POLIMI GRADUATE SCHOOL OF MANAGEMENT

INTRODUCTION TO NATURAL LANGUAGE PROCESSING - 2

Andrea Mor - andrea.mor@polimi.it

WORD EMBEDDING

Distributed Representations of Words (a.k.a. word embeddings) are geometric representation of words/entities learned from the data/corpus in such a way that semantically related words are often close to each other.

In practice we attempt to embed entities onto a low-dimensional metric space in which similar words are placed close

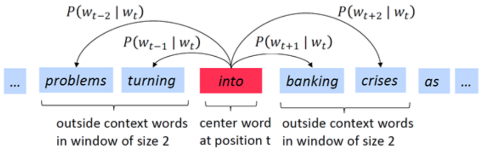
DISTRIBUTIONAL HYPOTHESIS

“You shall know a word by the company it keeps”

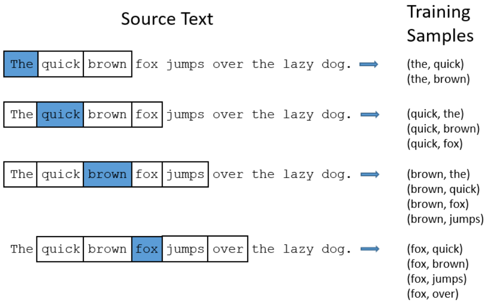
(Firth, 1957)

Similar words tend to occur in similar contexts, therefore a word can be represented based on the co-occurrence across the data in the same context.

WORD2VEC INPUT

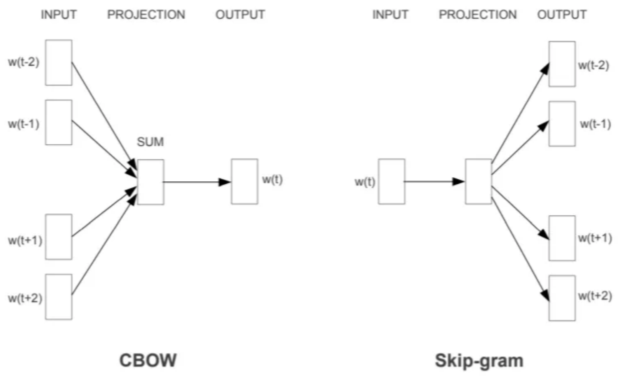


[1]



[1] http://web.stanford.edu/class/cs224n/

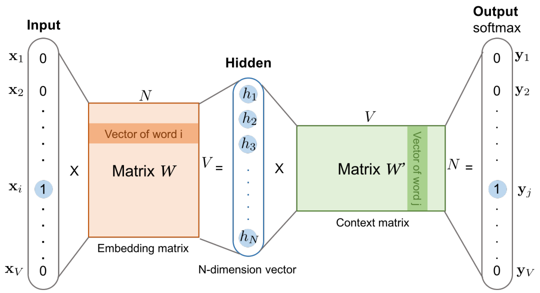
WORD2VEC: CBOW VS SKIP-GRAM



POLIMI GRADUATE SCHOOL OF MANAGEMENT

WORD2VEC SKIP-GRAM

A single-layer architecture based on the inner product between two word vectors.

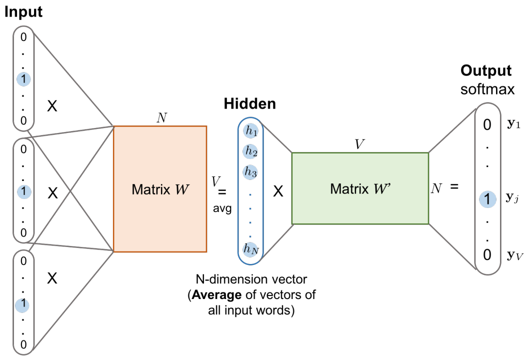


$$\underbrace{e_i^\top \times W_{N\times d}}_{h} \times W'_{d\times N} \xrightarrow{\text{softmax}} \mathbb{P}(\text{word}_j|\text{word}_i)$$

We maximize $\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq k\leq c, k\neq 0} log(p(w_{t+k}|w_t))$ where $p(o|c) = exp(v_o^\top v_c)/\sum_{w=1}^{V} exp(u_w^\top v_C)$

https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html

WORD2VEC CBOW



$$h = \frac{1}{|\text{window}|} \sum_{i=1}^{|\text{window}|} e_i^\top \times W_{N\times d}$$

https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html

GLOVE - GLOBAL VECTORS

- We use the co-occurrence matrix for the entire corpus.
- X_{ij}: # of times word j is in the context of word i.
- $P(j|i) = \frac{X_{ij}}{X_i}$ with $X_i = \sum_l X_{il}$, probability that j is in the context of i

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

COSINE SIMILARITY

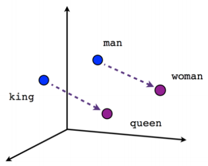
Similarity distance measure as the cosine similarity:

$$similarity(a,b) = \frac{a^\top b}{||a||\,||b||} = cos(\theta)$$

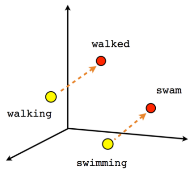
ANALOGICAL REASONING

The city of Rome is in relation with the country Italy in the same way as the city of Paris is in relation with the country France.

