Barcelona Supercomputing Center
Centro Nacional de Supercomputación

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

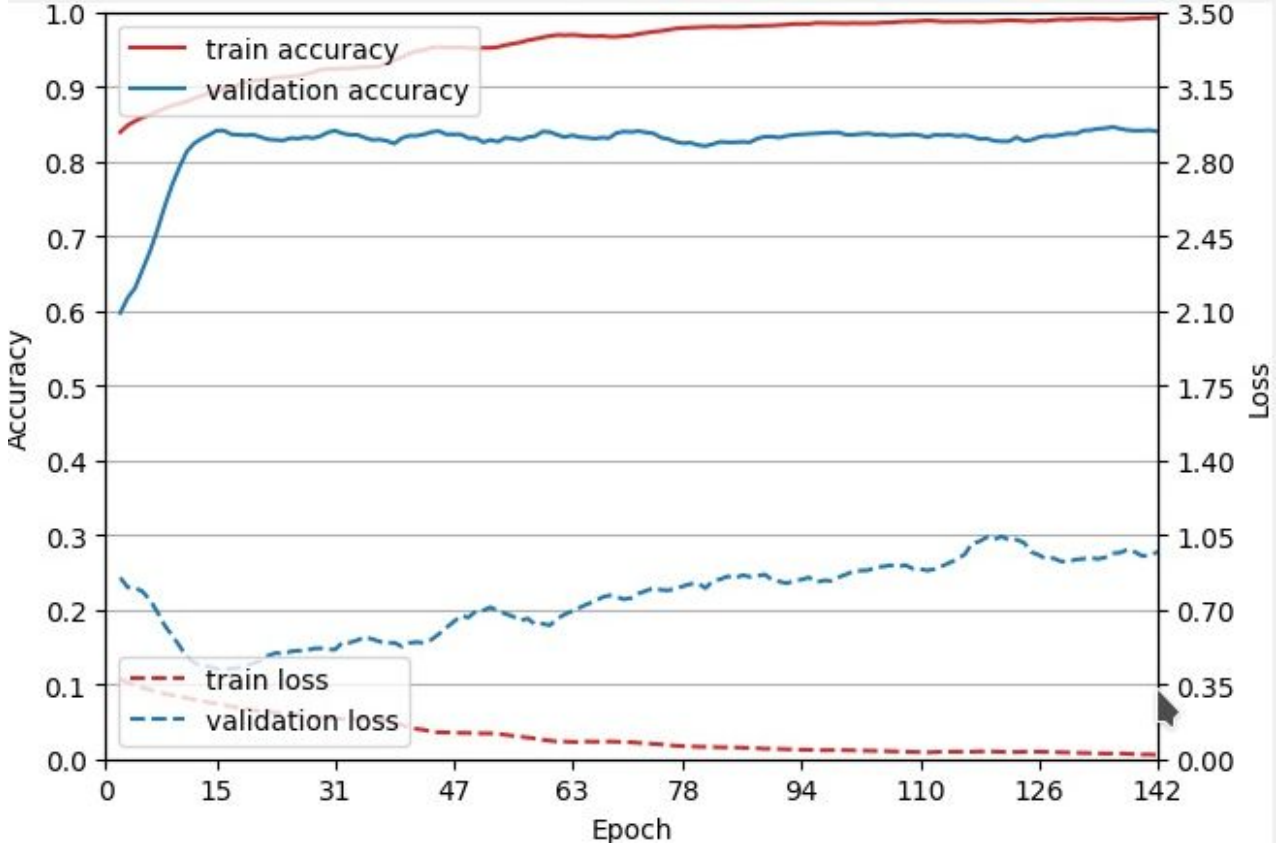
Deep Learning - MAI

before starting... a Horror Movie

Dario Garcia Gasulla
dario.garcia@bsc.es

Everything is fine

- Loss/Acc plot
- Cats vs Dogs classif.
- 1K train samples/class
- Epoch ≲ 16 = UF
- Epoch 16 ≃ Fit
- Epoch ≳ 16 OF



But wait

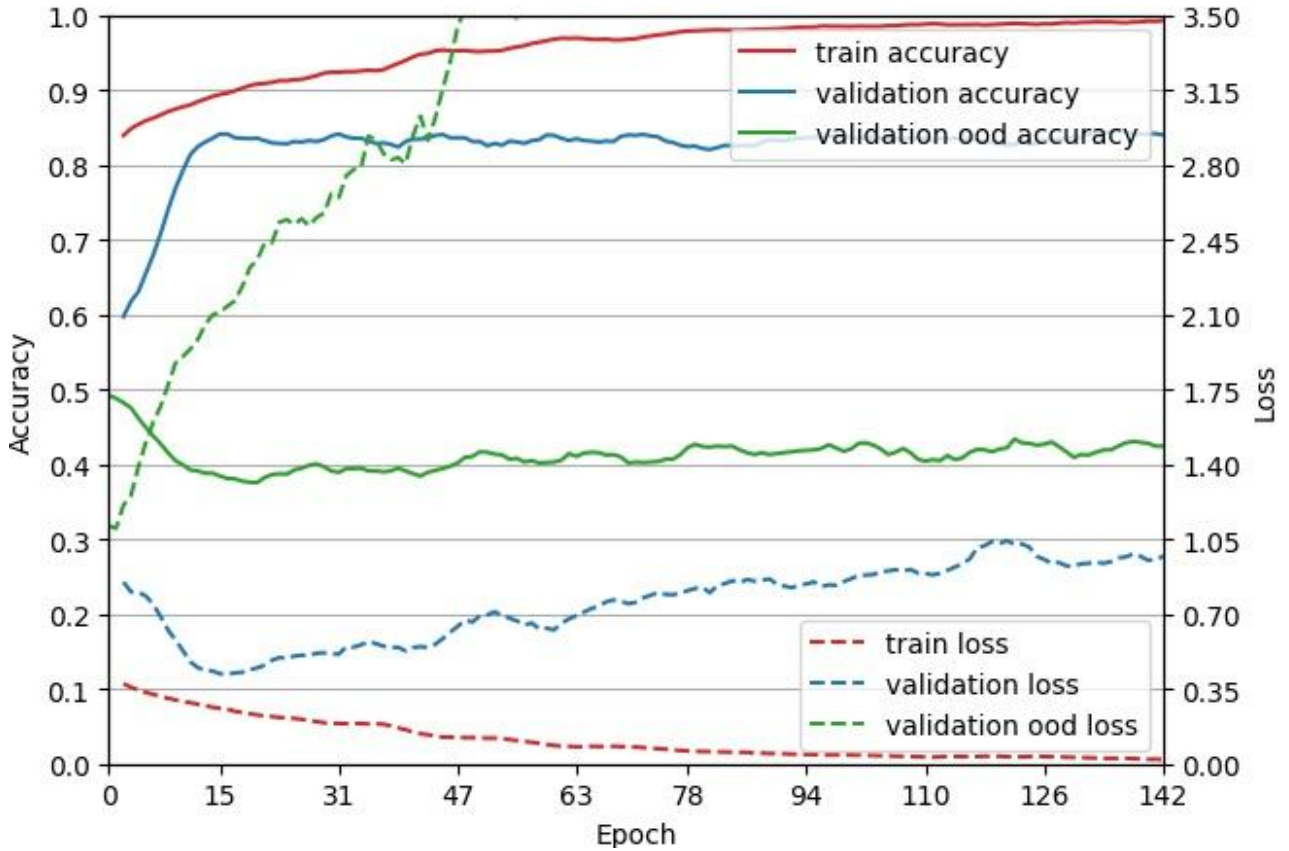
- **Cats** with 48 **indoor** contexts:
  - vase, laptop, rug, bowl, computer mouse, sink, shelf, blanket, carpet, desk, picture, bottle, bookshelf, lamp, suitcase, pillow, food, toy...
- **Dogs** with 27 **outdoor** contexts:
  - house, surfboard, car, fence, cow, trash can, trees, fire hydrant, bench, snow, flag, skateboard, helmet, water, sand, horse, frisbee...
- What happens if we test with OOD?
  - Cats (+7 outdoor objects)
  - Dogs (+27 indoor objects)

What is going on?

