Recurrent neural networks and attention mechanisms

Gustave Cortal

Introduction

Language is a **temporal** phenomenon

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

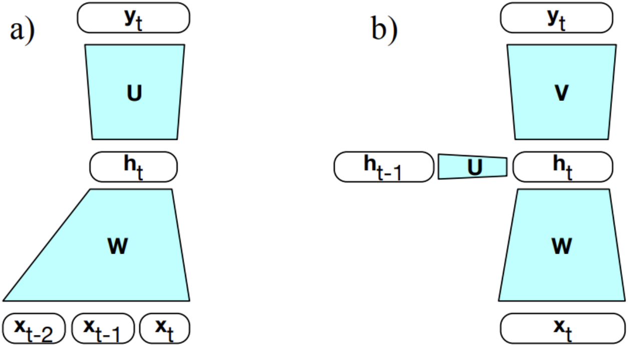
Introduction

Language is a **temporal** phenomenon

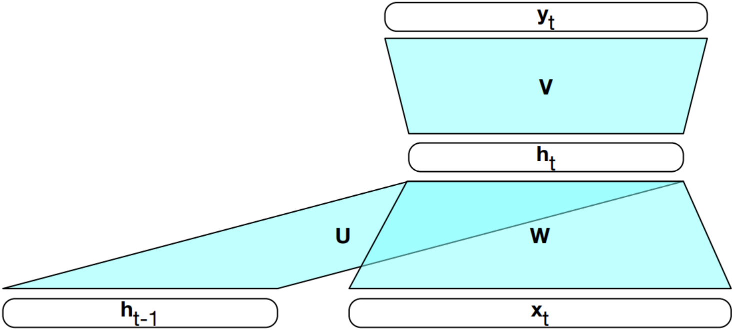
Feedforward neural networks assumed **simultaneous access**: for language modeling, they look only at a fixed-size window of words, then slide this window over the input

