Recurrent neural networks and attention mechanisms

Gustave Cortal

Summary

Last course's reminder

Reurrent neural networks

Attention mechanisms

Last course's reminder

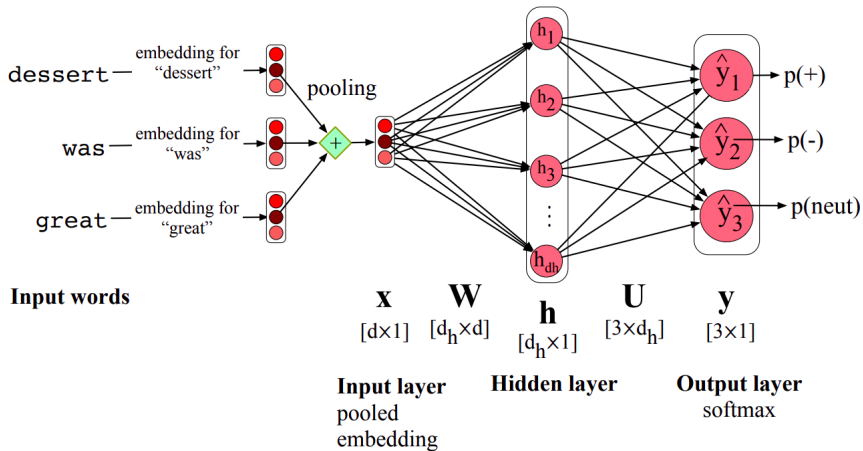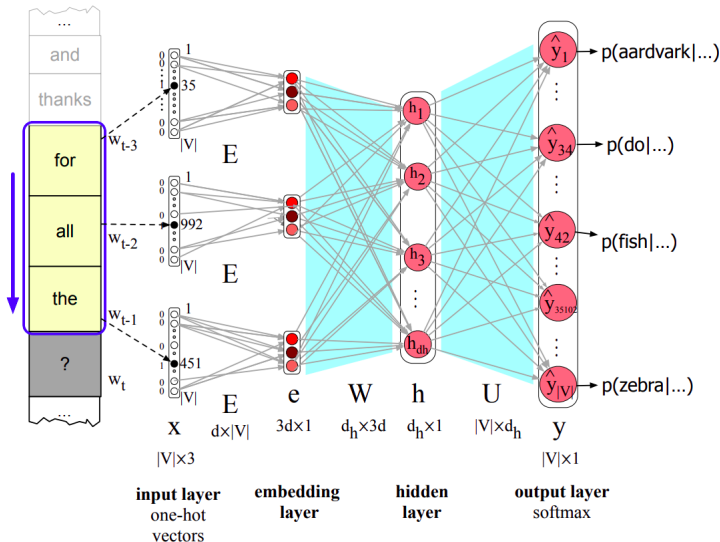# Feedforward network for text classification

Figure: Feedforward network sentiment analysis using a pooled embedding.

Feedforward neural networks for language modeling
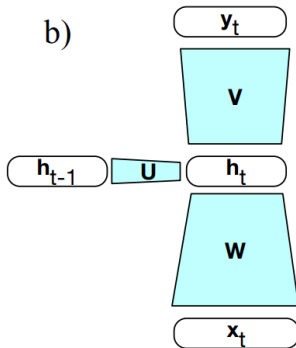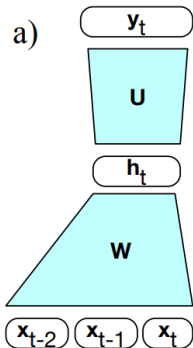
Recurrent neural networks

# Introduction

Language is a **temporal** phenomenon

Feedforward neural networks assumed **simultaneous access**: for language modeling, they look only at a fixed-size window of words, then slide this window over the input

