Recurrent Neural Networks

Javier Béjar

Deep Learning 2017/2018 Spring

Master in Artificial Intelligence (FIB-UPC)

Introduction

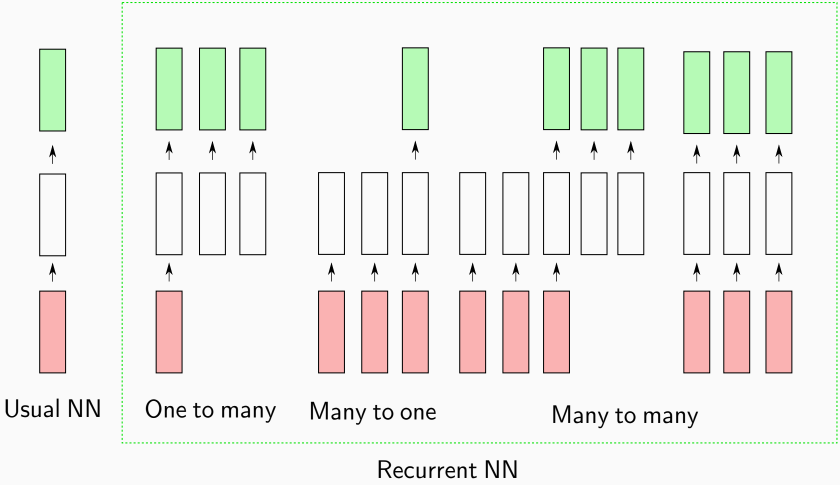
Sequential data

- Many problems are described by sequences
  - Time series
  - Video/audio processing
  - Natural Language Processing (translation, dialogue)
  - Bioinformatics (DNA/protein)
  - Control
- Model the problem = Extract elements' sequence dependencies

Long time dependences

- Sequences can be modeled using non sequential ML methods (e.g. Sliding windows), but
  - All sequences must have the same length
  - Order of the elements always matter
  - We cannot model dependencies longer than the chosen sequence length
- We need models that explicitly model time dependencies capable of:
  - Processing arbitrary length examples
  - Providing different mappings (one to many, many to one, many to many)

Input-Output mapping



One to many - Image captioning

⇒ Black and white dog jumps over bar

from Deep Visual-Semantic Alignments for Generating Image Descriptions Andrej Karpathy, Li Fei-Fei

Many to One - Sentiment Analysis

My flight was just delayed, s**t	$\Rightarrow$	Negative
Never again BA, thanks for the dreadful flight	$\Rightarrow$	Negative
We arrived on time, yeehaaa!	$\Rightarrow$	Positive
Another day, another flight	$\Rightarrow$	Neutral
Efficient, quick, delightful, always with BA	$\Rightarrow$	Positive

Many to Many - Machine Translation

[How, many, programmers, for, changing, a, lightbulb,?]

⇒ [Wie, viele, Programmierer, zum, Wechseln, einer, Glühbirne,?]

⇒ [Combien, de, programmeurs, pour, changer, une, ampoule,?]

⇒ [¿,Cuántos, programadores, para, cambiar, una, bombilla,?]

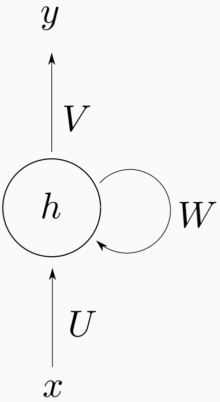
⇒ [Zenbat, bonbilla, bat, aldatzeko, programatzaileak,?]

Recurrent Networks

Recurrent Neural Networks

- RNN are feed forward NN with edges that span adjacent time steps (recurrent edges)
- At each time step nodes receive input from the current data and from the previous state
- This makes that input data from previous time steps can influence the output at the current time step
- RNN are universal function approximators (Turing Complete)