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

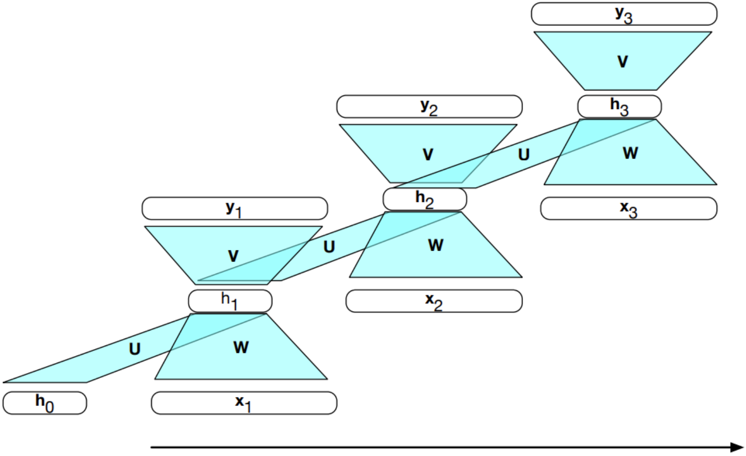


Recurrent neural networks

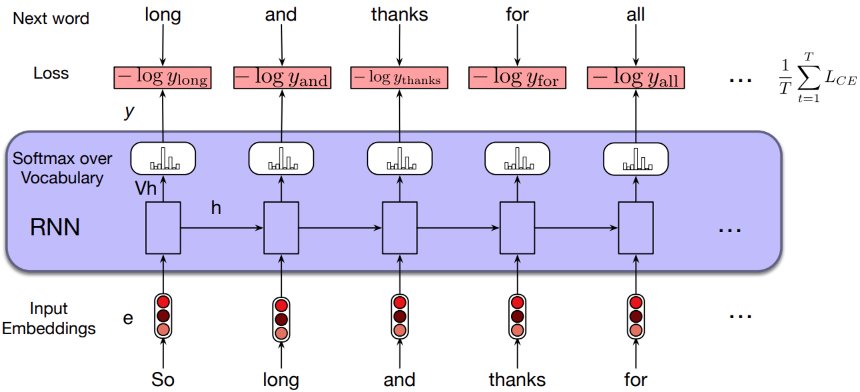


$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = softmax(Vh_t)$$

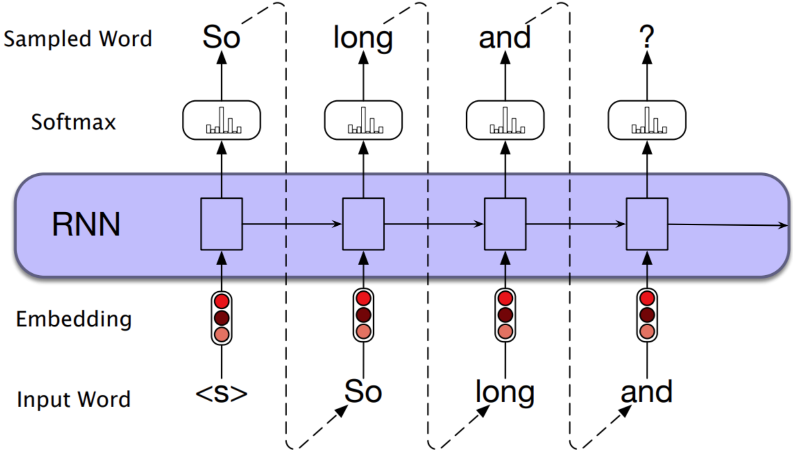
Recurrent neural networks



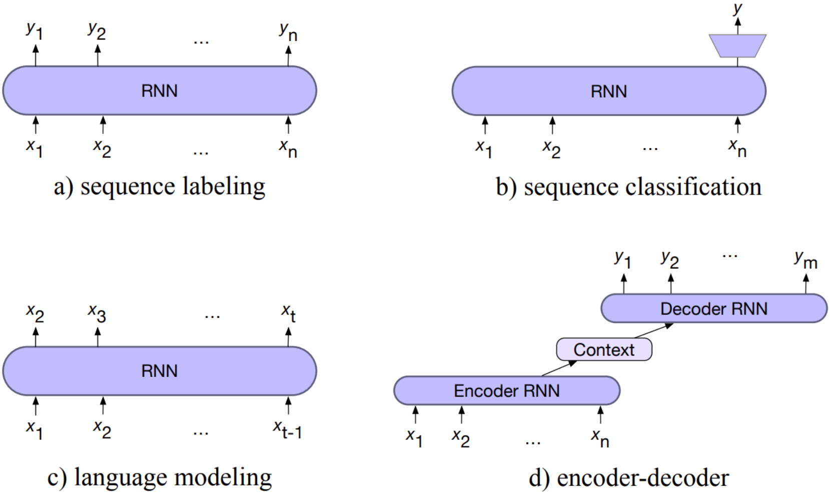
RNNs for language modeling



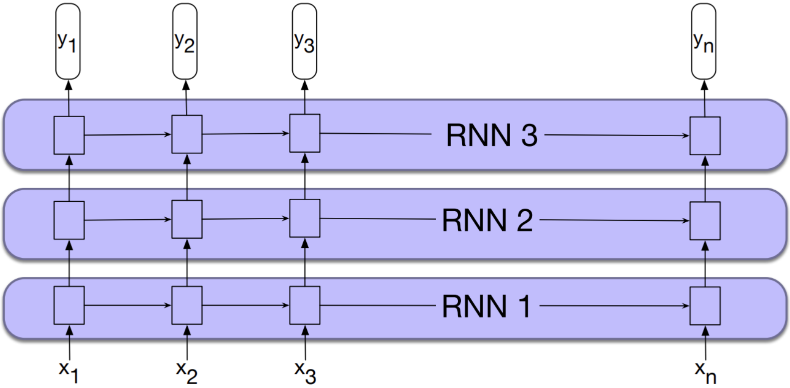
Sampling


Sampled Word
So
long
and
?
Softmax
RNN
Embedding
Input Word
<s>
So
long
and

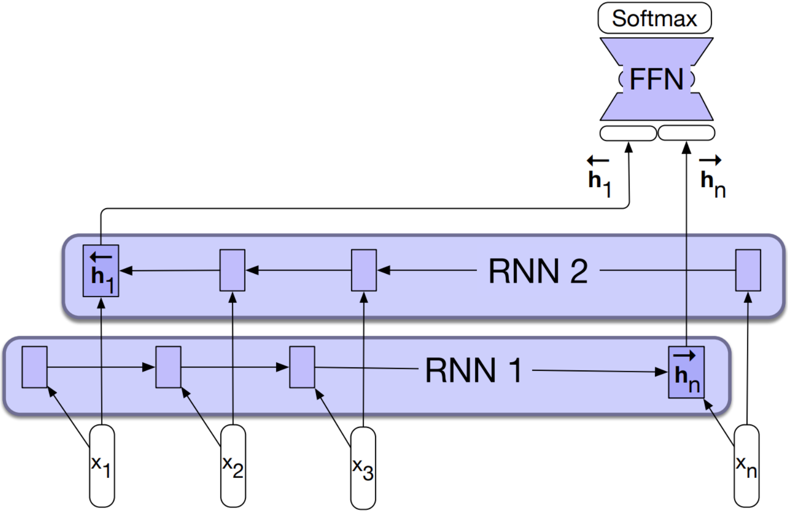
RNNs for other tasks



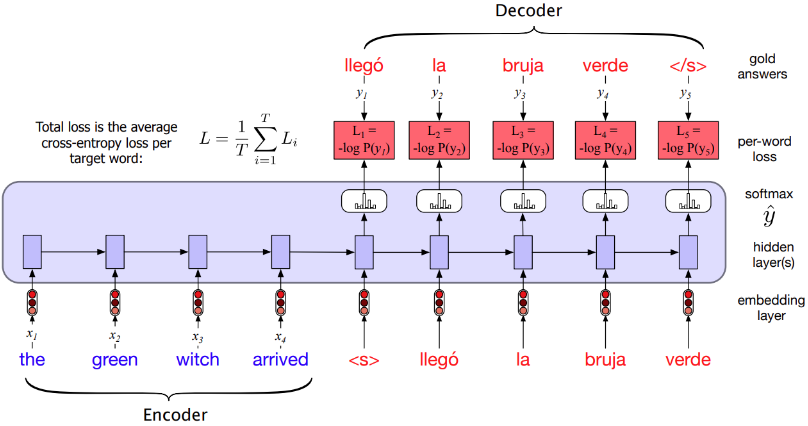
Stacked RNNs



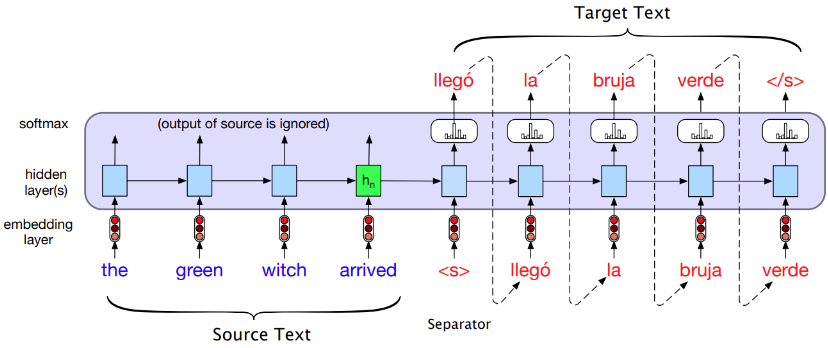
Bidirectional RNNs



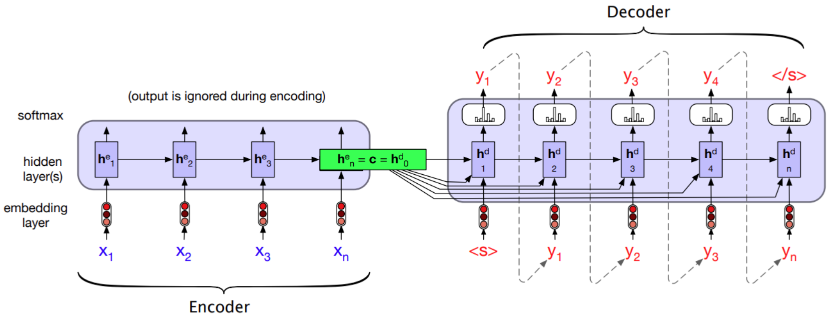
Training with encoder-decoder networks



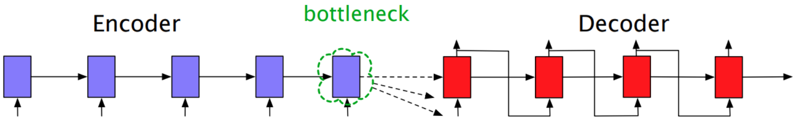
Inference with encoder-decoder networks