Recurrent neural networks handle the temporal nature of language without using arbitrary fixed-sized windows: the hidden layer from the previous step provides a **memory** that encodes earlier processing and informs the decisions to be made at later steps
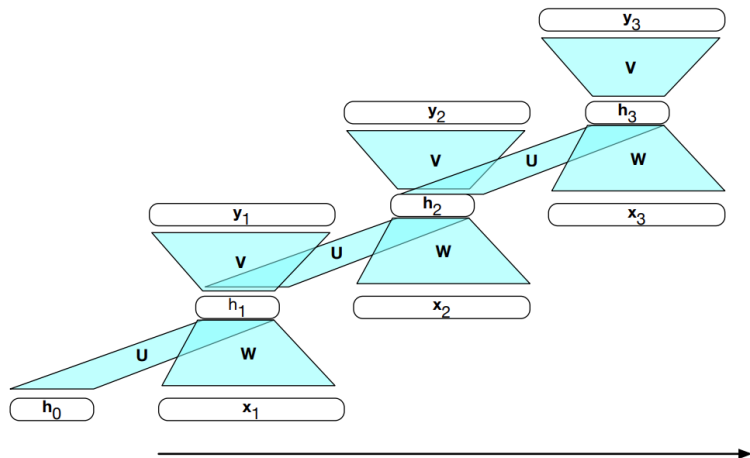
Feedforward vs recurrent neural networks
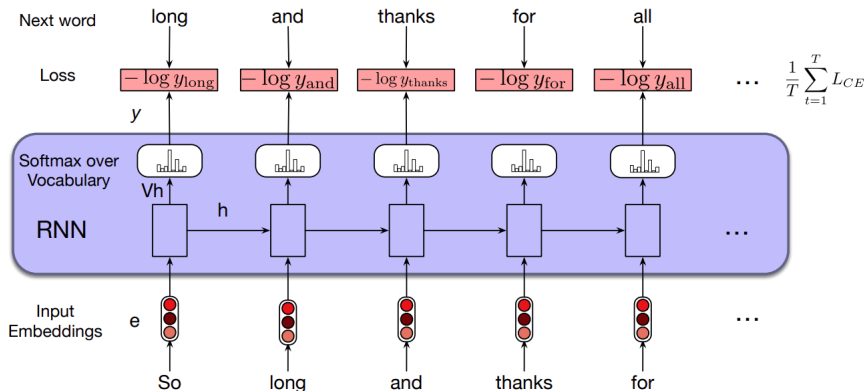
Recurrent neural networks
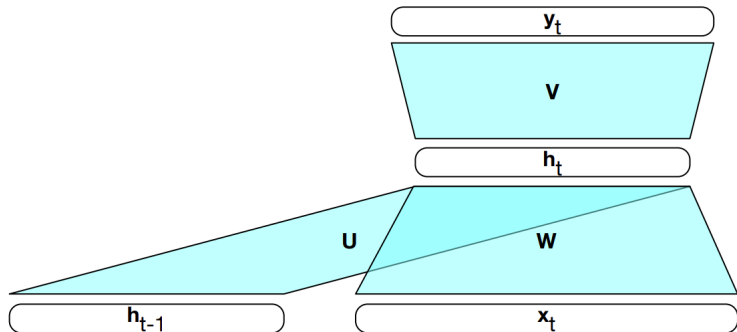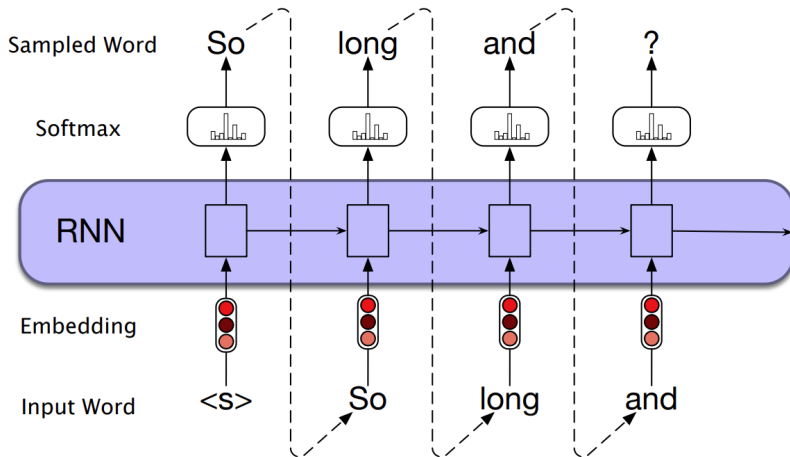

$$h_t = g(Uh_{t-1} + Wx_t)$$
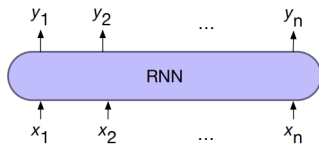$$y_t = softmax(Vh_t)$$