Recurrent Node



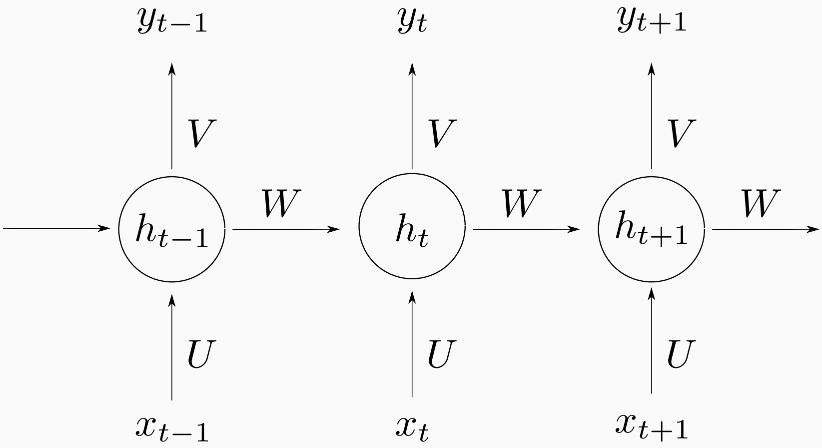
Recurrent Neural Networks

- Input (x) is a vector of values for time t
- The hidden node (h) stores the state
- Weights are shared through time
- Each step the computation uses the previous step

$$h^{(t+1)} = f(h^{(t)}, x_{t+1}; \theta) = f(f(h^{(t-1)}, x_t; \theta), x_{t+1}; \theta) = \cdots$$

- We can think of a RNN as a deep network that stacks layers through time

Training RNN (unfolding)



Activation Functions

- There are different choices for the activation function to compute the hidden state, but:
- The hyperbolic tangent function (*tanh*) is a popular choice versus the usual sigmoid function
- Good results are also achieved using the rectified linear function (ReLU) instead

RNN computation

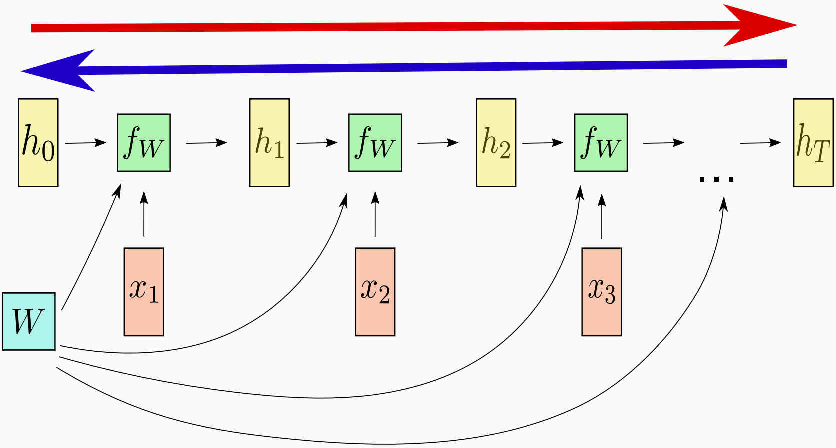
$$
\begin{aligned}
a^{(t)} &= b + W \cdot h^{(t-1)} + U \cdot x^{(t)} && (1)\\
h^{(t)} &= tanh(a^{(t)}) && (2)\\
y^{(t)} &= c + V \cdot h^{(t)} && (3)
\end{aligned}
$$

b and c are bias, an additional step can be added depending on the task

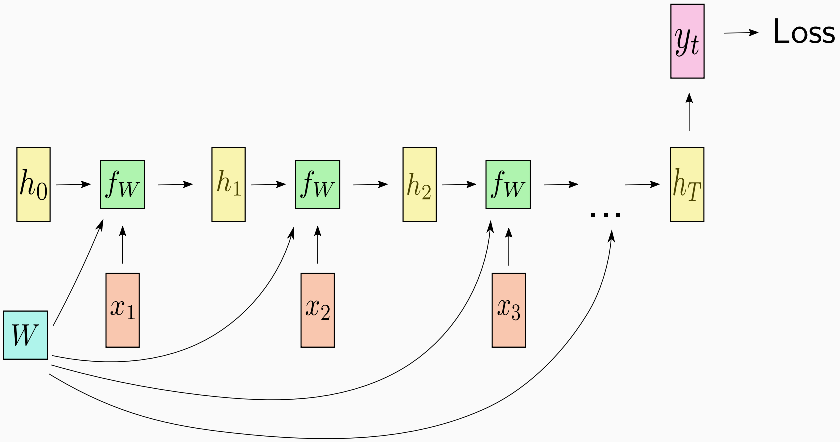
Training RNN

- RNN are usually trained using backpropagation
- The computation is unfolded through the sequence to propagate the activations and to compute the gradient
- This is known as Backpropagation Through Time (BPTT)