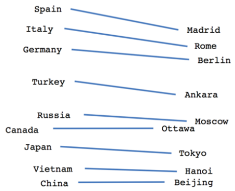
The propositional analogy task: find an x such that Rome : Italy = x : France



Male-Female



Verb tense



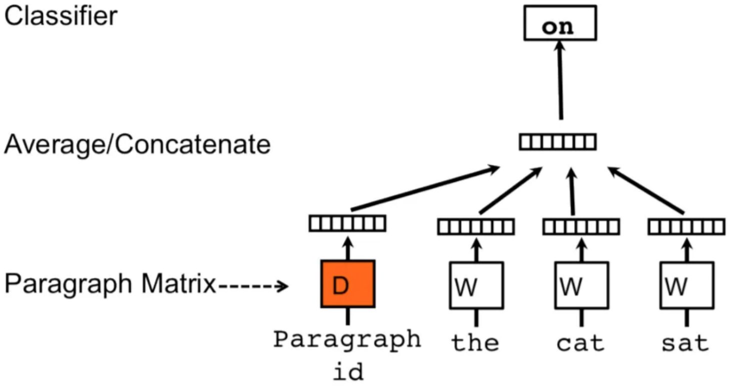
Country-Capital

$$v(Italy) - v(Rome) \sim v(France) - v(Paris)$$

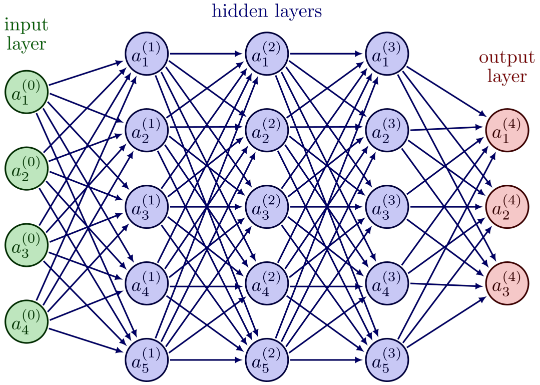
IMPLEMENTATIONS

- Neural Network: Word2Vec [Mikolov+, 2013], ELMO [Peters+,2018]
  `https://code.google.com/archive/p/word2vec`
- GloVe [Pennington+,2014]. Matrix Factorization/Neural Network.
  `https://nlp.stanford.edu/projects/glove/`

DOC2VEC

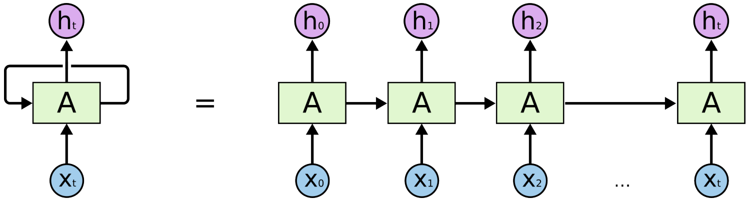

Classifier
on
Average/Concatenate
Paragraph Matrix----->
D
W
W
W
Paragraph id
the
cat
sat

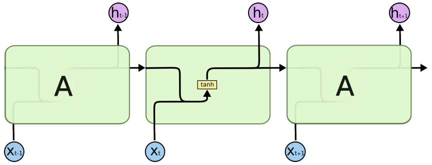
FAST FORWARD NN



RNN: RECURRENT NEURAL NETWORKS

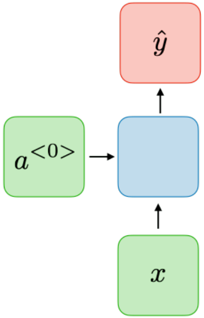
Given a sequence (of words): $x = x_1 x_2 \cdots x_t$



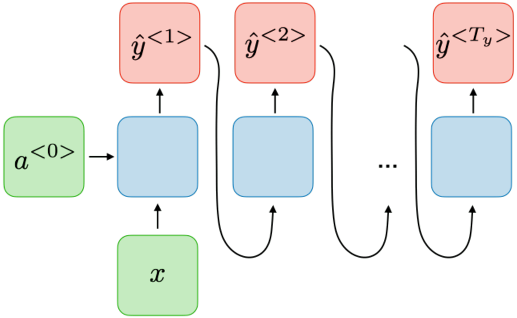


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN: RECURRENT NEURAL NETWORKS