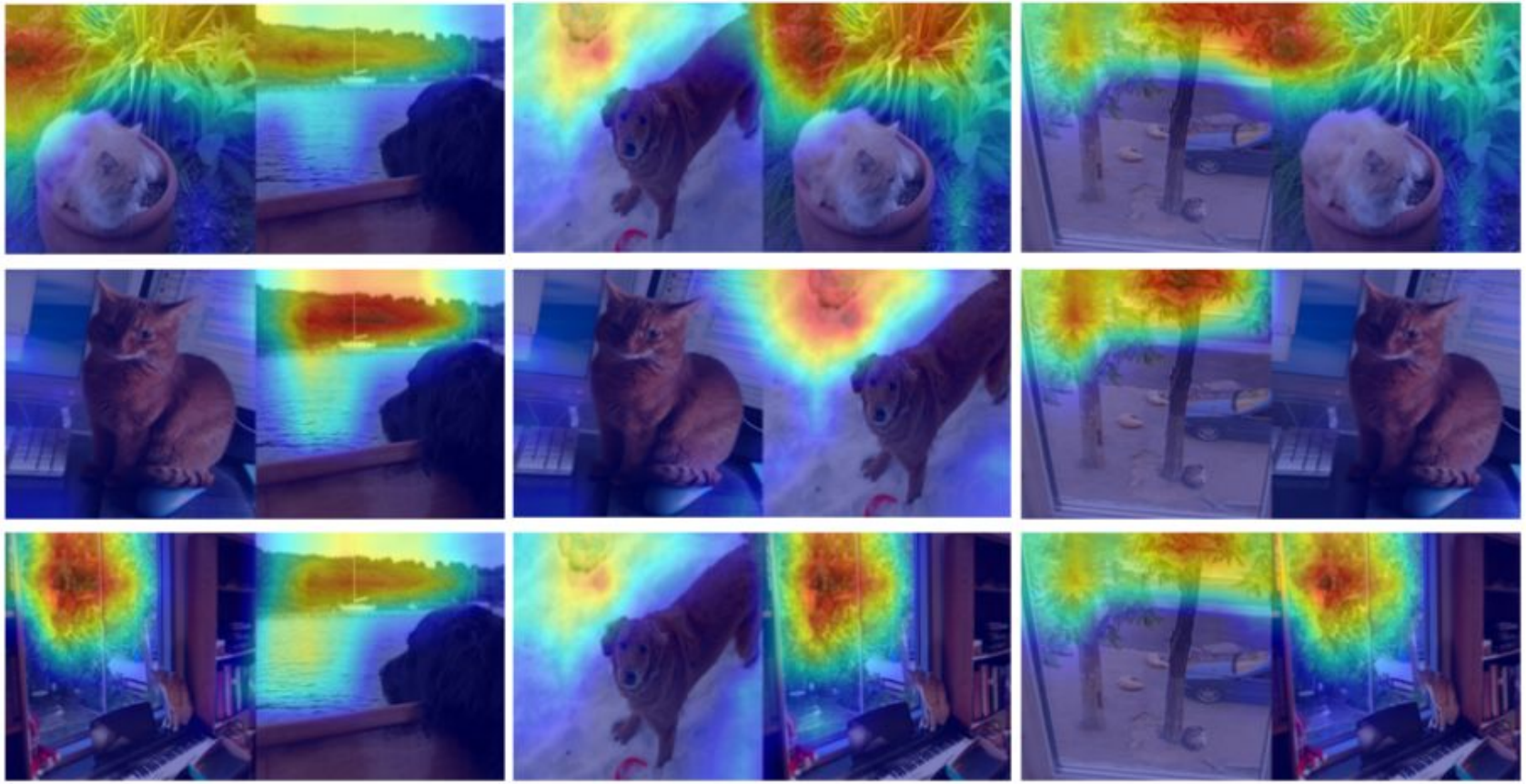
- **Worst dogs & Worst cats**

OMG

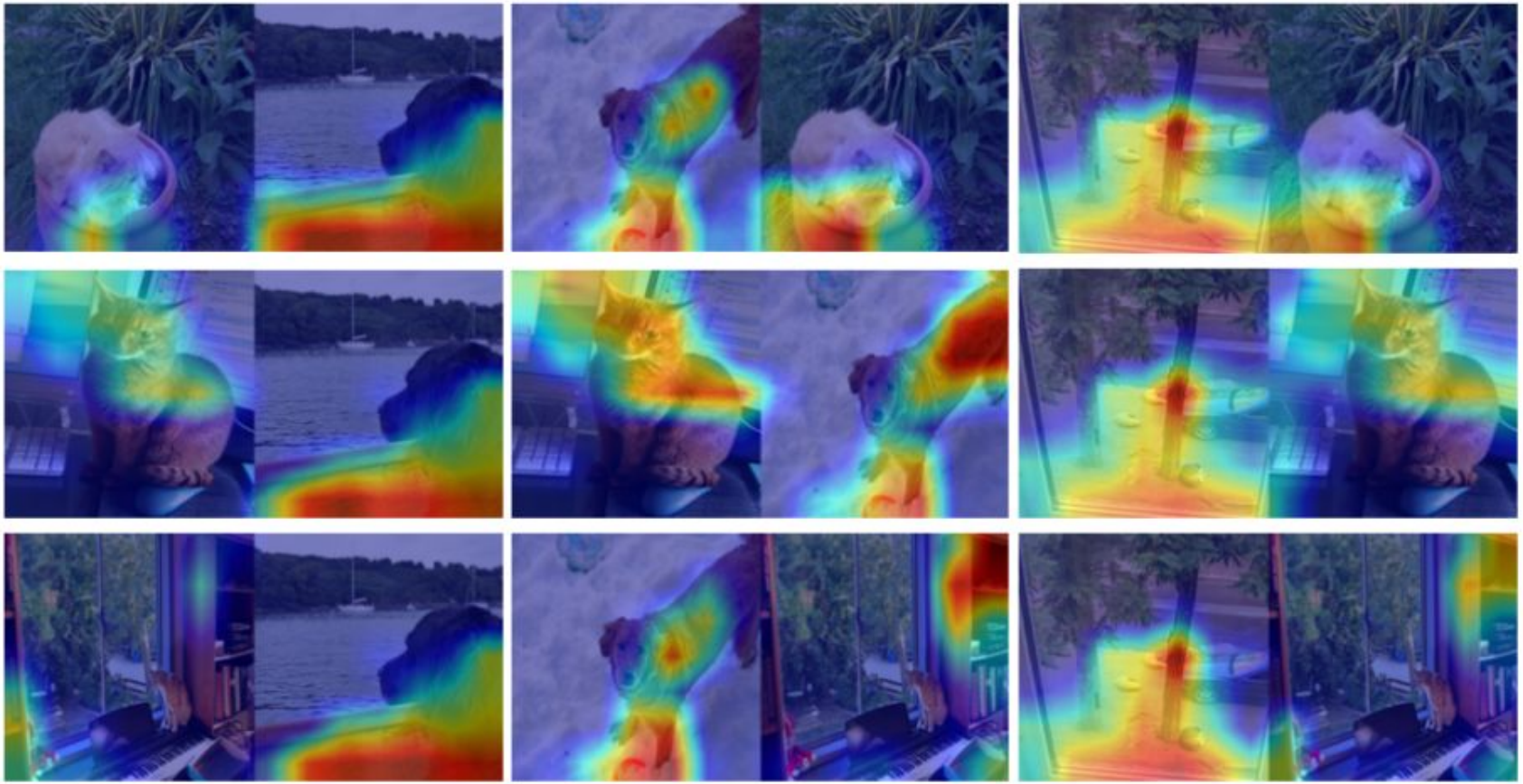
- Worst than random
- Cat? What's a *cat*?
- Beware of the bias!
- Trust no model



The target (dog)



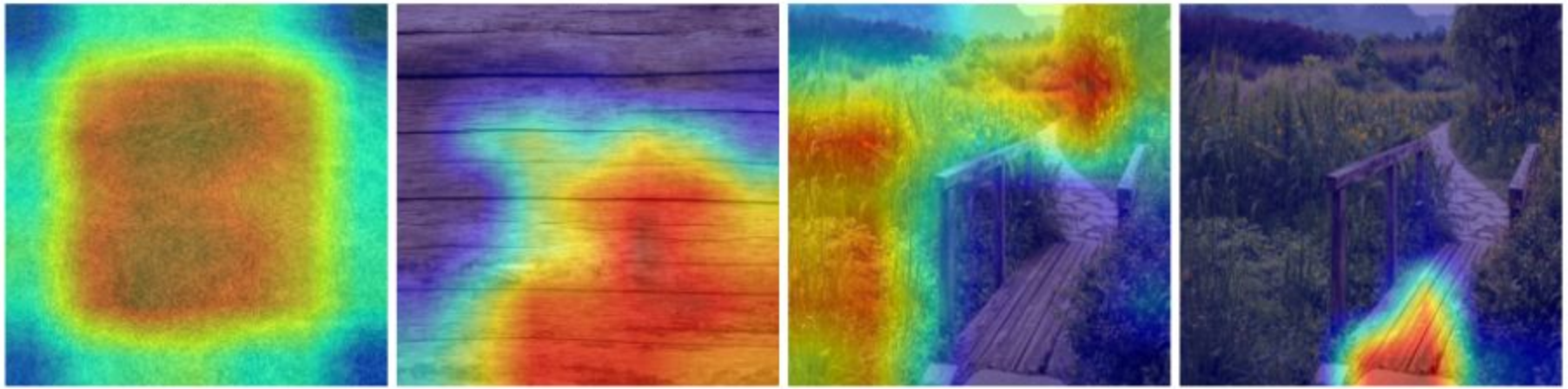
The target (cat)



The End

[55,56]

The End



99.98% Dog

96.30% Cat

99.40% Dog

[55,56]

Deep Learning - MAI

Theory - RNNs

Dario Garcia Gasulla
dario.garcia@bsc.es

Context

Vanilla RNNs

Advanced RNNs

RNNs extensions

Context

Dario Garcia Gasulla
dario.garcia@bsc.es

The Sequence

- Fully-connected and CNNs* have fixed inputs and outputs
- What if we have a variable shape input? Or output? And streams?