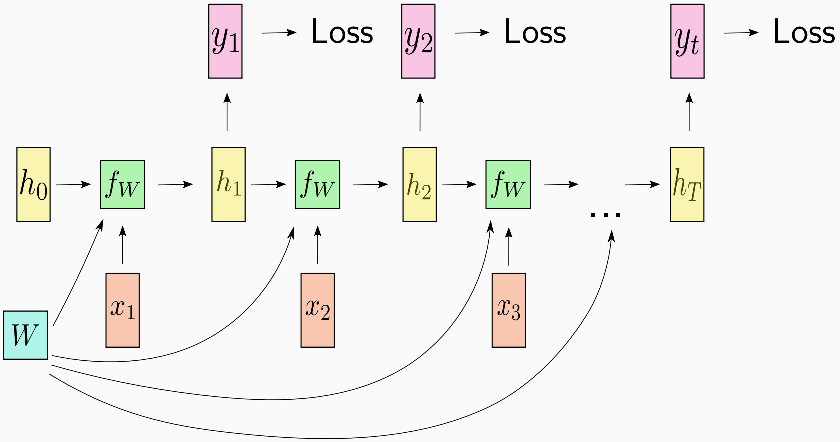
Recurrent NN unfolded



Recurrent NN (Regresssion/Classification)



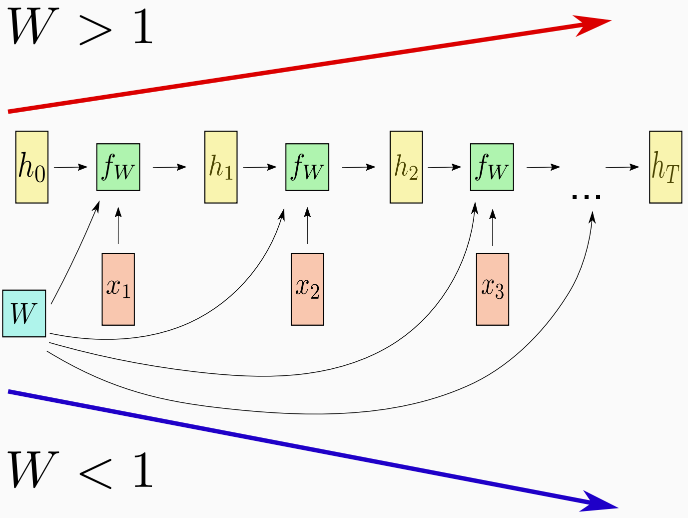
Recurrent NN (Sequence to Sequence)



Gradient problems

- Two main problems during training
  - Exploding Gradient
  - Vanishing Gradient
- Problems appear because of the sharing of weights
- Recurrent edge weights in combination with activation function magnify ($W > 1$) or shrink ($W < 1$) the gradient exponentially with the length of the sequence
- Clipping gradients and regularization are usual solutions to exploding gradient

Recurrent NN (Gradient problems)



Recurrent NN (Gradient problems)

Applying the chain rule makes that propagating the values forward and backward, the longer the sequence, the smaller the influence of the past:

$$\frac{\partial f_t}{\partial W} = \frac{\partial f_t}{\partial s_t}\frac{\partial s_t}{\partial W} = \frac{\partial f_t}{\partial s_t}\frac{\partial s_t}{\partial s_{t-1}}\frac{\partial s_{t-1}}{\partial W} = \cdots = \frac{\partial f_t}{\partial s_t}\frac{\partial s_t}{\partial s_{t-1}}\cdots\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W}$$