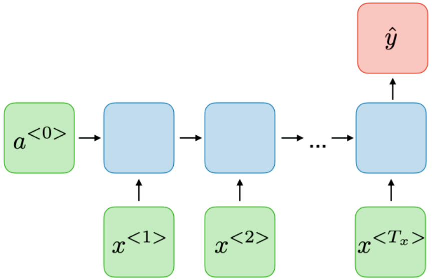


Traditional NN

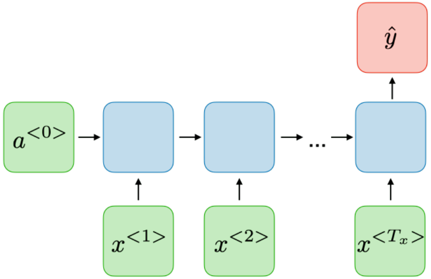


Music generation

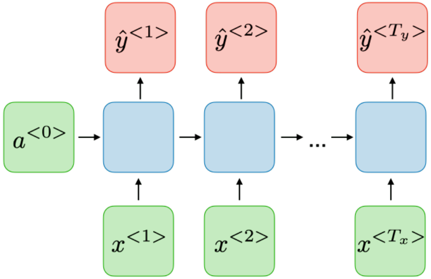
RNN: RECURRENT NEURAL NETWORKS



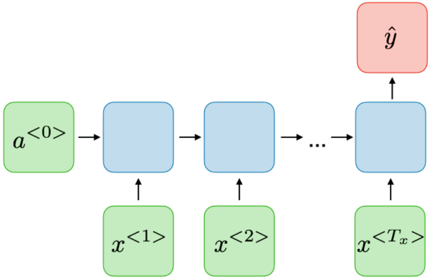
RNN: RECURRENT NEURAL NETWORKS



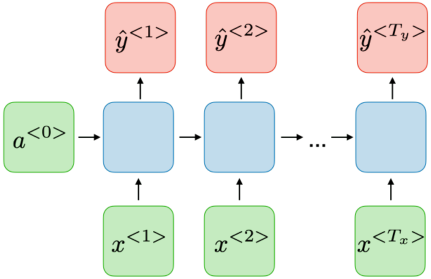
Sentiment classification



RNN: RECURRENT NEURAL NETWORKS

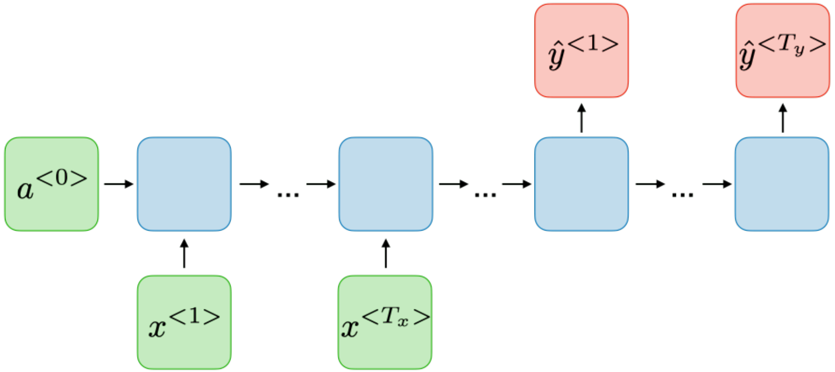


Sentiment classification

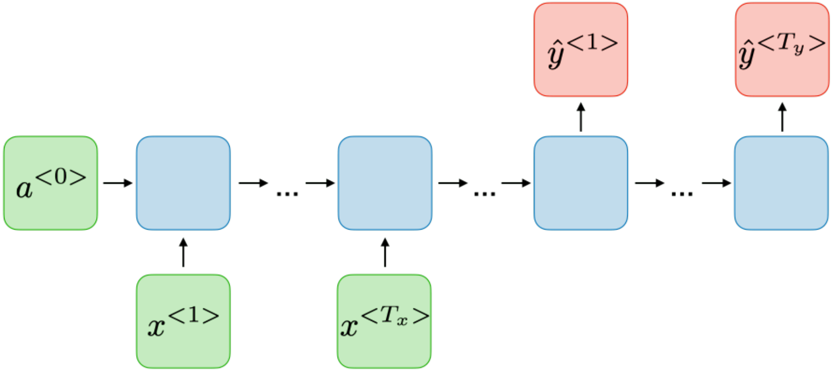


Name entity recognition

RNN: RECURRENT NEURAL NETWORKS



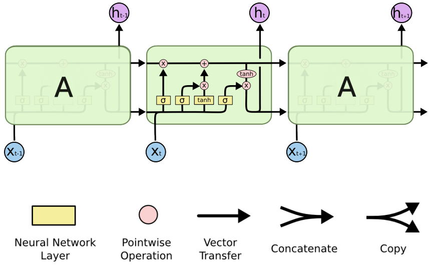
RNN: RECURRENT NEURAL NETWORKS



Text translation

LSTM: LONG SHORT-TERM MEMORY NETWORKS

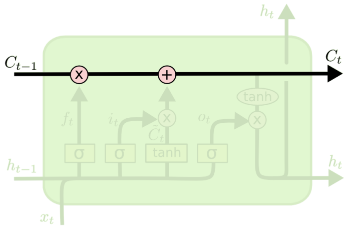
1. We keep a cell state across the sequence C_t
2. After each step t we:
    - forget something: f_t
    - include something : i_t
    - update the cell state: C_t
    - output something to the next step: h_t



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

LSTM: KEEP GLOBAL STATE

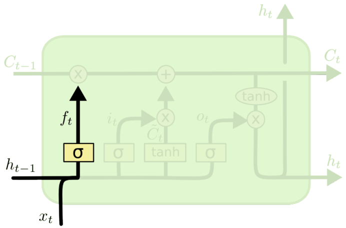
1. **We keep a cell state across the sequence C_t**
2. After each step t we:
   - forget something: f_t
   - include something : i_t
   - update the cell state: C_t
   - output something to the next step: h_t



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

LSTM: FORGET GATE STATE

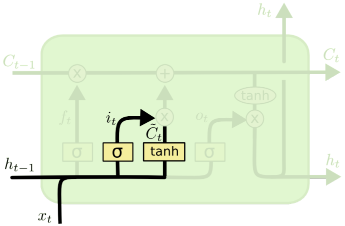
1. We keep a cell state across the sequence C_t
2. After each step t we:
   - **forget something**: f_t
   - include something : i_t
   - update the cell state: C_t
   - output something to the next step: h_t



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

LSTM: INPUT GATE STATE

1. We keep a cell state across the sequence C_t
2. After each step t we:
   - forget something: f_t
   - **include something** : i_t
   - update the cell state: C_t
   - output something to the next step: h_t

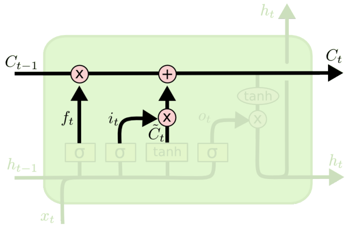


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

LSTM: UPDATE CELL STATE

1. We keep a cell state across the sequence C_t
2. After each step t we:
   - forget something: f_t
   - include something : i_t
   - **update the cell state**: C_t
   - output something to the next step: h_t