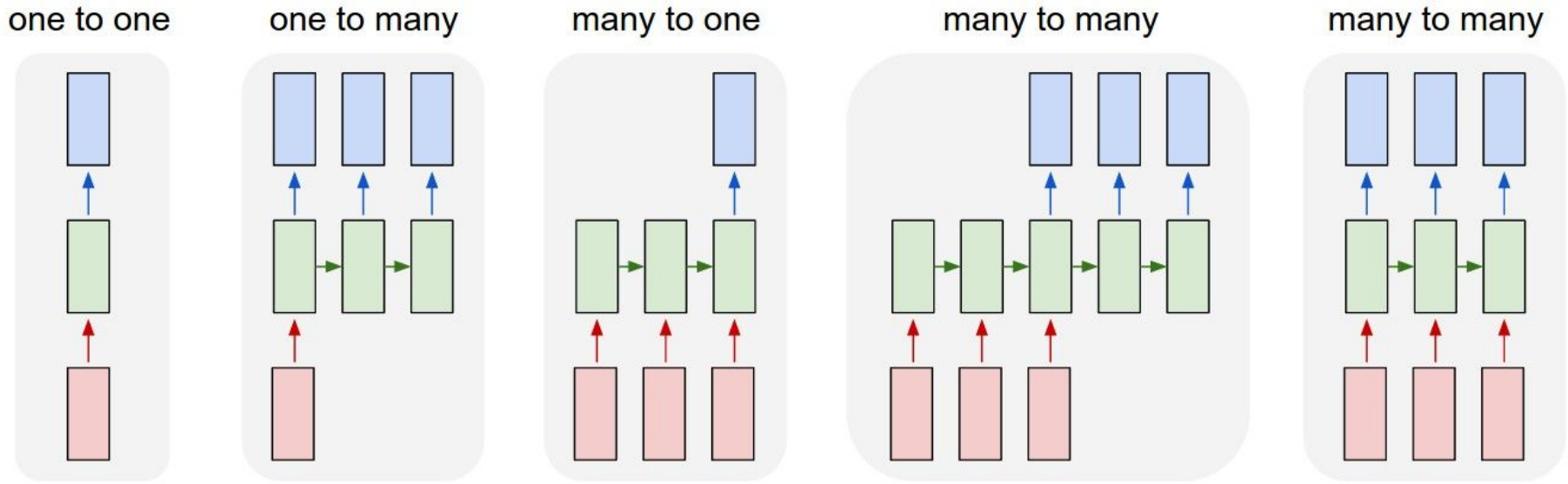
- Given a sequence (Text, video, audio, signal, bioinformatics...)
  - Learn the relations between the symbols that compose it, respecting its directionality

** CNNs tolerate a certain amount of inputs/outputs size variance thanks to convolution*

What do we want

- Sequences can be processed using traditional methods (e.g., sliding windows), but sequences need to have the same length
- We want to...
  - process sequences of arbitrary length
  - represent & exploit temporal dimension
  - have flexibility in input/output type

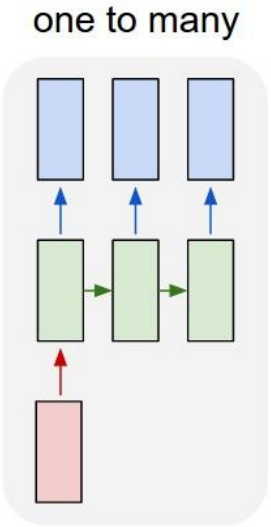
How do we want it



Fixed Input **Fixed** Output	**Fixed** Input *Sequence* Output	*Sequence* Input **Fixed** Output	*Sequence* Input *Sequence* Output	*Sequence* Input *Sequence* Output
Image Class.	Image Caption	Image Retrieval Sentiment Analysis	Translation	Frame classification Real time translation (synced)

One to many

- Image captioning (image to text)



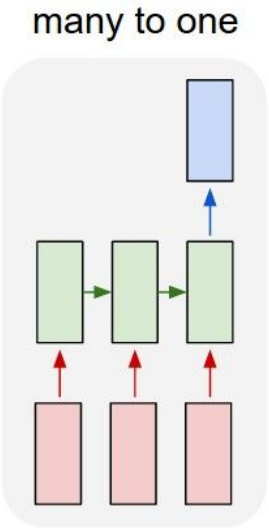
"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

Many to one

- Sentiment analysis (text to category)
  - My flight was just delayed, s**t ⇒ Negative
  - We arrived on time, yeehaaa! ⇒ Positive
  - Another day, another flight ⇒ Neutral

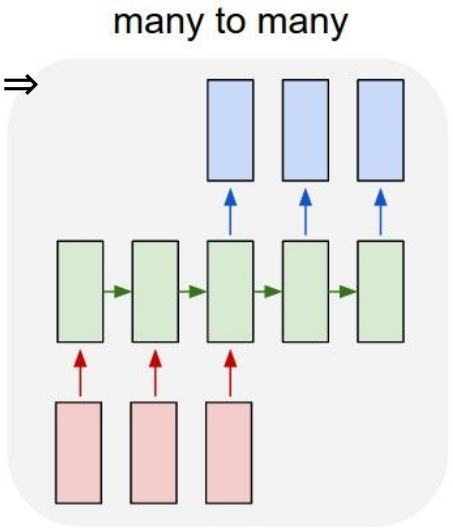


- Image retrieval (text to image)
  - Search engines

- Video labeling (images to category), images to image, ...

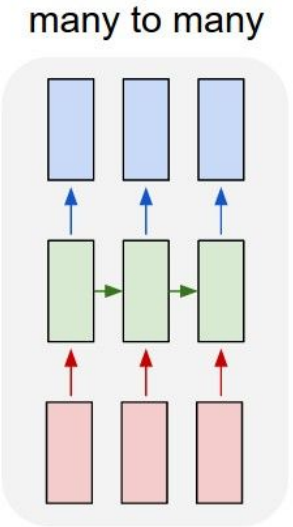
Sequence to Sequence

- Automatic translation
  - [How, many, programmers, for, changing, a,lightbulb,?] ⇒
  - [Wie, viele, Programmierer, zum, Wechseln, einer,Gl ühbirne,?] ⇒
  - [Combien, de, programmeurs, pour, changer, une,ampoule,?] ⇒
  - [¿,Cuantos, programadores, para, cambiar,una,bombilla,?] ⇒
  - [Zenbat, bonbilla, bat, aldatzeko,programatzaileak,?]



Synced Sequence

- Frame classification
- Real-time translation
- No full input info makes it harder

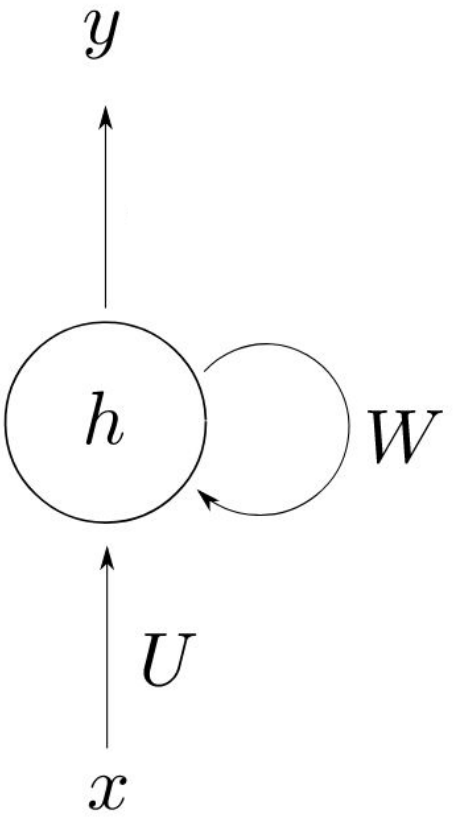


Vanilla RNNs

Dario Garcia Gasulla
dario.garcia@bsc.es

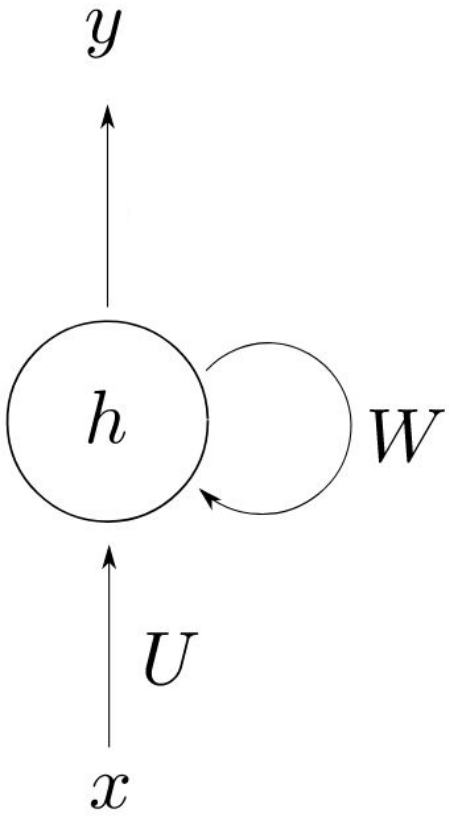
What makes a RNN?

