotted and play a role
- ▶ Longer relationships —farther than $[3, 4]$ words are ignored

What is missing?

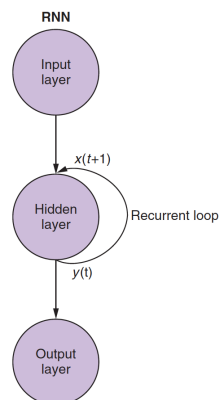
- ▶ Keeping track of what happened long ago
- ▶ Memory
- ▶ Language is **not an image** —no snapshots
- ▶ Language is a **sequence**; both text and speech

Keeping the past in mind

Remembering the Past

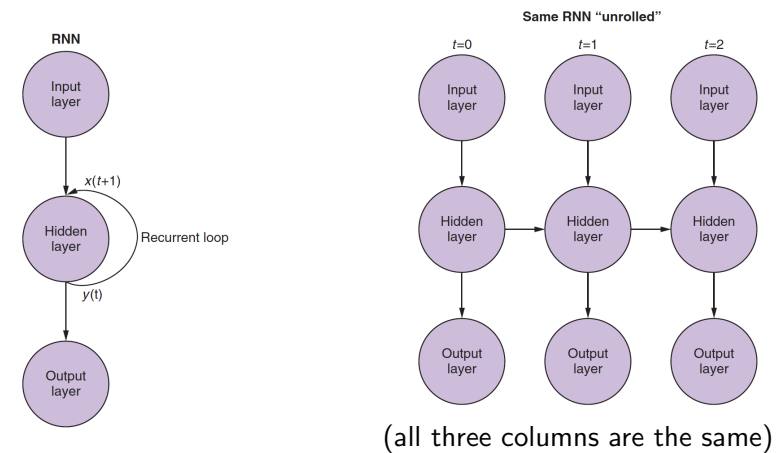
$w_0 w_1 w_2 w_3 \dots w_{t-1} w_t w_{t+1}$

- ▶ To understand a text at time t , we need to keep in mind what happened at time $t - k$
- ▶ Recurrent neural nets come into play
- ▶ **RNNs** combine what happened before with what happens now



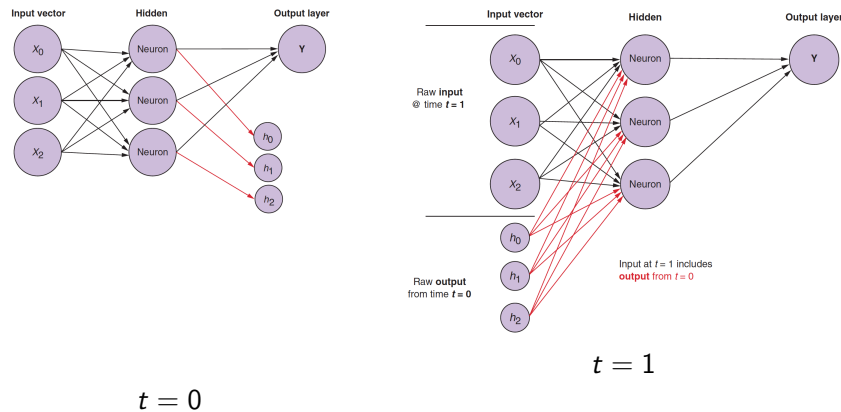
(Lane et al., 2019, p. 250)

Full feed-forward networks that consider their own output



(Lane et al., 2019, p. 252)

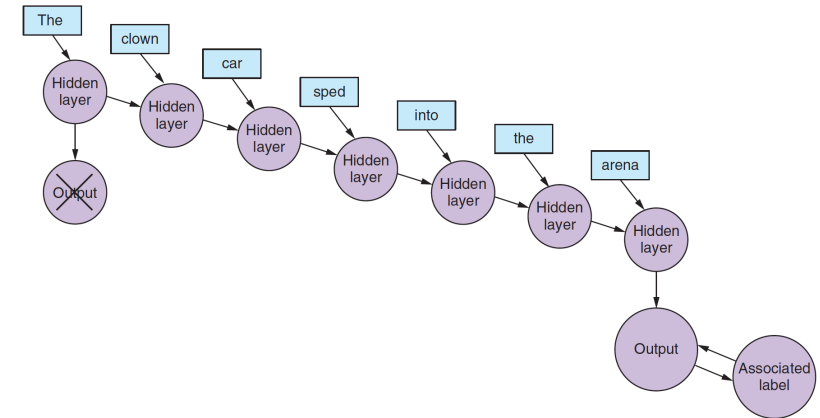
Zooming into the unrolled RNN: t and $t + 1$