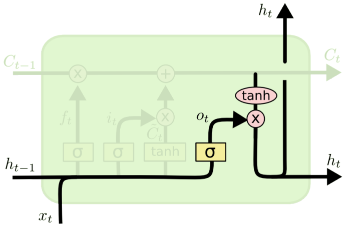


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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

LSTM: CELL OUTPUT

1. We keep a cell state across the sequence C_t
2. After each step t we:
   - forget something: f_t
   - include something : i_t
   - update the cell state: C_t
   - **output something to the next step**: h_t



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

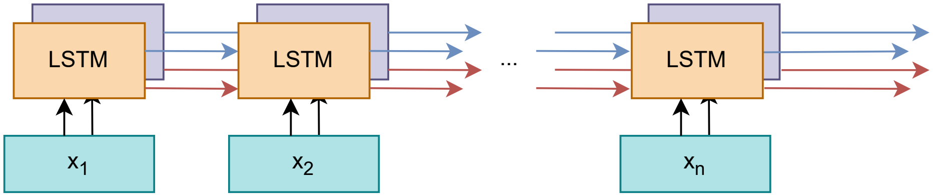
$$h_t = o_t * \tanh\left(C_t\right)$$

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

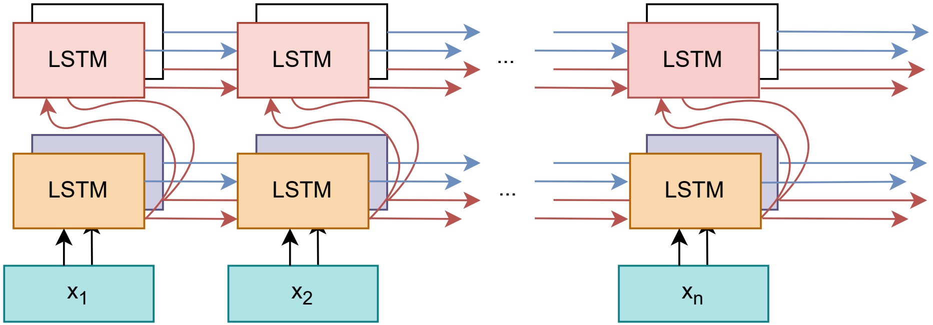
GENERATING TEXT

1. From the text, we create a training set form by couples $([x_1, \ldots, x_t], y_t)$ where:
    - $[x_1, \ldots, x_t]$ is a sequence of t elements (letters, words)
    - y_t is the element to be predicted
2. From a seed sequence we sequentially generate the text consider as input the last sequence.