- Recurrent Neural Networks are feed-forward networks with edges that span adjacent time steps (recurrent edges)
- On each step, a neuron receives inputs from data (x, as usual) and from previous time steps (h, history)
- In other words, a given time step influences the next one (and by transitivity, all following ones)
- RNNs are universal function approximators (Turing Complete)

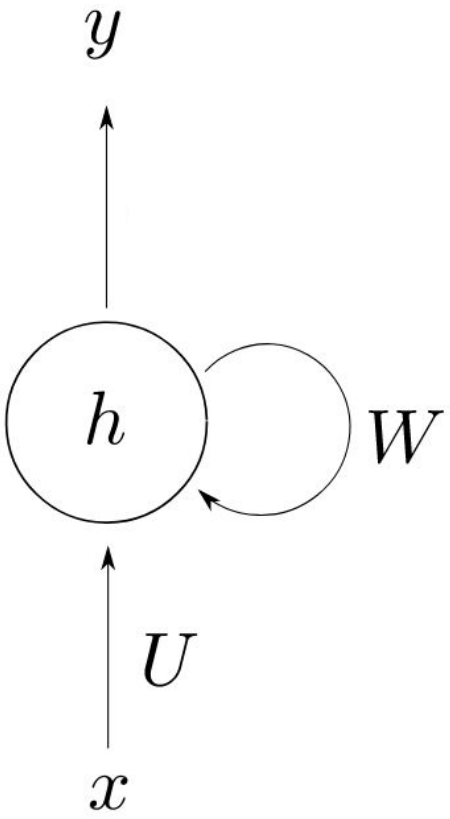


Who is who

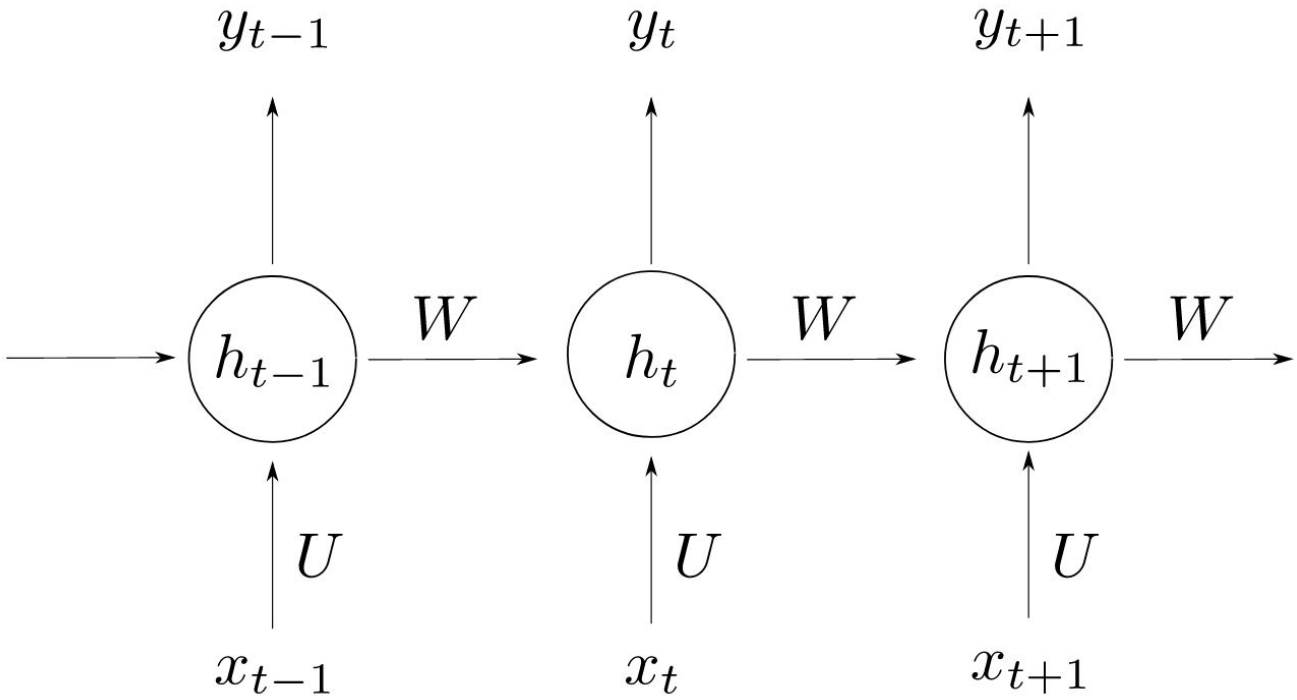
- RNNs Input (**x**) is a vector of values for time t
- The hidden node (**h**) stores the state
- Weights are shared through time (one weights, all time!)
- Each step the computation uses the previous step
  - $h^{(t+1)}=f(h^{(t)},x_{t+1};\theta) = f(f(h^{(t-1)},x_t;\theta),x_{t+1};\theta) =\cdots$
- RNN are a deep network that stacks layers through time (instead of stacking consecutive layers)



Inference: Before unrolling



Inference: After unrolling



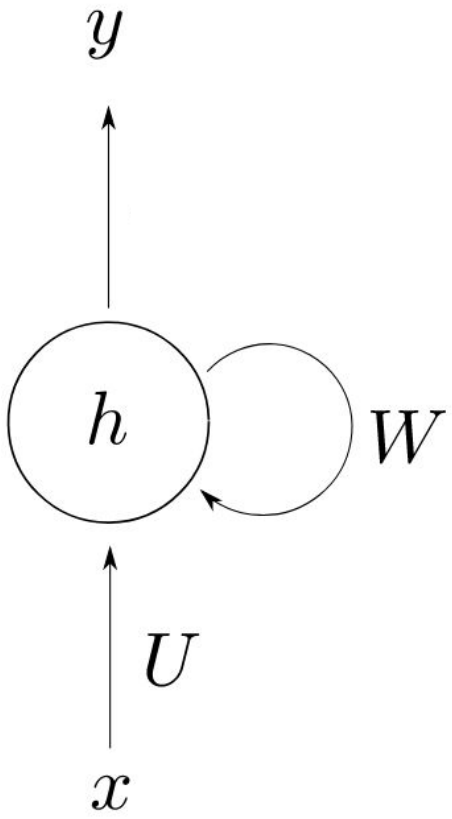
The computation

The past

The present

$$a^{(t)} = b + W \cdot h^{(t-1)} + U \cdot x^{(t)}$$

$$h^{(t)} = tanh(a^{(t)})$$



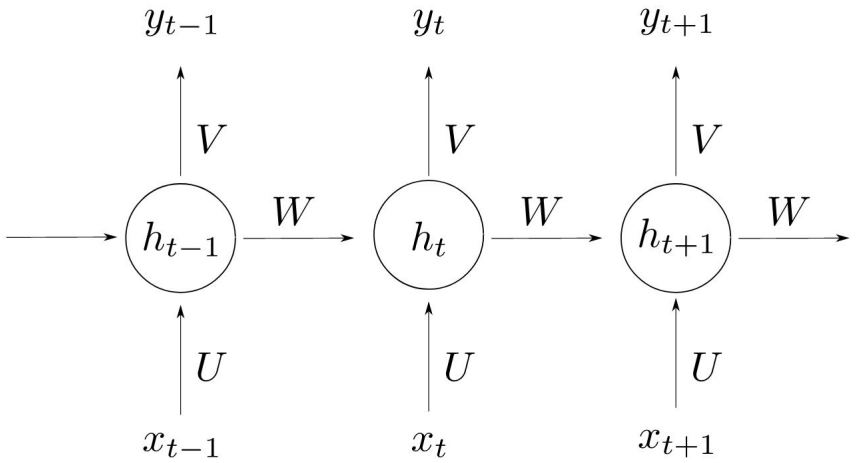
Practical Tips I

The power of RNNs lies in the unrolling, not in depth

- RNNs with many layers are very expensive to compute
- What we care about is memory
- Activation functions
  - Tanh is better than sigmoid. ReLU also popular
- When working with text, inputs are most frequently tokens or *word embeddings*