Beyond Vanilla RNN

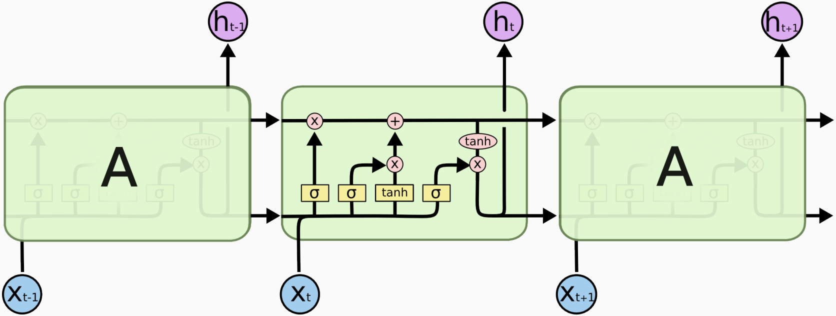
- Learning long time dependencies is difficult for vanilla RNN
- More sophisticated recurrent architectures allow reducing gradient problems
- **Gated RNNs** introduce memory and gating mechanisms
  - When to store information in the state
  - How much new information changes the state

LSTMs

Long Short Term Memory units (LSTMs)

- LSTMs specialize on learning long time dependencies
- They are composed by a memory cell and control gates
- Gates allow regulating how much the new information changes the state and flows to the next step
  - Forget Gate
  - Input Gate
  - Update Gate
  - Output Gate

LSTMs

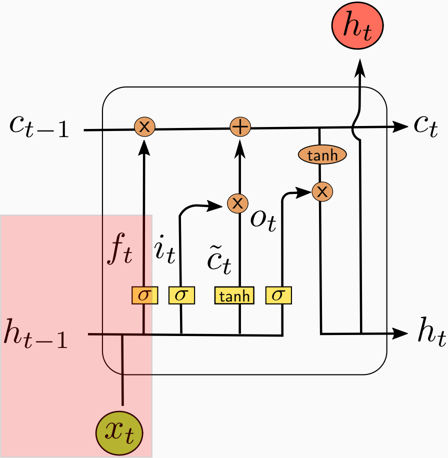


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Role of activation functions

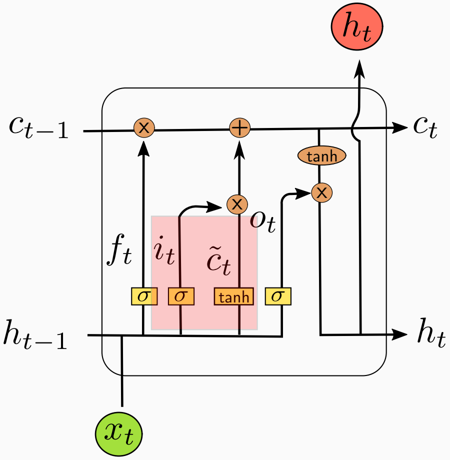
- Gates use as activation functions the sigmoid and tanh functions
- Their role is to perform fuzzy decisions
  - **tanh**: squashes value to range `[-1,1]` (substract, neutral, add)
  - **sigmoid**: squashes value to range `[0,1]` (closed, open)

LSTMs computations



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t]) \text{ (Forget)}$$

LSTMs computations

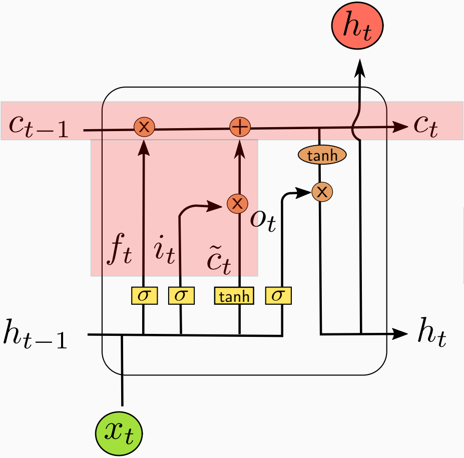


$f_t = \sigma(W_f \cdot [h_{t-1}, x_t])$ (Forget)

$i_t = \sigma(W_i \cdot [h_{t-1}, x_t])$ (Input)