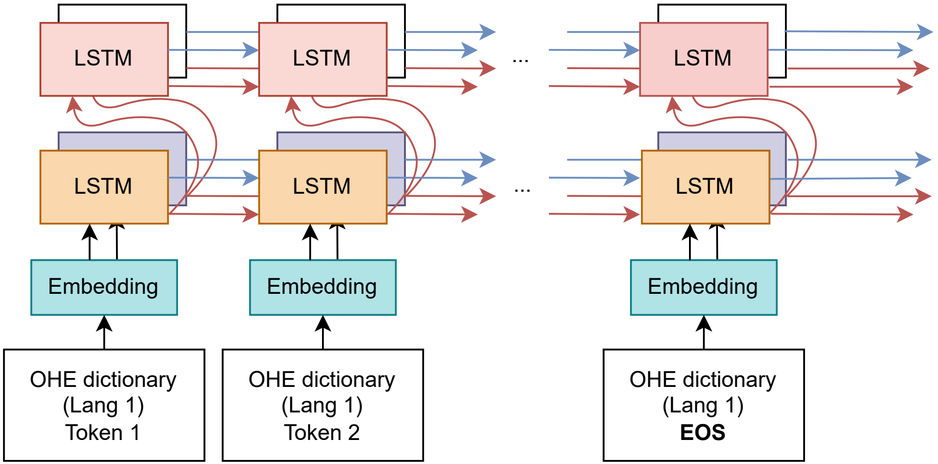
EXTENDING LSTM



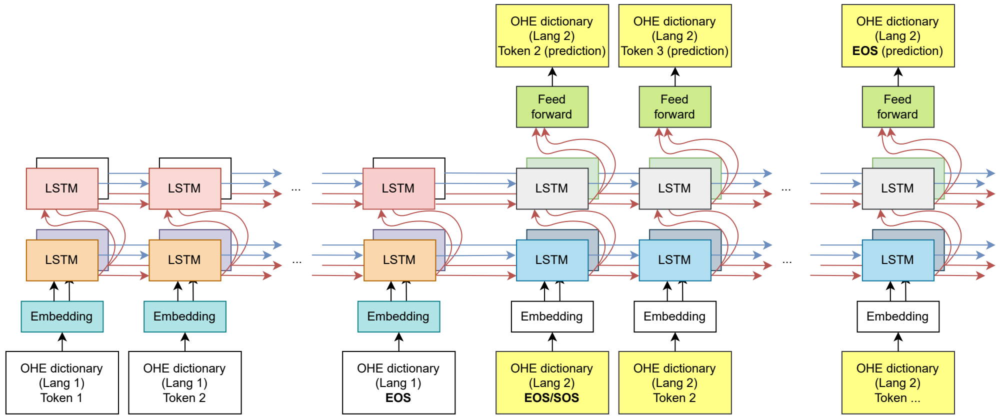
EXTENDING LSTM



EXTENDING LSTM

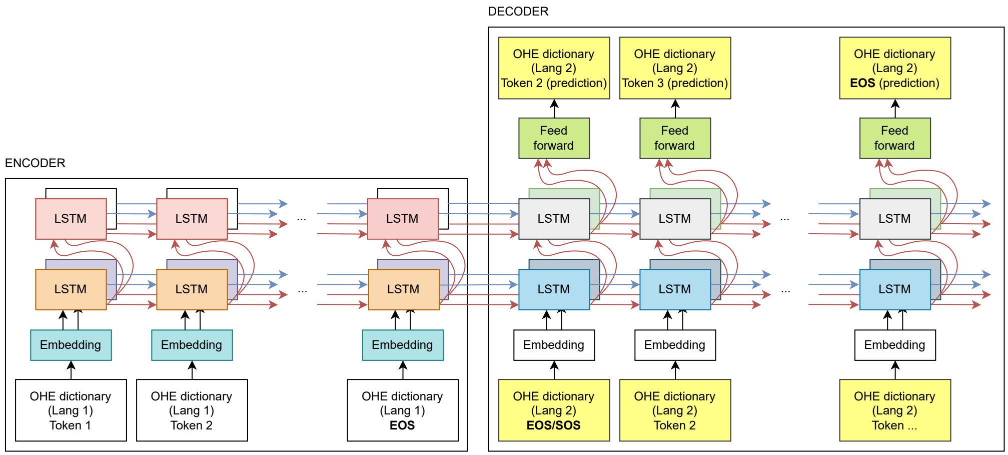

LSTM
LSTM
...
LSTM
LSTM
LSTM
...
LSTM
Embedding
Embedding
Embedding
OHE dictionary (Lang 1) Token 1
OHE dictionary (Lang 1) Token 2
OHE dictionary (Lang 1) **EOS**

THE ENCODER-DECODER PARADIGM OF SEQ2SEQ


OHE dictionary (Lang 2) Token 2 (prediction)
OHE dictionary (Lang 2) Token 3 (prediction)
OHE dictionary (Lang 2) **EOS** (prediction)
Feed forward
Feed forward
Feed forward
LSTM
LSTM
...
LSTM
LSTM
LSTM
...
LSTM
LSTM
LSTM
...
LSTM
LSTM
LSTM
...
LSTM
Embedding
Embedding
Embedding
Embedding
Embedding
Embedding
OHE dictionary (Lang 1) Token 1
OHE dictionary (Lang 1) Token 2
OHE dictionary (Lang 1) **EOS**
OHE dictionary (Lang 2) **EOS/SOS**
OHE dictionary (Lang 2) Token 2
OHE dictionary (Lang 2) Token ...

POLIMI GRADUATE SCHOOL OF MANAGEMENT

THE ENCODER-DECODER PARADIGM OF SEQ2SEQ



DECODER
OHE dictionary (Lang 2) Token 2 (prediction)
OHE dictionary (Lang 2) Token 3 (prediction)
OHE dictionary (Lang 2) **EOS** (prediction)
Feed forward
Feed forward
Feed forward
ENCODER
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
...
...
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
...
...
Embedding
Embedding
Embedding
Embedding
Embedding
Embedding
OHE dictionary (Lang 1) Token 1
OHE dictionary (Lang 1) Token 2
OHE dictionary (Lang 1) **EOS**
OHE dictionary (Lang 2) **EOS/SOS**
OHE dictionary (Lang 2) Token 2
OHE dictionary (Lang 2) Token ...

NETWORK SIZE

- NN with (8, 10, 5, 1)

151 parameters

- ”Bowie” LSTM (letters)

279’098 parameters

- ”Bowie” LSTM (words)

1’000’000 parameters

- Original seq2seq model:
  - input vocabulary: 160k tokens, output vocabulary 80k tokens;
  - embedding size: 1k values;
  - 4 layers of 1000k LSTM nodes each.

384’000’000 parameters

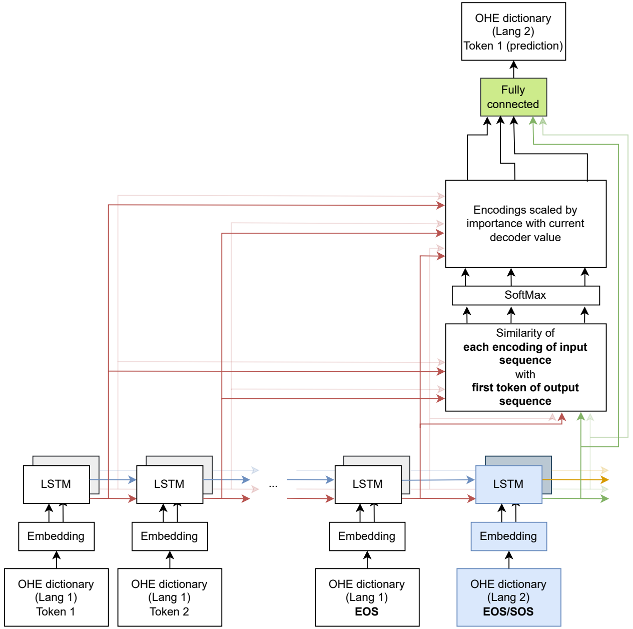
ATTENTION

- The vectors passed from the encoder to the decoder provide what is called **context**.
- LSTM has separate long term and short term vectors.
- The longer the sequence and the broader the dictionary, the more information we are trying to cram into these context vectors.