How to train RNNs

- Backpropagation Through Time (BPTT)
  - Unroll RNN
  - Feed full sequence. Forward, loss
  - Backprop + SGD with sums



- Input is truncated in mini-batches to reduce cost

Training issues

- Sharing weights, with longer sequences, easily yields
  - Vanishing gradient (favours close by patterns vs distant ones)
  - Exploding gradients and NaN losses (choose your poison)
    - Highly unstable loss, eventually NaN. Large weights, eventually NaN.
- Clipping gradients to prevent exploding gradients
  - Scales gradient if the norm is above an (**arbitrary**) threshold
- For vanishing gradients, ReLUs

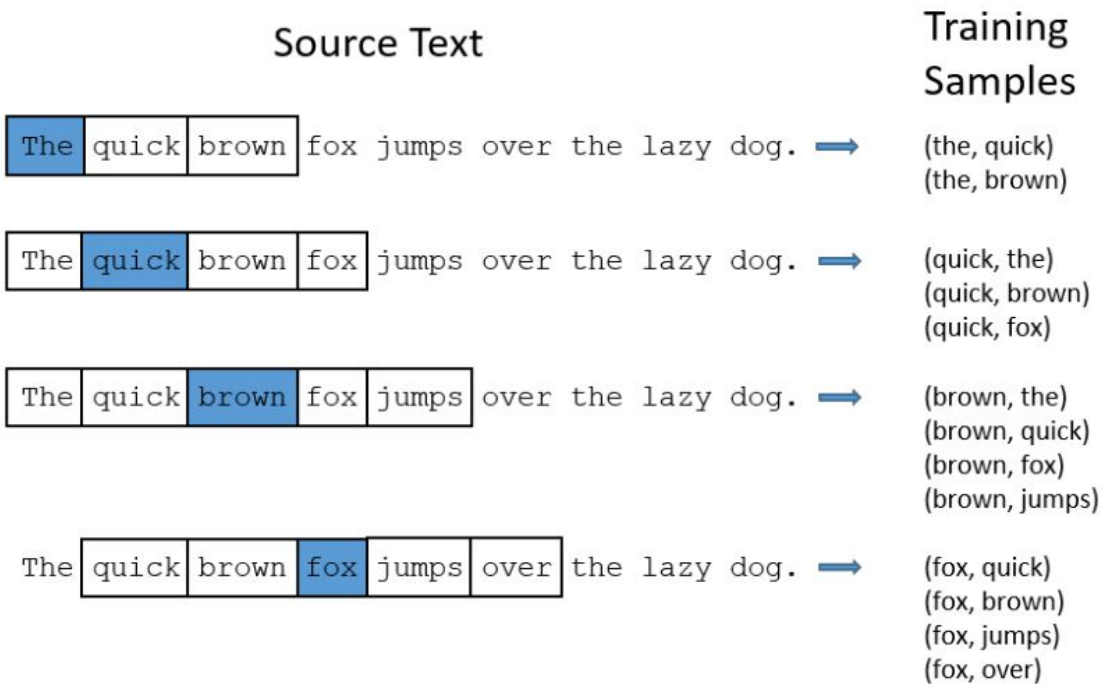
Key features of RNNs

- Can process inputs of any length (weight reuse)
  - For fixed sized inputs, other options are better (CNN, Transformers)
- Implements memory by design (recurrent edge, hidden state)
- Model complexity is independent of input length (weight reuse)
- Temporal invariance (weight reuse)
  - Weight reuse is cool, but it must account for all data relations

[51,52]

Word embedding task

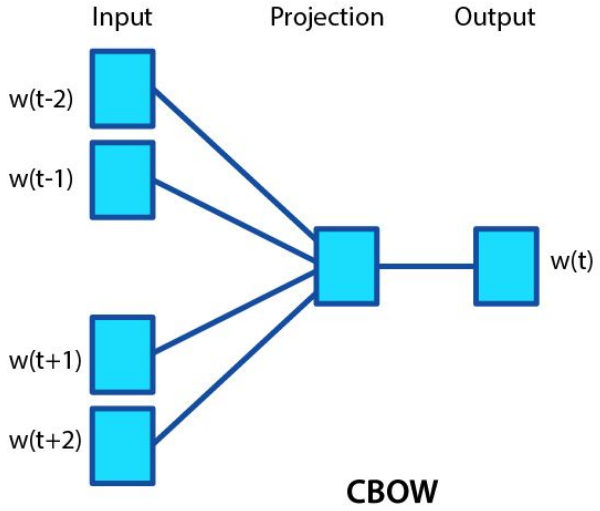
- Words are defined by their context
- Unlabeled text: Endless source of training data
- Use a sliding window of fixed length



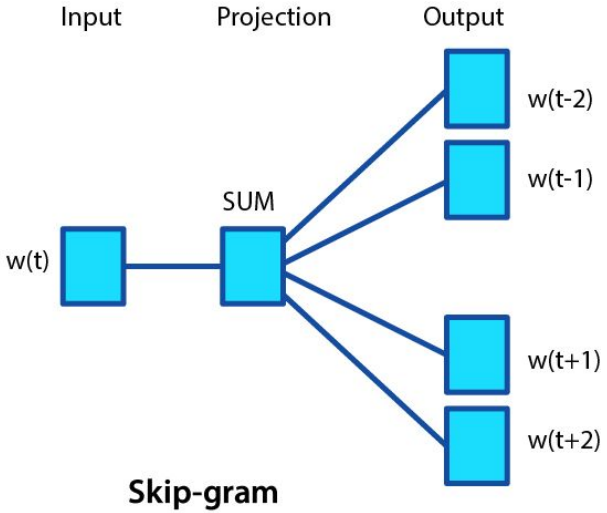
Word embedding models

Forget about the output!
Give me the intermediate representations

Predict **word** given *context*



Predict **context** given *word*



- Faster and slightly better for frequent words
- Good for little training data and rare word representation

Word embedding properties

- Compute word similarity (cosine distance among embedding vectors)
- Find *"regularities"*
  - Add & Substract
- So much bias...
- Go play
  - https://projector.tensorflow.org/ https://ronxin.github.io/wevi/

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Advanced RNNs

Dario Garcia Gasulla
dario.garcia@bsc.es

The limitations

- RNNs keep multiplying the same weight matrix over and over, which makes it error prone
- The state encodes both short and long term relations, which gets complicated as input sequences grow (how much can you store in a single set of weights???)
- What if we make the state more powerful? Let's complicate the model, yay!

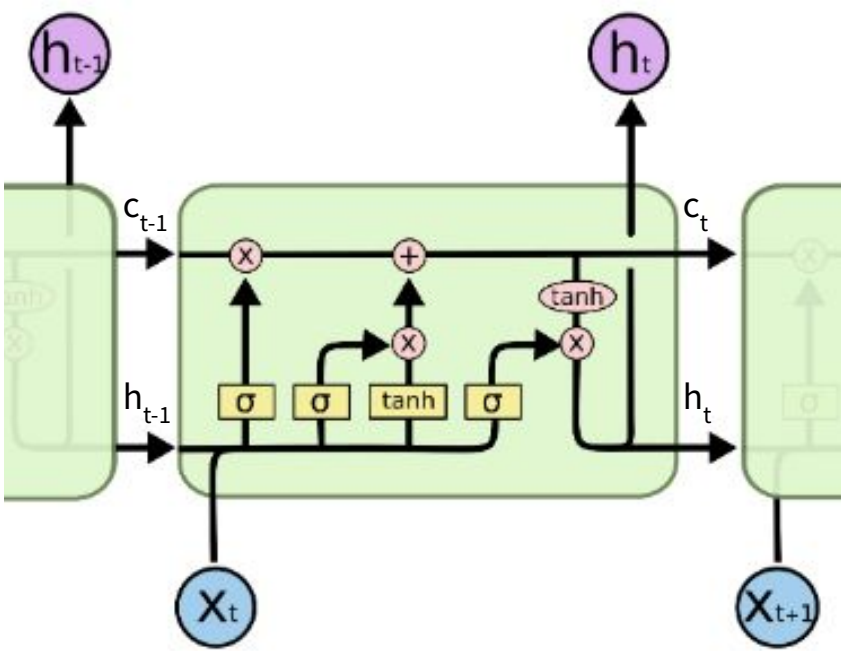
LSTM

- Long-Short Term Memory
  - In addition to the **hidden state**, add a **cell state**.
  - **Hidden** state characterizes previous data step (**short-term**, large updates)
  - **Cell** state characterizes historical data (**long-term**, small updates)
  - Include gate operators to **erase**, **write** and **read** from the cell (long)
    - Different sets of weights
- Gates regulate the cell state, decide the operation (e/r/w), it's target (cell state segment) and magnitude (gate in range [0,1]), based on context