$\tilde{c}_t = tanh(W_c \cdot [h_{t-1}, x_t])$

LSTMs computations



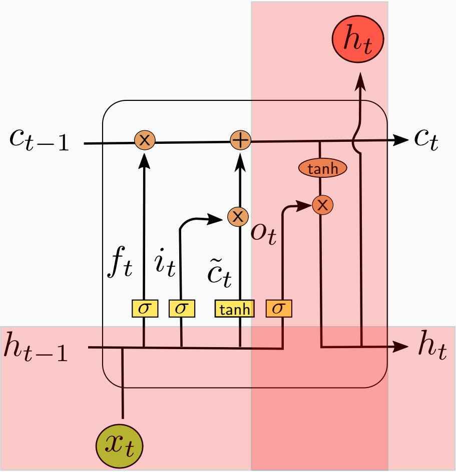
$f_t = \sigma(W_f \cdot [h_{t-1}, x_t])$ (Forget)

$i_t = \sigma(W_i \cdot [h_{t-1}, x_t])$ (Input)

$\tilde{c}_t = tanh(W_c \cdot [h_{t-1}, x_t])$

$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t$ (Update)

LSTMs computations



$f_t = \sigma(W_f \cdot [h_{t-1}, x_t])$ (Forget)

$i_t = \sigma(W_i \cdot [h_{t-1}, x_t])$ (Input)

$\tilde{c}_t = tanh(W_c \cdot [h_{t-1}, x_t])$

$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t$ (Update)

$o_t = \sigma(W_o \cdot [h_{t-1}, x_t])$

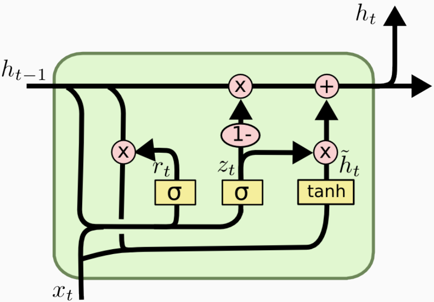
$h_t = o_t \times tanh(c_t)$ (Output)

GRUs

Gated Recurrent Units

- Reduces the complexity of the LSTMs
- Unifies the forgetting and the update gates as a unique update gate
- The update gate computes how the input and previous state are combined
- A reset gate controls the access to the previous state
  - near to one previous state has more effect
  - near to zero new state (updated) has more effect

GRU



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = tanh(W_h \cdot [r_t \times h_{t-1}, x_t])$$

$$h_t = (1 - z_t) \times h_{t-1} + z_t \times \tilde{h}_t$$

Pros/Cons

Vanilla RNN vs LSTMs vs GRUs

- Empirically Vanilla RNN underperforms on complex tasks
- LSTMs are widely used but GRUs are very recent (2014)
- There is not yet theoretical arguments in the LSTMs vs GRUs question
- Empirical studies do not shed light to the question (see references)

Vanilla RNN vs LSTMs vs GRUs

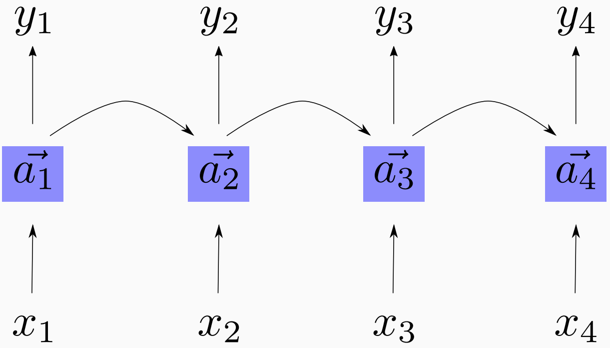
- Jozefowicz, R., Zaremba, W., Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (pp. 2342-2350).
- Chung, J., Gulcehre, C., Cho, K., Bengio, Y. (2014).Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Other RNN Variations