Final hidden state as a fixed context vector for the decoder

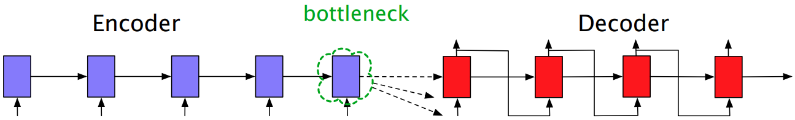


The final hidden state acts as a bottleneck



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

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

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 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^d_{i-1}

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^d_{i-1}

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

$$\text{score}(h^d_{i-1}, h^e_j) = h^d_{i-1} \cdot h^e_j$$

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^d_{i-1}

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

$$\text{score}(h^d_{i-1}, h^e_j) = h^d_{i-1} \cdot h^e_j$$

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^d_{i-1}:

$$\alpha_{ij} = \frac{\exp(\text{score}(h^d_{i-1}, h^e_j))}{\sum_k \exp(\text{score}(h^d_{i-1}, h^e_k))}$$

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^d_{i-1}

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

$$\text{score}(h^d_{i-1}, h^e_j) = h^d_{i-1} \cdot h^e_j$$

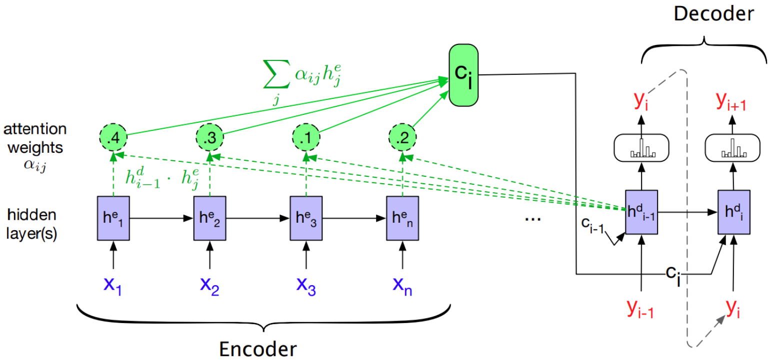
Then, apply a softmax to create a **vector of weights**, α_{ij}, that tells the proportional relevance of each encoder hidden state j to the prior hidden decoder state, h^d_{i-1}:

$$\alpha_{ij} = \frac{\exp(\text{score}(h^d_{i-1}, h^e_j))}{\sum_k \exp(\text{score}(h^d_{i-1}, h^e_k))}$$

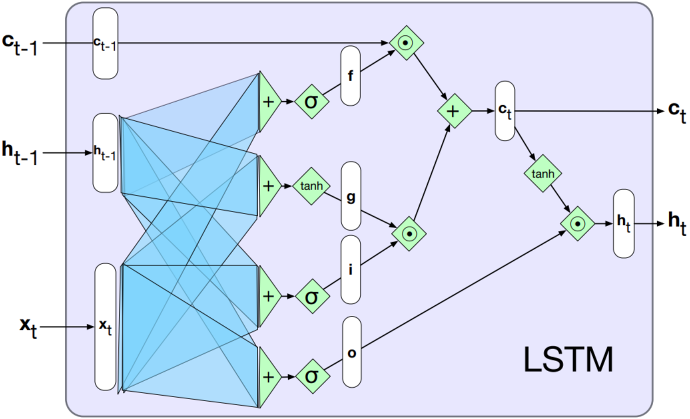
Finally, compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states:

$$c_i = \sum_j \alpha_{ij} h^e_j$$

Encoder-decoder networks with dot-product attention



Long Short Term Memory network



FNN vs RNN vs LSTM units