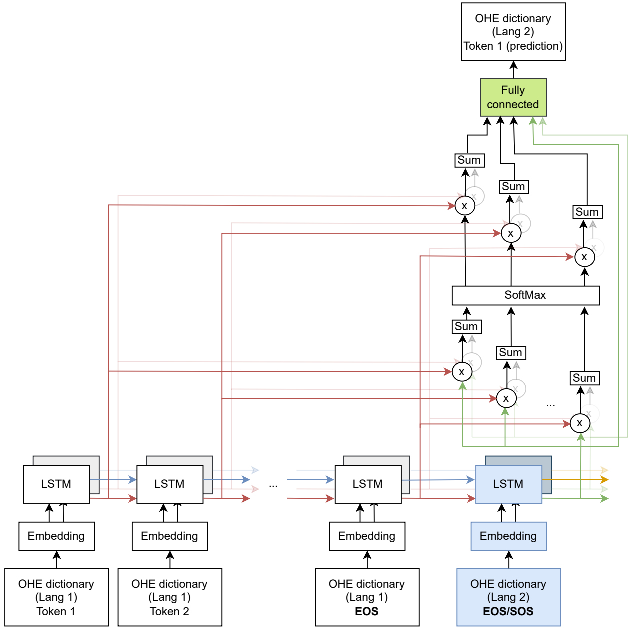
  $\rightarrow$ we might have troubles remember old relevant elements of the sequence.

- What if we could learn what information to bring forward for each element of the sequence? This is called **attention**.

ATTENTION


OHE dictionary (Lang 2) Token 1 (prediction)
Fully connected
Encodings scaled by importance with current decoder value
SoftMax
Similarity of **each encoding of input sequence** with **first token of output sequence**
LSTM
LSTM
...
LSTM
LSTM
Embedding
Embedding
Embedding
Embedding
OHE dictionary (Lang 1) Token 1
OHE dictionary (Lang 1) Token 2
OHE dictionary (Lang 1) **EOS**
OHE dictionary (Lang 2) **EOS/SOS**

ATTENTION



OHE dictionary (Lang 2) Token 1 (prediction)
Fully connected
Sum
Sum
Sum
x
x
x
SoftMax
Sum
Sum
Sum
x
x
x
...
LSTM
LSTM
...
LSTM
LSTM
Embedding
Embedding
Embedding
Embedding
OHE dictionary (Lang 1) Token 1
OHE dictionary (Lang 1) Token 2
OHE dictionary (Lang 1) **EOS**
OHE dictionary (Lang 2) **EOS/SOS**

POLIMI GRADUATE SCHOOL OF MANAGEMENT

TRANSFORMERS

Now that we have all this attention information available:

TRANSFORMERS

Now that we have all this attention information available:

- do we still need the context from LSTMs?

TRANSFORMERS

Now that we have all this attention information available:

- do we still need the context from LSTMs?
- do we still need LSTMs to begin with?

TRANSFORMERS

Now that we have all this attention information available:

- do we still need the context from LSTMs?
- do we still need LSTMs to begin with?

No

TRANSFORMERS

Now that we have all this attention information available:

- do we still need the context from LSTMs?
- do we still need LSTMs to begin with?

No (but)

TRANSFORMERS

Now that we have all this attention information available:

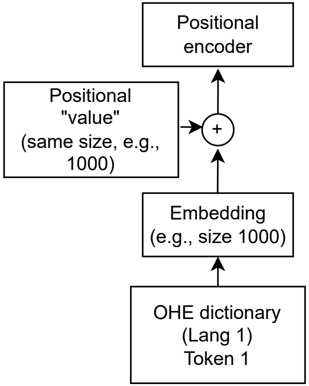
- do we still need the context from LSTMs?
- do we still need LSTMs to begin with?

No (but)

- we still need some way to encode the position in the sequence
  $\rightarrow$ positional values

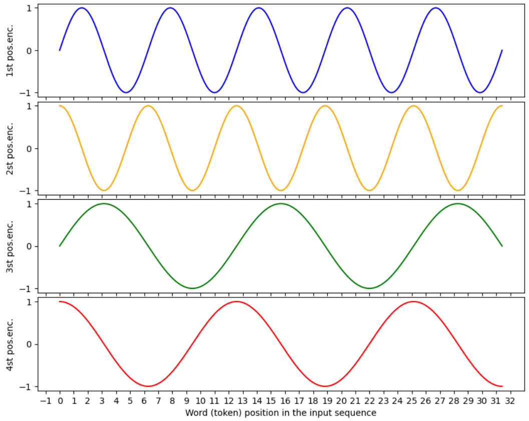
TRANSFORMERS

Let's start from scratch:

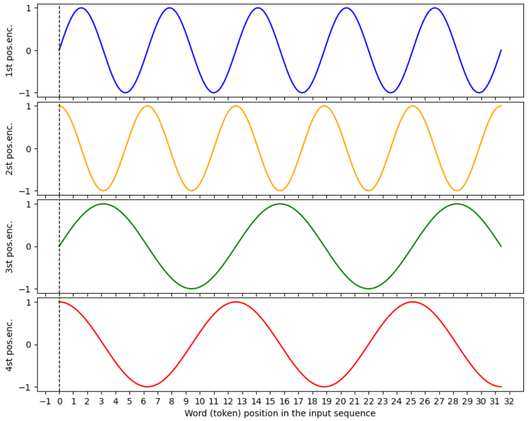


Positional encoder
Positional "value" (same size, e.g., 1000)
+
Embedding (e.g., size 1000)
OHE dictionary (Lang 1) Token 1