- The red arrows are just *standard* connections, with weights
- Now we can feed the text, one word at a time

(Lane et al., 2019, p. 252–253)

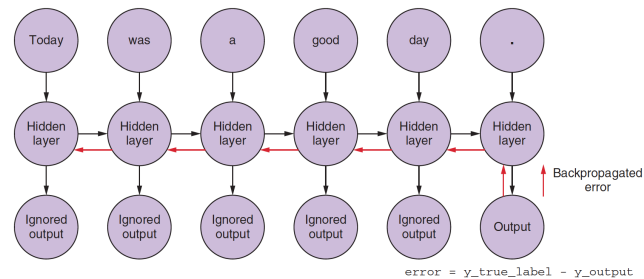
“Multiple inputs, one output”



- No more length constraints (although we have to be reasonable)
- No more a bunch of snapshots; there is a sense of time

(Lane et al., 2019, p. 254)

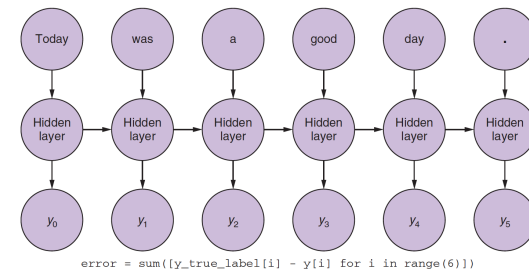
Backpropagation through Time: the “Vanilla” Way



- All intermediate outputs are ignored; the loss is computed at the end
- The same chain rule is applied to do backpropagation; but this time it heads to “the past”
- The weight corrections are calculated for each t
- The combined updates are applied **only** until reaching $t = 0$

(Lane et al., 2019, p. 256)

Backpropagation through Time: the Better Way



- We compute the loss combining all intermediate outputs
- The weight corrections are still additive: the update is applied until
 1. computing all errors and
 2. reaching back to the weight adjustments in $t = 0$

Let us see

(Lane et al., 2019, p. 258)

RNNs in Keras

RNN in Keras: what we have so far

We have setup a simple recurrent neural network

- ▶ The input sequences have fixed length: 400 tokens (each 300D)
- ▶ Our recurrent layer contains 50 neurons
- ▶ The output will be 400×50 :
 - ▶ 400 elements
 - ▶ one 50D vector each

`return_sequences=True`

True return the network value at each t : 400 50D vectors

False return a single 50D vector (default)

True → **this is why we are padding**

 Let us see

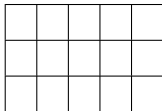
RNN in Keras: further details

- ▶ A Dense layer expects a *flat* vector

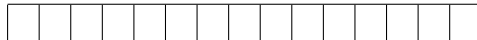
`model.add(Flatten())`

5×3


1×15



+ Flatten →



- ▶ In our case: $400 \times 50 \rightarrow 1 \times 20,000$

 Let us see

Example derived from <https://stackoverflow.com/questions/43237124/role-of-flatten-in-keras>

Some parameters are “free”

`embedding_dims` comes from the embedding space; hard to change, but possible: other embeddings, 1-hot

`num_neurons` kind of arbitrary; can be changed

`maxlen` kind of arbitrary; can be changed (or neglected)

`batch_size` bigger → faster (higher local minimum risk)

`epochs` trivial to increase (don't start from scratch each time)

 Let us see

Important: unless you have access to HPC, don't go bananas when exploring parameters (and perhaps even in that case)

Try some sensitive configurations and keep track of all the settings and outputs

References

Lane, H., C. Howard, and H. Hapkem
2019. *Natural Language Processing in Action*. Shelter Island,
NY: Manning Publication Co.