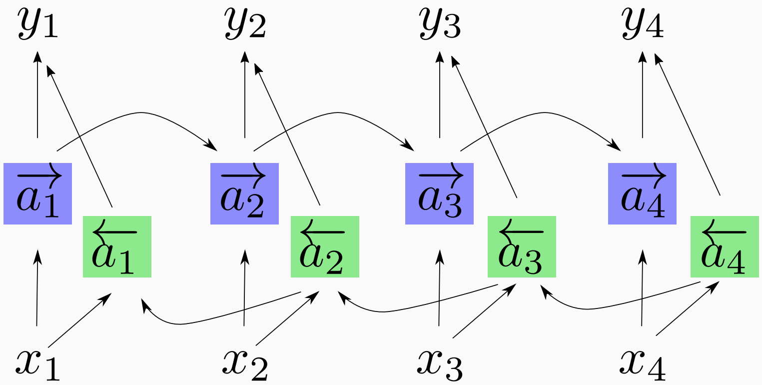
Bidirectional RNNs - Back from the future

- In some domains it is easier to learn dependencies if information flows in both directions
- For instance, domains where decisions depend on the whole sentence
  - Part of Speech tagging
  - Machine translation
  - Speech/handwritting recognition
- A RNN can be split in two, so sequences are processed in both directions

Regular RNNs (only forward)



Bidirectional RNNs (forward-backward)

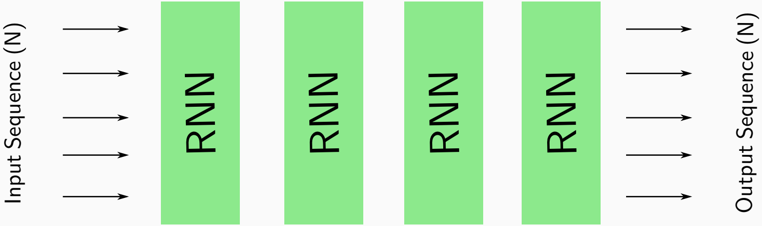


Bidirectional RNNs - Training

- Both directions are independent (no interconnections)
- Backpropagation needs to follow the graph dependencies, not all weights can be updated at the same time
  - Propagating Forward: First compute the RNN forward and backward pass from the inputs, then the outputs
  - Propagating Backward: First compute the RNN forward and backward pass from the outputs, then the inputs

Sequence to sequence

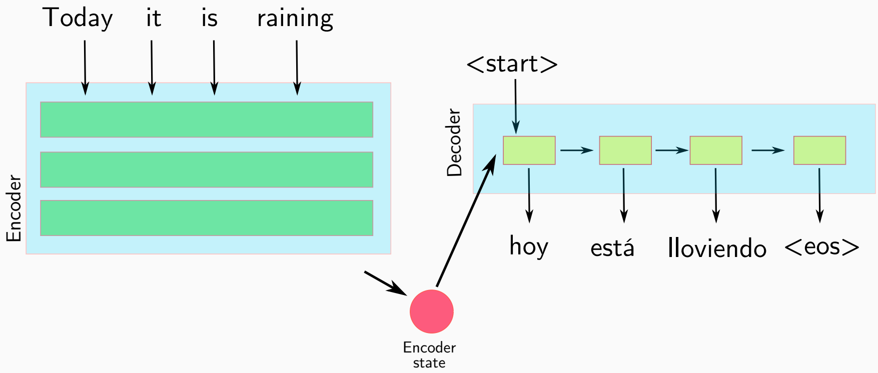
- Direct sequence association
  - Input and output sequences have the same lengths
  - All the outputs of the BPTT are used for the output
  - Training is straightforward