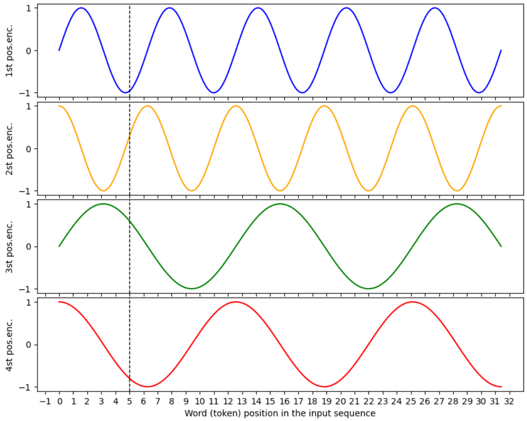
POSITIONAL VALUES (SIZE 4)



POSITIONAL VALUES



POSITIONAL VALUES



POSITIONAL ENCODING

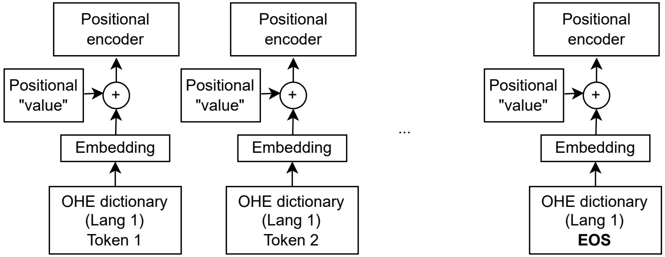
Notice how

- while some values might be equal for two words, for a large enough positional encoder size, the encoder itself is different.

and also that

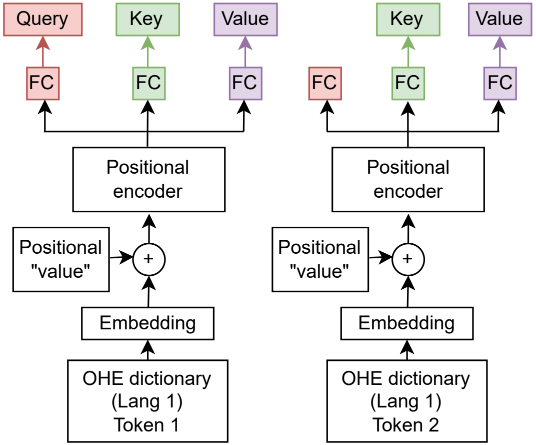
- one word has always the same word embedding
- the word in a specific position has always the same positional encoder
- if a word appears in different positions, the different occurrences will have different positional encodings

SELF ATTENTION IN TRANSFORMERS

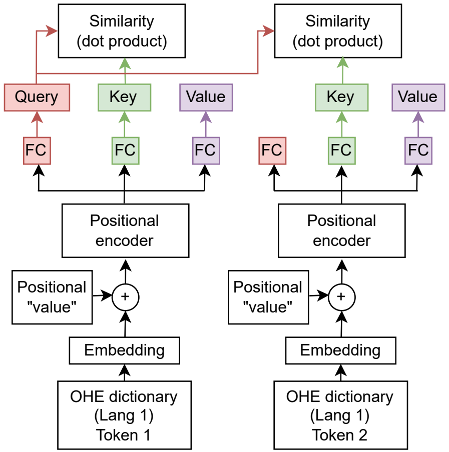


How do we keep track of context if we don’t have LSTM?