[42,54]

Inside a LSTM

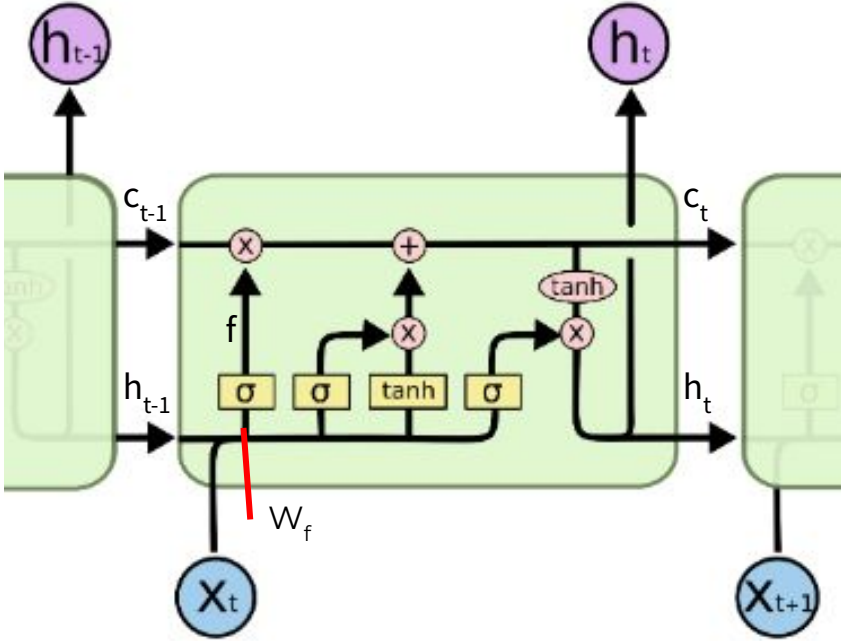
- **Cell** state (**long-term**, small updates)
  - Information flow
- **Hidden** state characterizes previous data step (**short-term**, large updates)



Inside a LSTM

All gates combine current data (x_t) and past state (h_{t-1})
tanh: Encode & Normalize [-1,1] sig: Weight & Scale [0,1]

- Forget gate:
- Given current data and short memory, what do we keep from the long
  - $f_t = \sigma(W_f \cdot [h_{t-1}, x_t])$
    - $f_t \approx 0$ -> forget
    - $f_t \approx 1$ -> retain
  - $c_t = f_t * c_{t-1}$
    - New cell state, sparsified

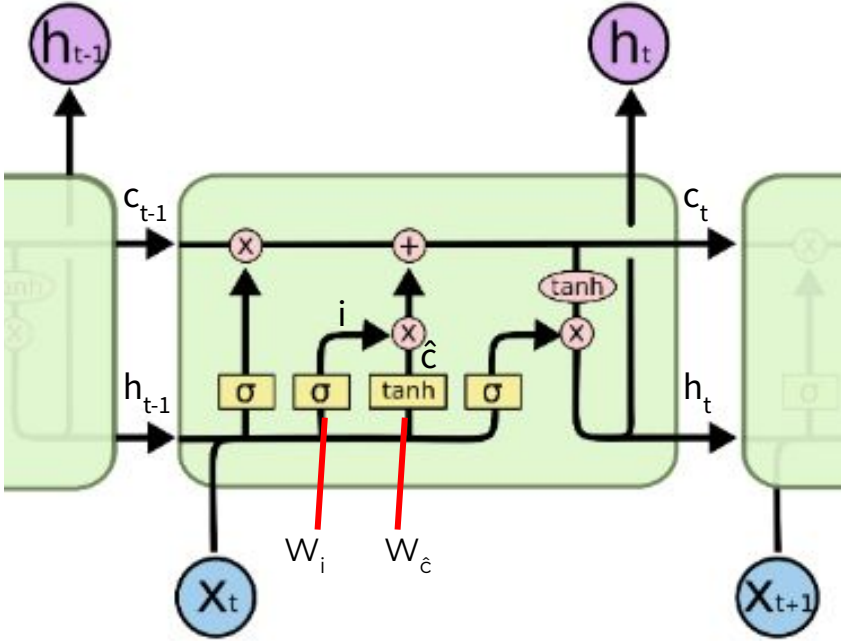


[43,53]

Inside a LSTM

All gates combine current data (x_t) and past state (h_{t-1})
tanh: Encode & Normalize [-1,1] sig: Weight & Scale [0,1]

- Input gate:
- Given current data and short memory, *how much* do we add to long
  - $i_t = \sigma(W_i \cdot [h_{t-1}, x_t])$
- And *what* do we add to long
  - $\hat{c}_t = \mathbf{tanh}(W_c \cdot [h_{t-1}, x_t])$
- New cell state, complemented
  - $c_t = (i_t * \hat{c}_t) + (f_t * c_{t-1})$

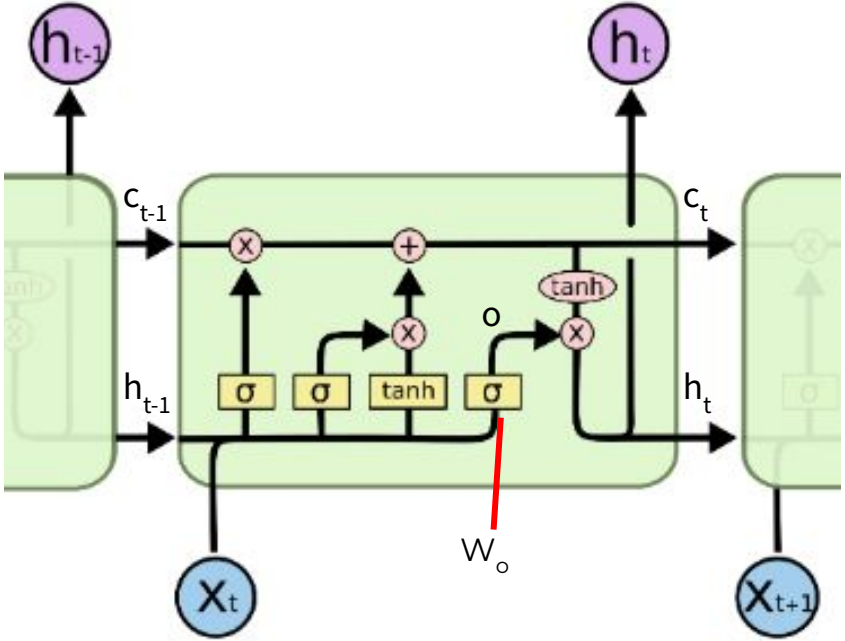


[43,53]

Inside a LSTM

All gates combine current data (x_t) and past state (h_{t-1})
tanh: Encode & Normalize [-1,1] sig: Weight & Scale [0,1]

- Output gate:
- What parts of long are output
  - $o_t = \sigma(W_o \cdot [h_{t-1}, x_t])$
- Filter them
  - $h_t = o_t \times \mathbf{tanh}(c_t)$ (new hidden state)



- 4 weight matrices of same size
  - num. hidden units * (num. hidden units + input emb. size) + num. hidden units

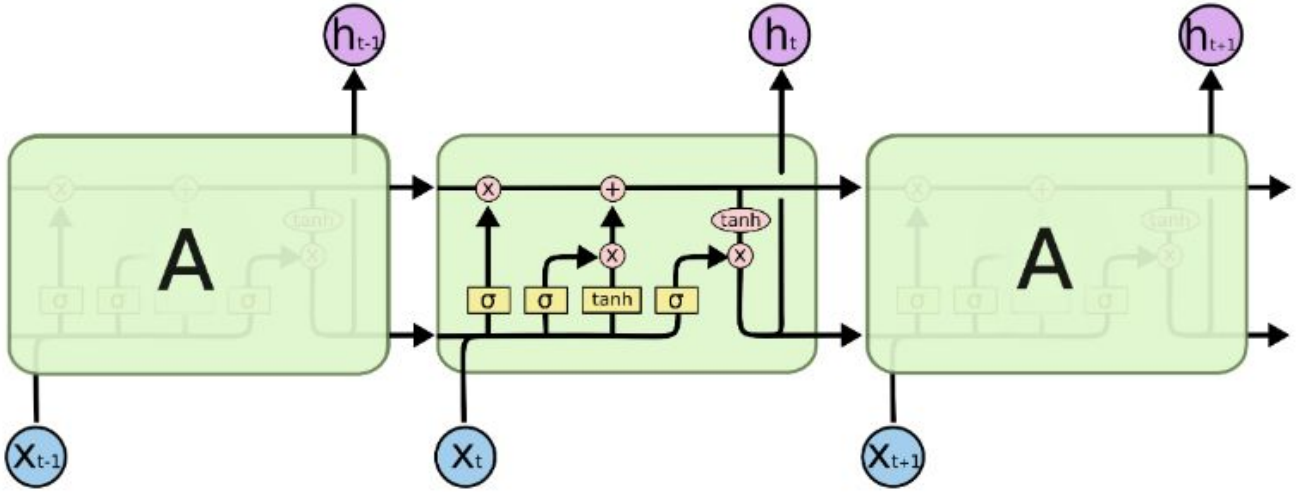
input weights bias

[43,53,59]