Recurrent neural networks

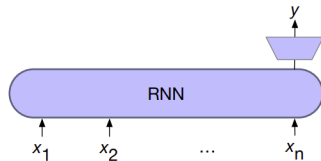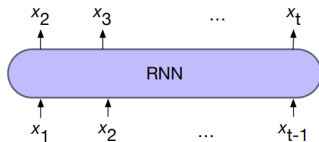# RNNs for language modeling

Sampling
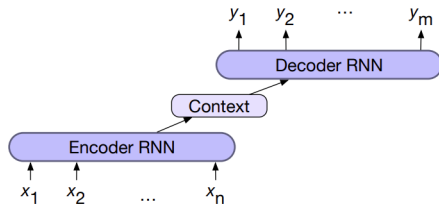
RNNs for other tasks

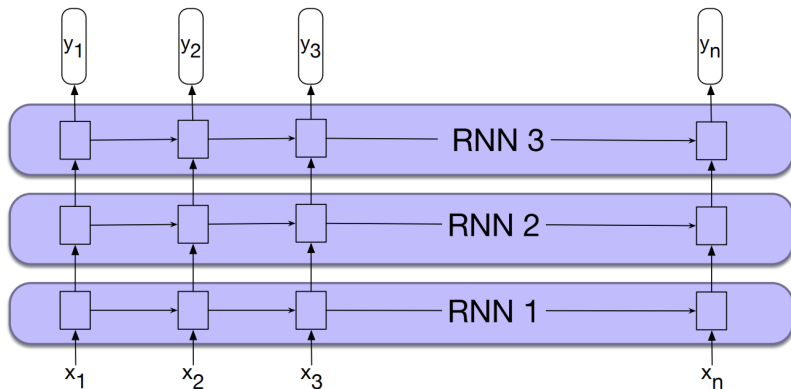a) sequence labeling

b) sequence classification

c) language modeling

d) encoder-decoder

Stacked RNNs

Bidirectional RNNs
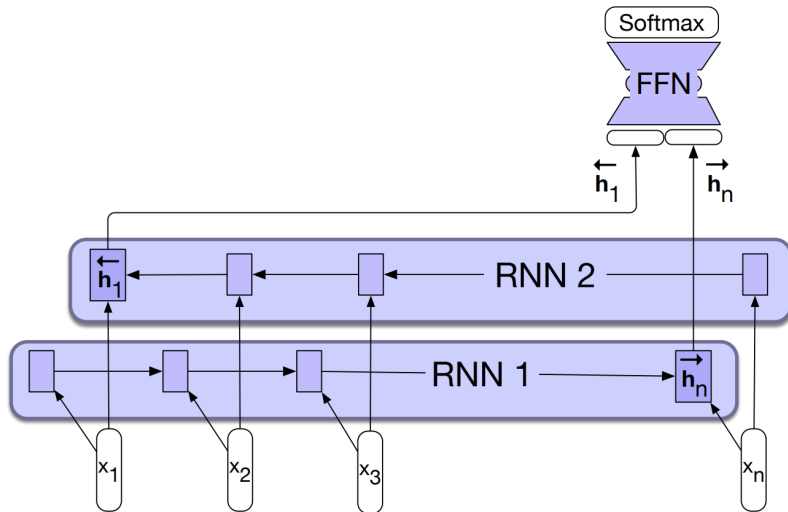
Training with encoder-decoder networks