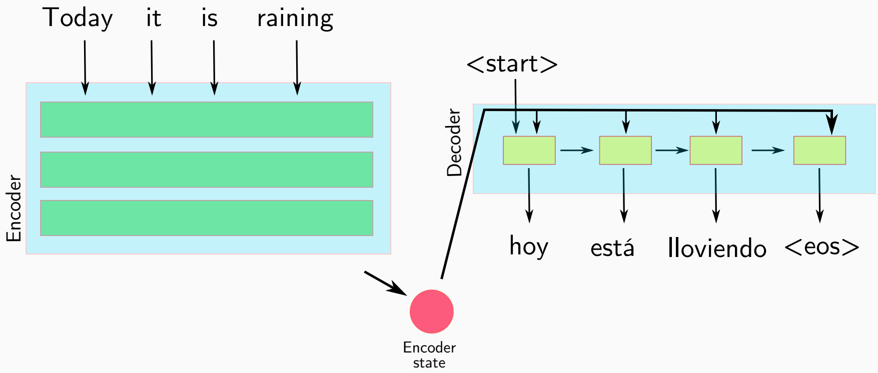
Sequence to sequence

- Encoder-decoder architecture
  - Input and output sequences can have different lengths
  - Encoder RNN summarizes input in a coding state
  - Decoder RNN generates output from that state
  - Different options connecting Encoder-Decoder (direct, peeking, attention) or training (teacher forcing)
  - Inference: The sequence is generated element by element using the output of the previous step or using a beam search

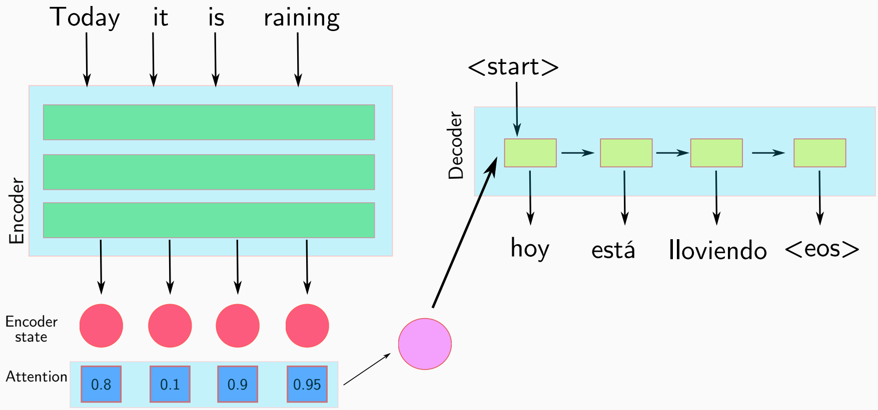
Encoder-Decoder (Plain)



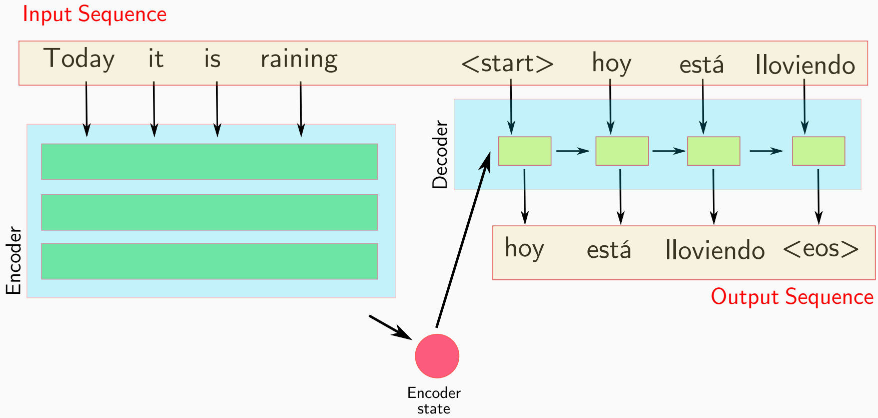
Encoder-Decoder (Peeking)



Encoder-Decoder (Attention)



Encoder-Decoder (Teacher Forcing)



Augmented neural networks

- RNNs are Turing complete, but it is difficult to achieve it in practice
- New architectures include:
  - Data structures to store information (Read/Write Operations)
  - RNNs control the operations
  - Attention mechanisms
- Graves, Wayne, Danihelka **Neural Turing Machines**, ArXiv preprint arXiv:1410.5401
- C. Olah, S. Carter, Attention and Augmented Recurrent Neural Networks, Distill, Sept 9, 2016

Applications

Large variety of applications

- Time series prediction
- NLP
  - POS tagging, Machine traslation, Question answering
- Reasoning
- Multimodal
  - Caption Generation for images/video
- Speech recognition/generation
- Reinforcement learning