Why do LSTM work

- If forget gate is set to 0 (remember everything), long-term relations are always kept
- It includes a sort of short-cut, as long as you do not forget everything, gradient will flow!*

* No more W exp, but sigmoid can still vanish the gradient

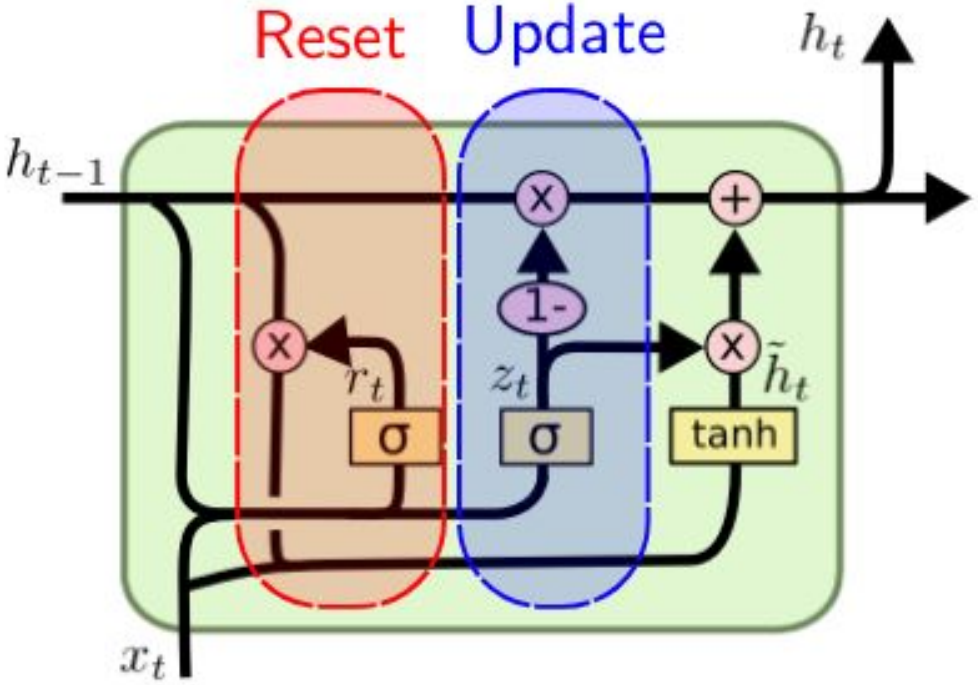


Gated Recurrent Units

- A simplified version of LSTMs (thanks?)
  - No cell state
  - Update gate: What in the hidden state is updated/left as it is
    - LSTMs forget gate + input gate
  - Reset gate: Which part of the *previous* hidden state is used
    - Close to 1: Previous state has more relevance
    - Close to 0: New state has more relevance

Inside a GRU

- Update gate: How to change hidden state
  - $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$
- Reset gate: How to combine
  - $r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$
- New hidden state content
  - $\hat{h}_t = \tanh(W_h \cdot [r_t \times h_{t-1}, x_t])$
- Hidden state output
  - $h_t = (1-z_t) \times h_{t-1} + z_t \times \hat{h}_t$



Vanilla RNN vs LSTM vs GRU

- Of all RNN variants, LSTMs and GRUs are the most widely used
  - Vanilla falls short in complex tasks, lacking long memory capacity
- Main difference between LSTM and GRUs: Complexity
  - Training GRUs is faster and includes less parameters
  - Performance-wise, there are no consistent results
- RNN variants are the state-of-the-art architectures for handling memory
  - Spoiler: This is why Transformers **CANNOT** fully replace RNNs

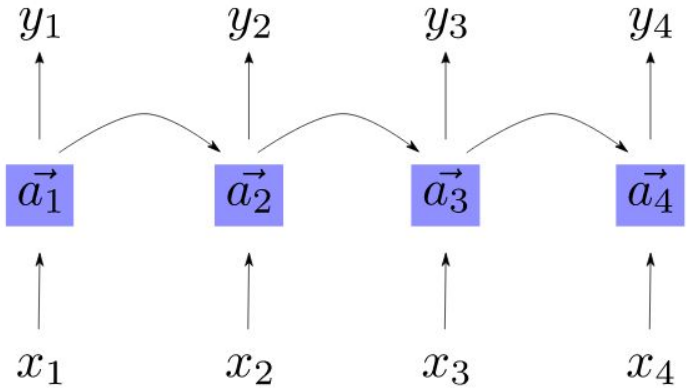
[45,46]

RNNs Extensions

Dario Garcia Gasulla
dario.garcia@bsc.es

Bidirectional RNNs

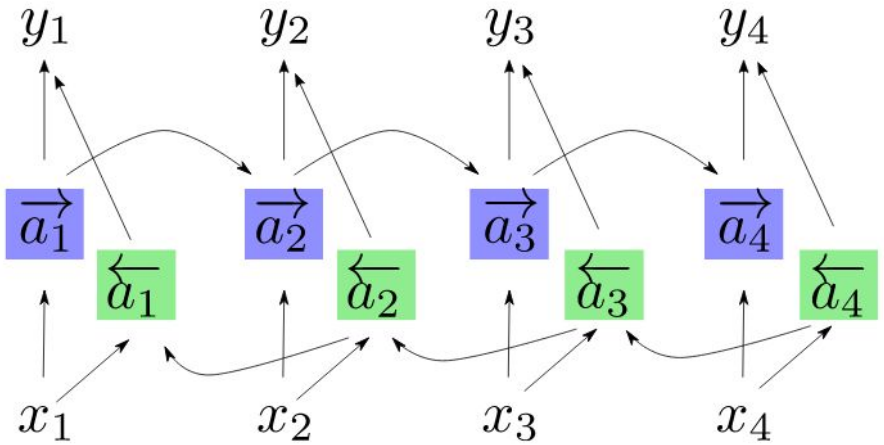
- Hidden states encode *past* states to influence current state



- What about future states? What if read from end to start?
  - PoS tagging, machine translation, speech/handwritting recognition

Bidirectional RNNs

- “Reading” in both directions at the same time
- Two RNNs, one in each direction, with different weights, concatenating their outputs

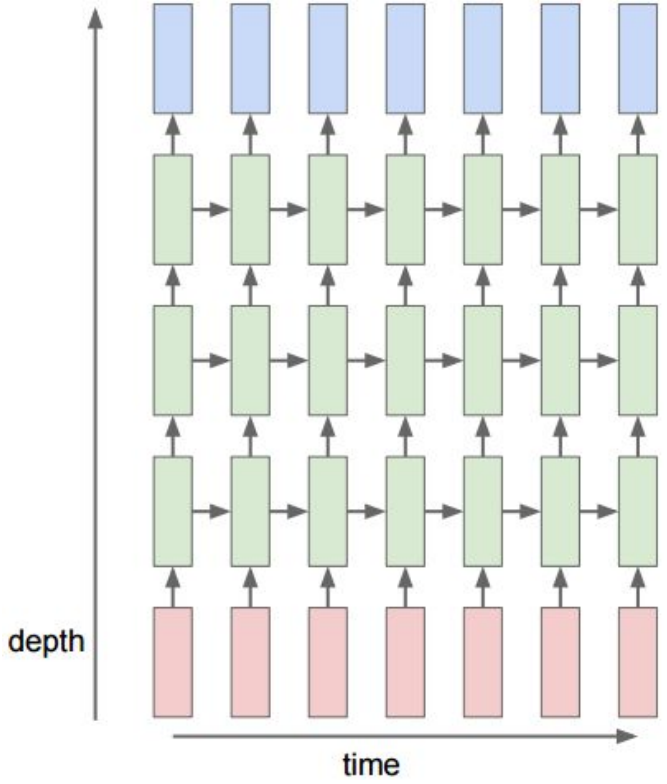


Training Bidirectional RNNs

- Both RNNs trained concurrently, but dependencies must be respected
  - Forward: Compute output of both RNNs and combine step by step
  - Backward: Compute gradient of both RNNs and combine step by step
- If possible (complete input sequence available), and in general (if the problem does not contradict) bidirectionality is a plus