Decoder

gold answers: llegó, la, bruja, verde, </s>

Total loss is the average cross-entropy loss per target word:

$$L = \frac{1}{T} \sum_{i=1}^{T} L_i$$

$L_1 = -\log P(y_1)$ $L_2 = -\log P(y_2)$ $L_3 = -\log P(y_3)$ $L_4 = -\log P(y_4)$ $L_5 = -\log P(y_5)$

per-word loss

softmax $\hat{y}$

hidden layer(s)

embedding layer

x_1 the x_2 green x_3 witch x_4 arrived <s> llegó la bruja verde
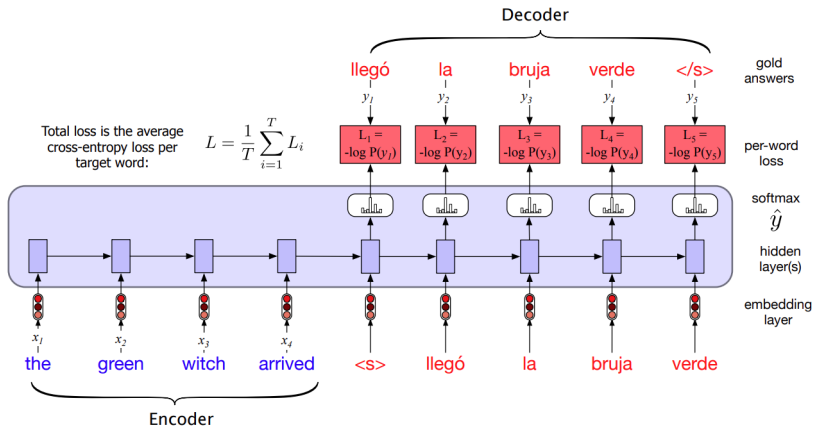
Encoder

Inference with encoder-decoder networks
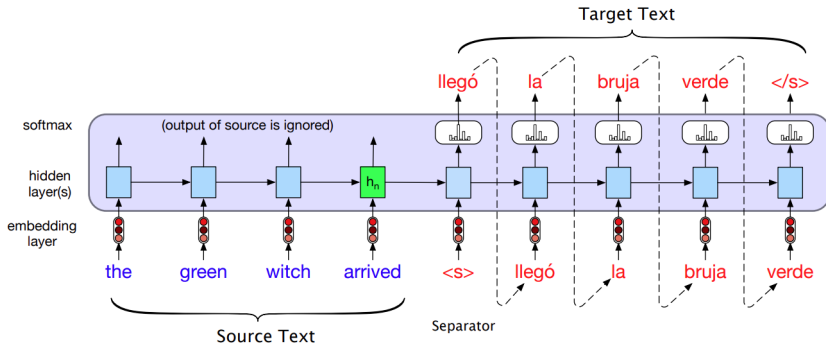
Final hidden state as a fixed context vector for the decoder

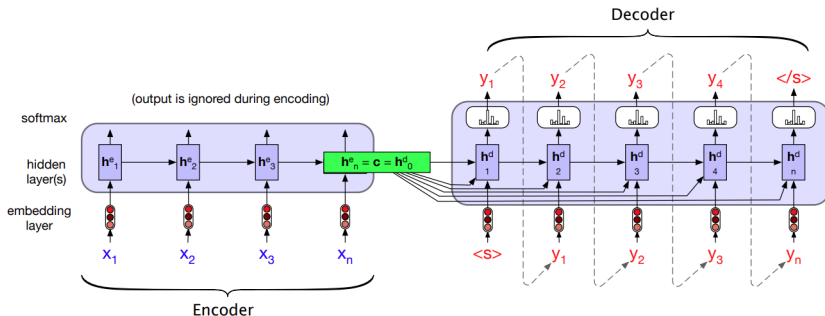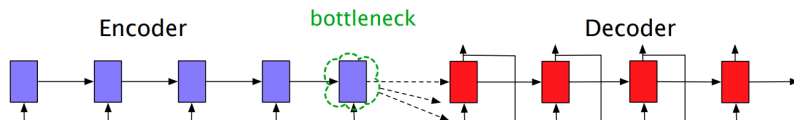# Attention mechanisms