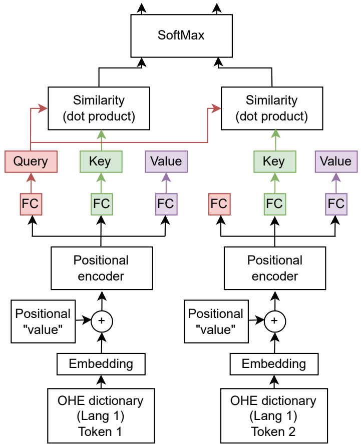
SELF ATTENTION IN TRANSFORMERS



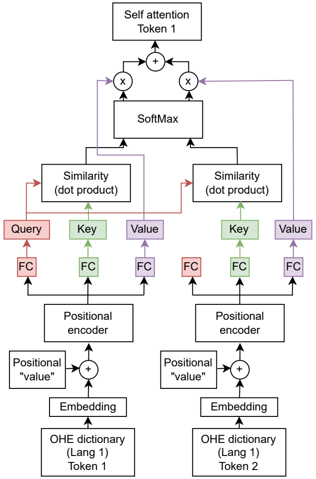
SELF ATTENTION IN TRANSFORMERS



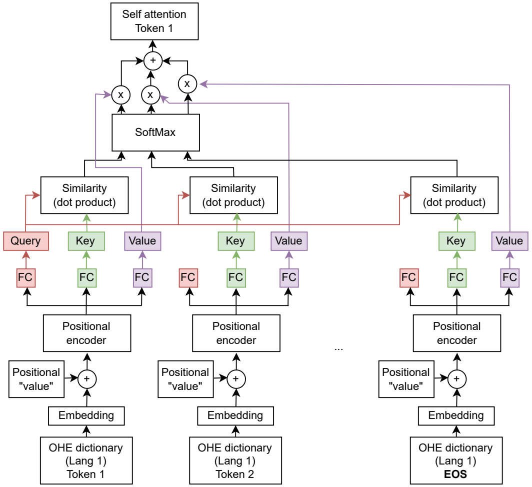
SELF ATTENTION IN TRANSFORMERS



SELF ATTENTION IN TRANSFORMERS



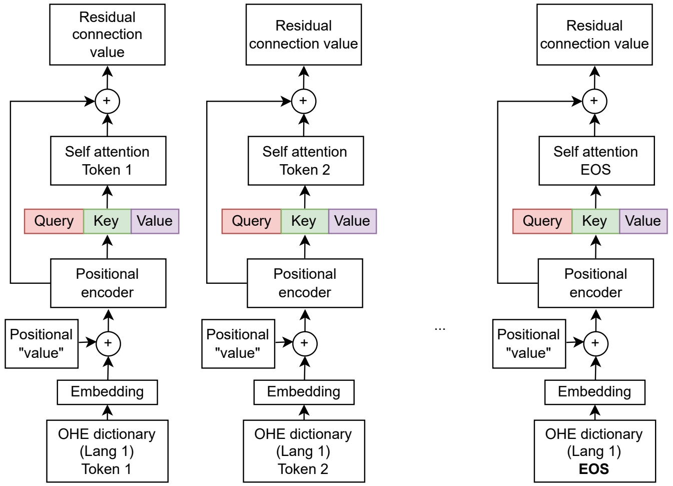
SELF ATTENTION IN TRANSFORMERS



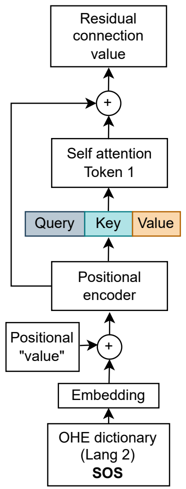
SELF ATTENTION IN TRANSFORMERS

- The weights used to compute Query, Key, and Value for each word are the same.
- This allows for parallel computing, and for the handling of sequences of any length
- The comparison is shown for just two tokens but its made with the keys of “**all**” tokens
- Just like we had multiple LSTMs in parallel, we can have multiple self-attention cells in parallel.

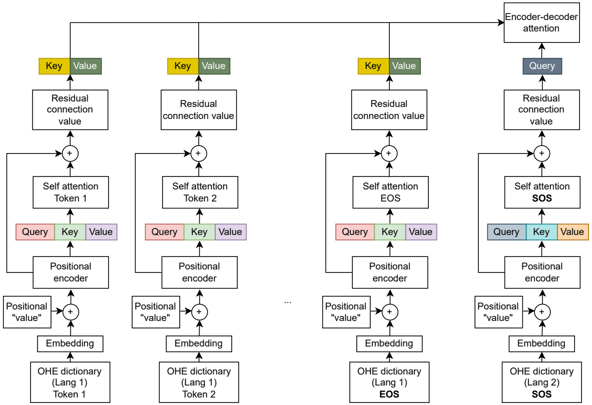
ENCODER IN TRANSFORMERS



DECODER IN TRANSFORMERS



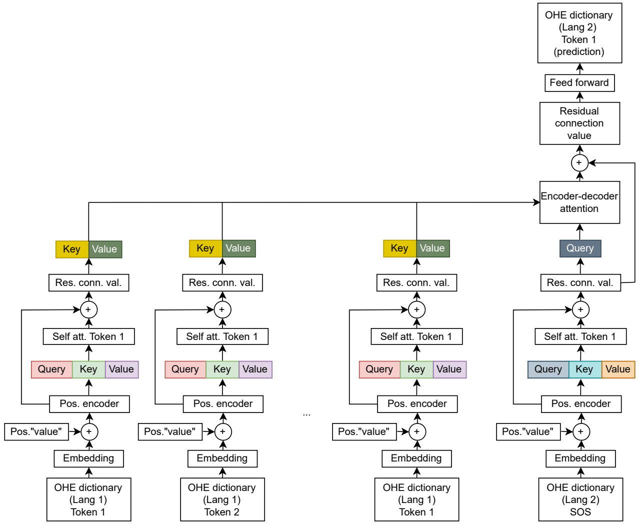
ENCODER/DECODER ATTENTION IN TRANSFORMERS



Encoder-decoder attention
Key
Value
Key
Value
Key
Value
Query
Residual connection value
Residual connection value
Residual connection value
Residual connection value
+
Self attention Token 1
Self attention Token 2
Self attention EOS
Self attention **SOS**
Query
Key
Value
Positional encoder
Positional "value"
Embedding
OHE dictionary (Lang 1) Token 1
OHE dictionary (Lang 1) Token 2
...
OHE dictionary (Lang 1) **EOS**
OHE dictionary (Lang 2) **SOS**

Note that in decoders we often have **masked** self attention.

TRANSFORMERS



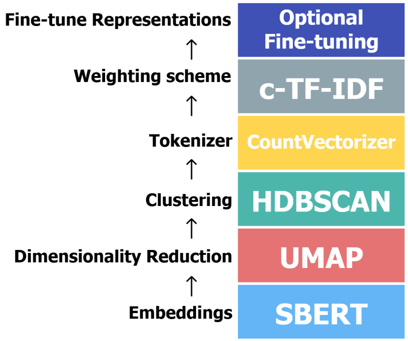
ENCODER-ONLY AND DECODER-ONLY TRANSFORMERS

- To train a encoder-decoder transformer you need to associate an input sequence to an output sequence.
- Turns out that encoders and decoders can work also as separate entities.
- This is an active research area but, very roughly speaking:
    - Encoders: representation for the input.
    - Decoders: generation of the output sequence

  as such:
    - Encoders: representation (embedding) for a known input text, supervised/unsupervised learning based on this representation.
    - Decoders: unsupervised **training**: you only need to predict the next word, hence you don't need (input,output) sequences pairs

ENCODER-ONLY TRANSFORMERS

An example for a familiar application:



THANK YOU