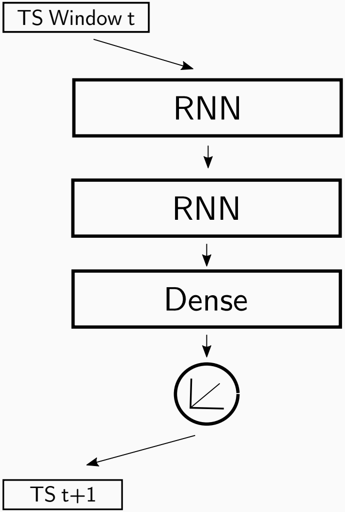
Guided Laboratory

Sequence prediction

- Sequence to value
  - Predicting the next step of a time series
  - Classification of time series
  - Predicting sentiment from tweets
  - Text generation (predicting characters)
- Sequence to sequence
  - Learning to add

Task 1: Air Quality prediction

- Time series regression
- Dataset: Air Quality every hour
- Goal: Predict wind speed next hour



Training time

35064 Train/ 8784 Test/6 lag

32 Neurons /1 Layer/30 epochs

$\sim$ 30 sec

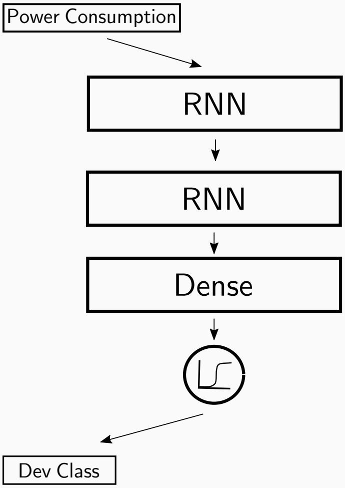
Task 2: Electric Devices

- Time series classification
- Daily power consumption of household devices (7 classes, 96 attributes)
- Goal: Predict household device class

Training time

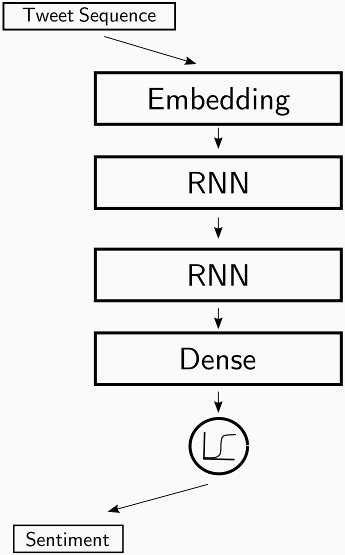
64 Neurons /2 Layers/30 epochs

$\sim$ 1 min



Task 3: Sentiment analysis

- Tweets (neg, pos, neutral)
- Tweets as sequences of words
- Preprocess: Generate vocabulary, recode sequences
- Embed sequences to a more convenient space



Training time

5000 words/ 40 dim embedding

64 Neurons /1 Layer/50 epochs

$\sim$ 3 min

Task 4: Text Generation

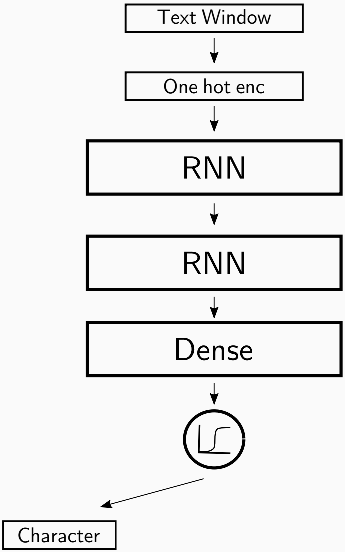
- Poetry text
- Character prediction from text windows
- Preprocess: sequences of characters' one hot encoding
- Text generation by predicting characters iteratively

Training time

poetry1/50 chars/3 skip

64 Neu/1 Ly/10 it/10 ep_it

$\sim$ 23 min



Task 5: Learning to add

- Predicting addition results from text
- Input sequence: `NUMBER+NUMBER`
- Output sequence: `NUMBER`
- Preprocess: sequences of characters' one hot encoding
- RNN Encode + RNN Decode

Training time

50000 ex/3 digits

128 Neu/1 Ly/50 ep

$\sim$ 5 min 30 sec

Task 5: Learning to add