The final hidden state acts as a bottleneck

This final hidden state must represent everything about the meaning of the source text

However, information at the beginning of the sentence may not be equally well represented in the context vector

Attention mechanisms: introduction

The *attention mechanism* is a **solution to the bottleneck problem**: it allows the decoder to get information from all the hidden states of the encoder

The idea of attention is to create the single fixed-length vector c by taking **a weighted sum of all the encoder hidden states**. The weights focus on a particular part of the source text that is relevant to the token the decoder is currently producing

Attention thus replaces the static context vector with one that is **dynamically** derived from the encoder hidden states, **different** for each token in

Dot-product attention

The first step in computing c_i is to compute how relevant each encoder state is to the decoder state captured in h_{i-1}^d

Then, implement relevance as **dot-product similarity**:

$$\text{score}(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

Then, apply a softmax to create a **vector of weights**, α_{ij}, that tells the proportional relevance of each encoder hidden state j to the prior hidden decoder state, h_{i-1}^d:

$$\alpha_{ij} = \frac{\exp(\text{score}(h_{i-1}^d, h_j^e))}{\sum_k \exp(\text{score}(h_{i-1}^d, h_k^e))}$$
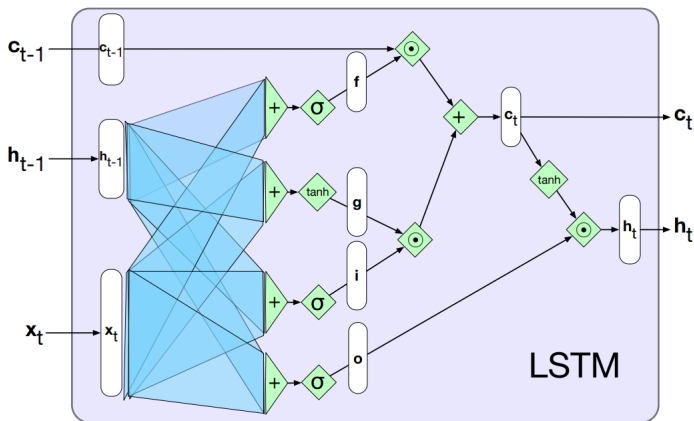
Finally, compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states:
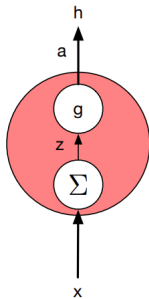
$$c_i = \sum_j \alpha_{ij} h_j^e$$

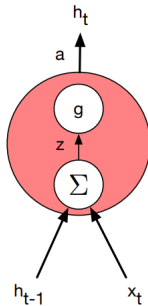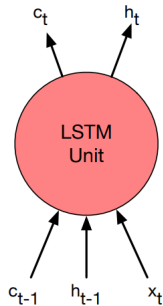# Encoder-decoder networks with dot-product attention
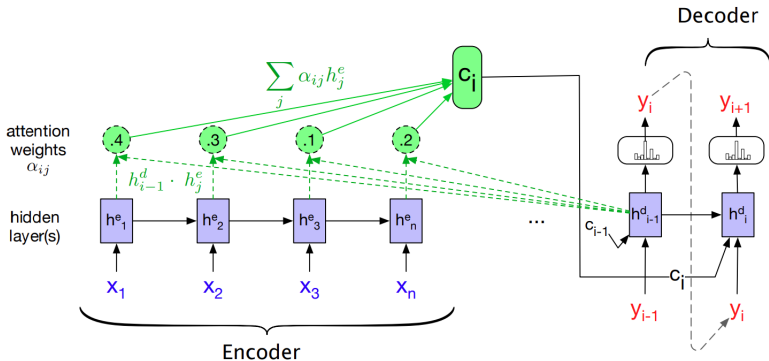
Long Short Term Memory network

LSTM

FNN vs RNN vs LSTM units

(a)   (b)   (c)