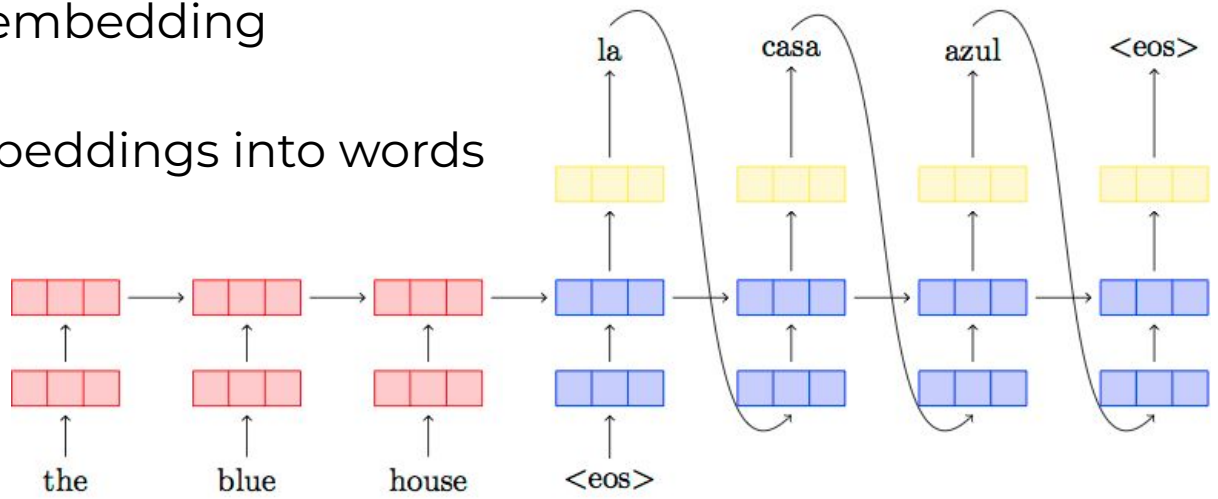
Multi-layer RNNs

- So far, only one layer used
  - Depth is provided by unrolling
- Multi-layer RNNs
  - Stacking layers (rarely more than 4)
  - Provide extra abstraction capacity
  - A computational nightmare



Encoder-Decoder RNNs (seq2seq)

- Sequence to sequence of variable length (Neural Machine Translation)
- **One RNN** encodes words within their context
- Generates a sentence embedding
- **One RNN** decodes embeddings into words
- *Start* and *End* tokens



Encoder-Decoder practical details

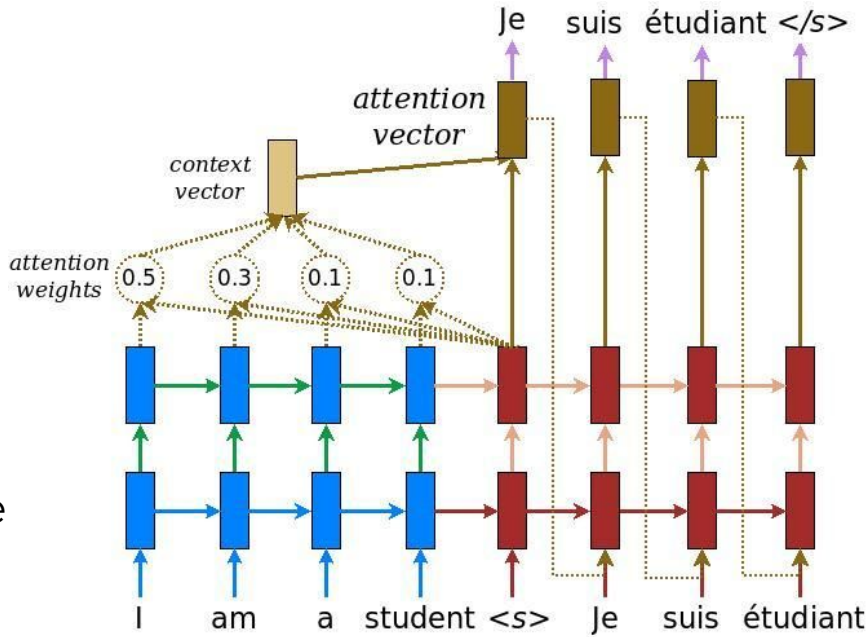
- Applications
  - Automatic language translation
  - Dialogue generation
  - Document summarization
  - Automatic response generation
  - Input parsing
- Training
  - Each decoding step generates one loss
  - Negative log-likelihood of the true outcome at that point
  - Average all losses at each step
  - Decoder state input is true outcome of the previous one (not the actual prediction)

From Encoder-Decoder to Attention

- seq2seq limitations
  - Full sentence into a fixed-sized, unique embedding (bottleneck)
  - Different parts of the decoder focus on different parts of the input
  - In inference
    - Greedy decoding: Carrying on errors
    - Beam search decoding: Keep top *k* branches, find most probable path
- Solution: Let each decoder step decide the part of the whole input to use

Seq2seq with attention

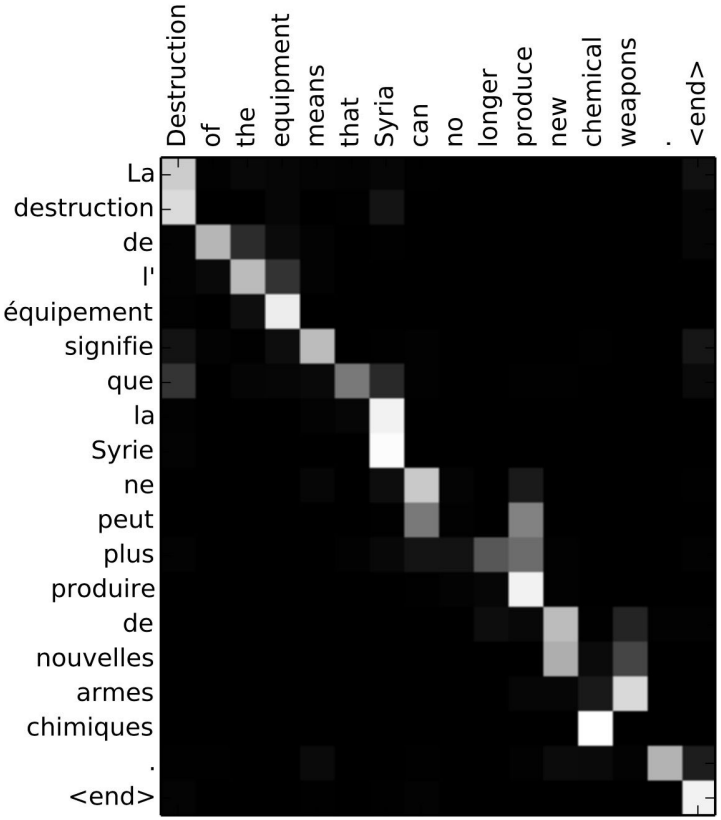
- Each decoder state
  - Compute a score per enc. hidden state
  - Turn into probabilities (softmax)
  - Dot prod. w/ hidden enc. states
  - Sum to make the fix-len
  - Concatenate with hidden decoder state
  - Output and fed to next step



[49,57,58]

Why seq2seq with attention

- Enables one different context for each decoding step
  - No fix-sized bottleneck
- Provides shortcuts (better gradient flows)
- More fine-grained -> better interpretability



A typical RNN model today...

"An encoder-decoder

bidirectional,

multilayered

LSTM-based

RNN with an

attention mechanism"

References

[40] Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks, May 21, 2015. http://karpathy.github.io/2015/05/21/rnn-effectiveness/
[41] https://cs.stanford.edu/people/karpathy/deepimagesent/
[42] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780.
[43] http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[44] Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:1412.3555 (2014).
[45] Jozefowicz, R., Zaremba, W., Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In Proceedings of the 32nd International Conference on Machine Learning(ICML-15) (pp. 2342-2350).

References

[46] Chung, J., Gulcehre, C., Cho, K., Bengio, Y. (2014).Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.
[47] https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks/
[48] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." arXiv preprint arXiv:1409.3215 (2014).
[49] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).
[50] Britz, Denny, Anna Goldie, Minh-Thang Luong, and Quoc Le. "Massive exploration of neural machine translation architectures." arXiv preprint arXiv:1703.03906 (2017).

References

[51] Mikolov, Tomas, et al. Distributed Representations of Words and Phrases and their Compositionality.
[52] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.
[53] https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2
[54] https://www.geeksforgeeks.org/understanding-of-lstm-networks/
[55] Arias-Duart, A., Parés, F., Garcia-Gasulla, D., & Giménez-Ábalos, V. (2022, July). Focus! Rating XAI Methods and Finding Biases. In 2022 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) (pp. 1-8). IEEE.
[56] Arias-Duart, A., et. al. (2022, October). Focus and Bias: Will it Blend?. In 2022 Catalan Conference on Artificial Intelligence (CCIA).

References

[57] https://guillaumegenthial.github.io/sequence-to-sequence.html
[58] https://distill.pub/2016/augmented-rnns/
[59] https://medium.com/analytics-vidhya/demystifying-lstm-weights-and-biases-dimensions-c47dbd39b30a

Dario Garcia-Gasulla (BSC)

dario.garcia@bsc.es

HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE