DATA SCIENCE CONSULTING
Session 5

March 6th, 2020

Agenda

1. **Feedback on intermediary restitution**

2. Final presentation

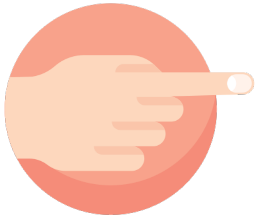
3. Advanced word representations
   A. Word2Vec
   B. FastText
   C. Misspelling Oblivious Word Embeddings

4. Attention Mechanism
   A. RNN, Encoder Decoder, Sequence to Sequence
   B. Attention mechanism and variants
   C. HAN : example of Attention models for Document classification
   D. Openings: Self Attention, Transformer, Language Models

Feedback on intermediary restitution

General comments

- Very satisfying level overall
- Good preparation work (ppt visualization, research, data exploration to answer the business challenge)
- Quality of delivery (storyboarding, presentation skills)



Areas of improvement

- Work on clear presentations' structure
- Explicit use of data research for business purposes
- Choose the right level of explanation of DS techniques for your client
- Add visual advancement percentages to indicate project status
- Do not forget to identify risks you will potentially need to mitigate

Agenda

1. Feedback on intermediary restitution

2. **Final presentation**

3. Advanced word representations
   A. Word2Vec
   B. FastText
   C. Misspelling Oblivious Word Embeddings

4. Attention Mechanism
   A. RNN, Encoder Decoder, Sequence to Sequence
   B. Attention mechanism and variants
   C. HAN : example of Attention models for Document classification
   D. Openings: Self Attention, Transformer, Language Models

Final presentation

You will be presenting to the client **recommendations for the rebranding** of its restaurant & bar **based on the results of your analysis**.

- Remind the **objectives**
- Provide your **methodology**
- **Explicit** the insights based on NLP results, the **trend analysis**
- **Make recommendations** (Suggest a menu, …)
- **Estimate the costs** of your recommendations* , the revenues and the **breakeven**
- Bonus: detail **Next Steps**

**Focus on COGS (Cost of Goods Sold), targeted communication and rebranding pricing*

Workshop – How to structure your presentation?

VISION AND PLANNING

WHY?

Set your strategic goals

- Describe your context?
- What is your business challenge?

WHY NOW?

Why do you want to make a business move now?

- What are the market evolutions?
- How is competition reacting?

WHERE TO PLAY?

- Evaluate competitiveness of current organization, products and business model
- Explore where you can use NLP tools

WHAT?

- Focus on how to create new value propositions for your product portfolio and/or processes
- Prioritize your goals

WHEN?

- Roadmap

CAPABILITIES

HOW?

- Organizational structure & governance
- KPIs
- Workforce
- Description of NLP techniques

SET-UP & RUN

- Implement new operating model?
- Train personnel?
- Set-up partnerships?

DELIVERY

PROTOTYPE & VALIDATE

- Explore data/NLP approaches
- Train, measure, learn & iterate
- Align your stakeholders

RECOMMENDATION

- Define clear recommendations & explicit your argumentation
- Align with the client

Strategic decisions → Define your vision/ambition → Prepare organization → Time for action! → Deliver impact

Workshop – How to build your storyboard?

Storyboarding helps you **structure your argumentation** and build the steps of an **effective communication**.

Design the scenario : write the plan

Design the arguments within the scenario

Produce views checking the fluidity between the messages

Write the introduction and the conclusion

Read it loud

Master the content

Repeat

Manage transitions

Repeat

Respect the time limit

Repeat

Brainstorm

20’

Start to storyboard your final presentation !

Agenda

1. Feedback on intermediary restitution

2. Final presentation

3. **Advanced word representations**
   A. Word2Vec
   B. FastText
   C. Misspelling Oblivious Word Embeddings

4. Attention Mechanism

   A. RNN, Encoder Decoder, Sequence to Sequence

   B. Attention mechanism and variants

   C. HAN : example of Attention models for Document classification

   D. Openings: Self Attention, Transformer, Language Models

Introduction

What need(s) for word/text representations ?

Examples

- Sentence 1 : “The lion eats the gazelle”
- Sentence 2 : “The gazelle eats the lion”

One (among others) problem for bag-of-words representation

- Sentences 1 & 2 have the **same bag-of-words representations while they mean the opposite** !
- One another objective : **capture semantic similarity** and analogies

What applications ?

- Document classification
- Sentiment Analysis
- Question Answering
- Summarization
- Machine Translation
- Named Entity Recognition

With word/text representations and the appropriate modeling, we can address the above tasks

Example of Application : Recommander system (CV screening)

Skill2vec : skill similiarities in a vector space from embeddings learned from 250K CVs

$\overrightarrow{strategy} \approx ?$

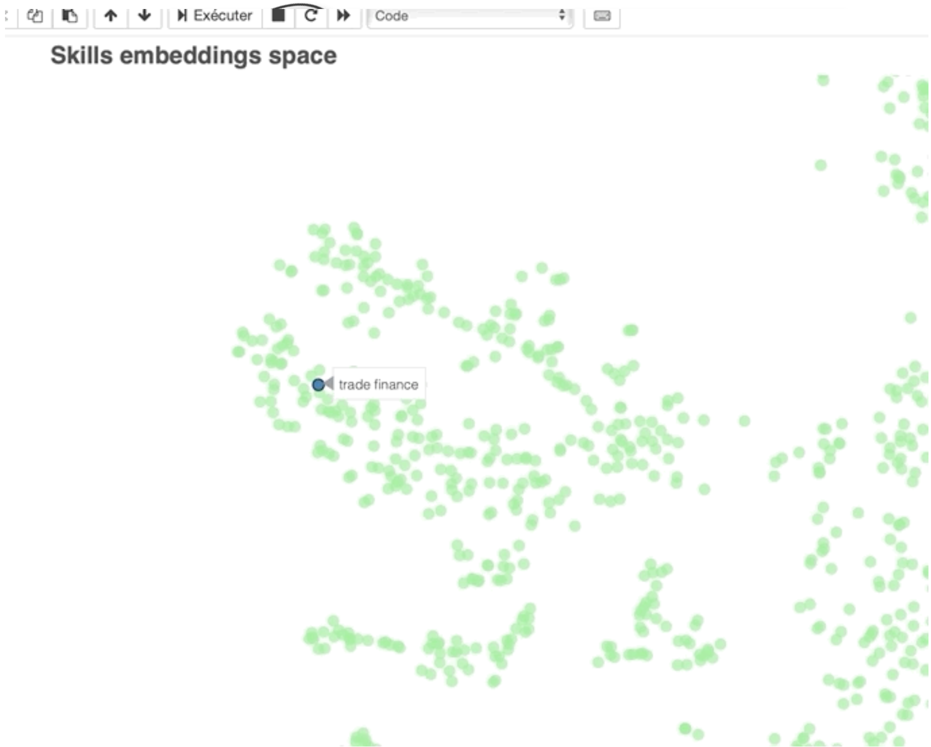
```
: skill2vec.wv.most_similar('strategy', topn=40)

[('strategic consulting', 0.8910848498344421),
 ('small business development', 0.8831638097763062),
 ('competitive intelligence', 0.8769704699516296),
 ('digital transformation', 0.875297069549605),
 ('strategic analysis', 0.8707324266433716),
 ('entrepreneurship', 0.8647657036781311),
 ('business consulting', 0.8591558933258057),
 ('client relationship management', 0.858771562576294),
 ('strategy development', 0.856164813041687),
 ('thought leadership', 0.8527849912643433),
 ('corporate development', 0.8523099422454834),
 ('innovation management', 0.8494170308113098),
 ('strategic partnerships', 0.8475629091262817),
 ('business modeling', 0.8429510593414307),
 ('start-ups', 0.838660717010498),
 ('general management', 0.8316691517829895),
```

$\overrightarrow{it\ strategy} \approx ?$

```
: skill2vec.wv.most_similar('it strategy', topn=40

[('it transformation', 0.9370318651199341),
 ('it outsourcing', 0.9303467869758606),
 ('enterprise architecture', 0.9300419688224792),
 ('solution architecture', 0.9213783740997314),
 ('it service management', 0.9179699420928955),
 ('solution design', 0.912237286567688),
 ('it governance', 0.9074496030807495),
 ('offshoring', 0.9000965356826782),
 ('togaf', 0.8977797031402588),
 ('it project management', 0.8964648246765137),
 ('systems integration', 0.8909523487091064),
 ('business transformation', 0.8880074620246887),
 ('transformation', 0.8867650032043457),
 ('certified project manager', 0.8830263614654541),
 ('global delivery', 0.8817064762115479),
 ('release management', 0.8778325319290161),
 ('consultancy', 0.871580958366394),
 ('service delivery', 0.8680799007415771),
 ('it operations', 0.8670356273651123),
 ('service management', 0.866657018661499),
 ('infrastructure', 0.8637057542800903),
 ('project portfolio management', 0.861572444438934
```



$\overrightarrow{android\ develpment} + \overrightarrow{objectiveC} - \overrightarrow{java} \approx \overrightarrow{ios\ development}$

```
s2v.wv.most_similar(positive=["android development", "objective-c"], negative=["jav

('xcode', 0.8560093641281128),
('ios development', 0.854840099811554),
('game programming', 0.834832489490509)]
```

$\overrightarrow{data\ science} \approx ?$

```
: skill2vec.wv.most_similar('data science', topn=40)

[('machine learning', 0.9702425599098206),
 ('natural language processing', 0.9659499526023865)
 ('deep learning', 0.9507393836975098),
 ('logistic regression', 0.9406247735023499),
 ('neural networks', 0.9269893765449524),
 ('spark', 0.9259625673294067),
 ('linear regression', 0.9214867353439331),
 ('keras', 0.915131688117981),
 ('data visualization', 0.9127278327941895),
 ('tensorflow', 0.905403733253479),
```

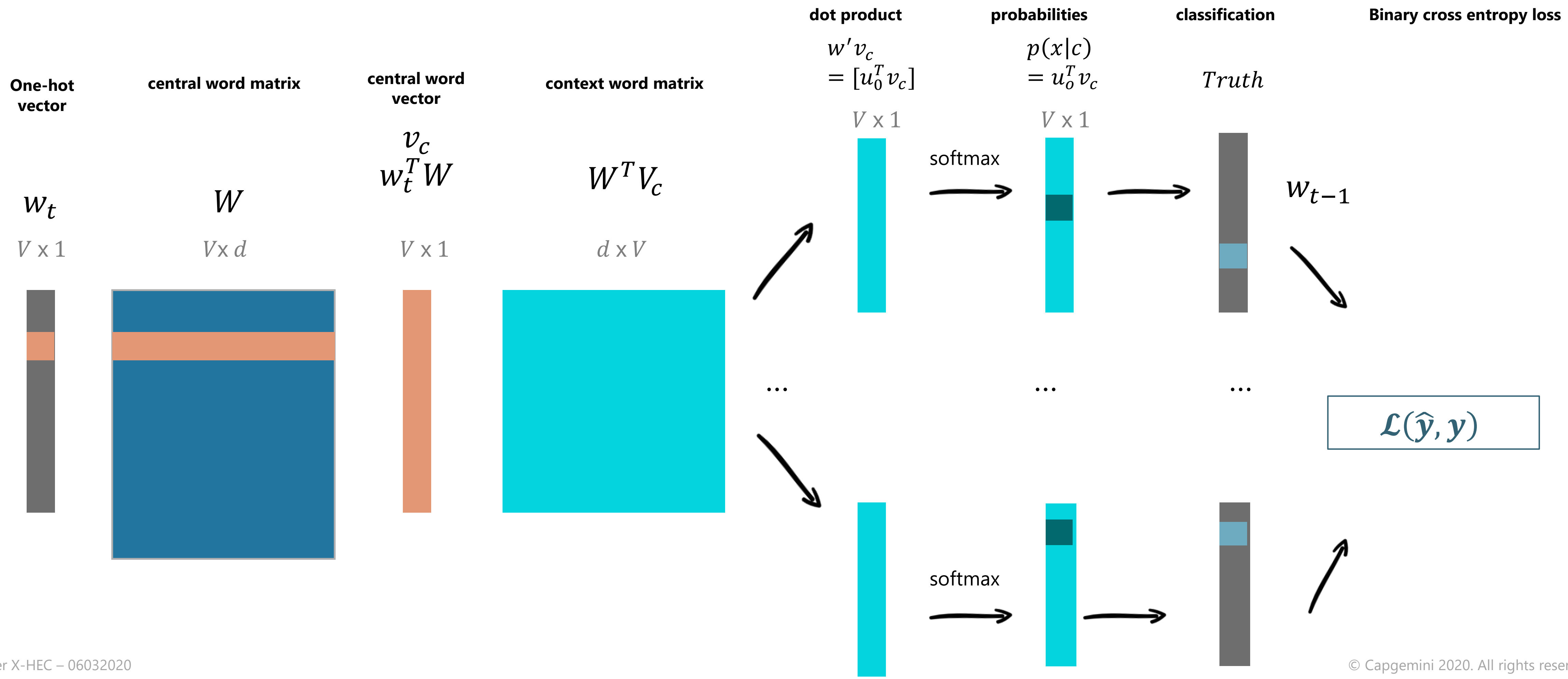
$\overrightarrow{design} \approx ?$

```
: skill2vec.wv.most_similar('design', topn=40)

[('design thinking', 0.874811589717865),
 ('concept development', 0.8679816126823425),
 ('visual design', 0.864403843879697),
 ('rapid prototyping', 0.8643200397491455),
 ('ideation', 0.850453555583953 9),
 ('design strategy', 0.8440836668014526),
 ('user-centered design', 0.8141094446182251),
 ('interaction design', 0.812559962272644),
 ('prototyping', 0.8112373352050781),
 ('service design', 0.8000921010971069),
 ('product design', 0.796481668949127 2),
 ('print design', 0.793822646141052 2),
 ('experience design', 0.791591763496398 9),
 ('user experience design', 0.7888928651809692),
```

Why do we need word vectors in a first place ?

Skipgram



Word2Vec Skipgram

Word2vec

- Reduced continuous (dense) word representations trained on large unlabeled corpora ✅
- Ignores the morphology of words, by assigning a distinct vector to each word ❌
- Represents a limitation for rich languages with large vocabularies and many rare words or misspellings ❌

Skipgram : predict surrounding words from central word

- Given a word vocabulary of size W, where a word is identified by its index $w \in \{1, \dots, W\}$, we want to maximize the log likelihood :

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{c\in C_t}\log p(w_c \mid w_t)$$

where the context C_t is the set of indices of words surrounding w_t

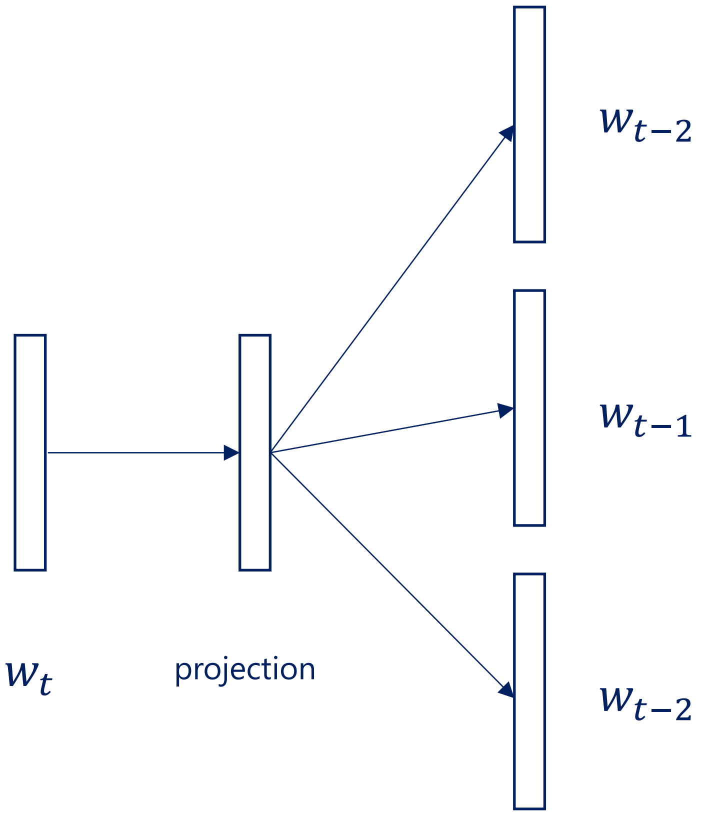
- The basic Skipgram formulation of $p(w_c \mid w_t)$ uses the softmax function :

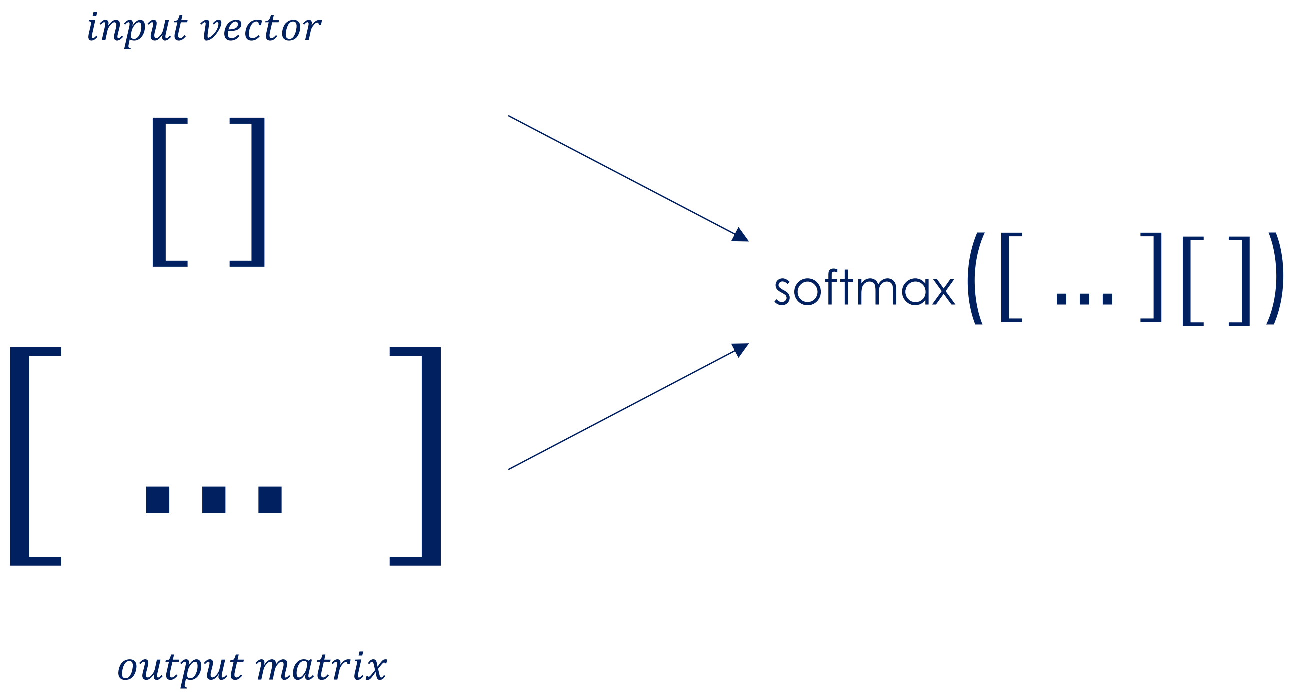
$$p(w_c|w_t) = \frac{e^{s(w_t,w_c)}}{\sum_{j=1}^{W} e^{s(w_t,j)}}$$

Word2Vec Skipgram

- s is a scoring function between a word w_t and a context word w_c
- For each word w, we define two trainable $input$ and $output$ vectors u_w and v_w such that :

$$s(w_t, w_c) = < u_{w_t} \,,\ v_{w_c} > = u_{w_t}^{\top} v_{w_c}$$





Negative Sampling Skipgram

- Problem with previous softmax formulation :
  - Implies to predict only one context word w_c given a word w_t.
  - Computing softmax is very expensive when the number of classes is large.

Alternative : **predict context words as a set of independant binary classification** (presence or absence of a word in context)

- For a word w_t :
  - We consider all context words as positive examples and sample negatives at random from dictionary.
  - For a chosen word context w_c, using the binary logistic loss, we obtain the following negative log-likelihood, where $\mathcal{N}_{t,c}$ is a set of negative examples sampled from the vocabulary :

$$\underbrace{\log\left(1 + e^{-s(w_t, w_c)}\right)}_{\text{Binary loss for true context}} + \sum_{n \in \mathcal{N}_{t,c}} \underbrace{\log\left(1 + e^{s(w_t, n)}\right)}_{\text{Binary losses for negative samples}}$$

- By denoting the logistic loss function $\ell : x \mapsto \log(1 + e^{-x})$, we can re-write the objective as :

$$\frac{1}{T} \sum_{t=1}^{T} \left[\sum_{c \in C_t} \ell\left(s(w_t, w_c)\right) + \sum_{n \in \mathcal{N}_{t,c}} \ell\left(-s(w_t, n)\right) \right]$$

Negative Sampling Skipgram

$$\frac{1}{T}\sum_{t=1}^{T}\Big[\sum_{c\in C_t}\underbrace{\ell(s(w_t,w_c))}_{\text{Binary loss for true context}} + \sum_{n\in\mathcal{N}_{t,c}}\underbrace{\ell(-s(w_t,n))}_{\text{Binary losses for negative samples}}\Big]$$

context word

sliding window

center/target word

Machine learning (ML) is the study of **computer** **algorithms** that improve automatically through experience.

It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as « training data », in order to make predictions or decisions without being explicitly programmed to do so.

negative sample

FastText : Subword Information

Reminder about word2vec limitations :

- Ignores the internal structure of words by assigning a distinct vector to each word ✗
- Represents a limitation for rich languages with large vocabularies and many rare words or misspellings ✗

FastText introduces *Subword Information* :

- Learning word representations from **character n-grams** as the **sum of character n-gram** vectors using a different scoring function s.
- For instance, for the word $where$ with n = 3 :

$$< wh,\ whe,\ her,\ ere,\ re > \text{and} < where >$$

- Consider a dictionary of n-grams of size G and given a word w, we denote $\mathcal{G}_w \subset \{1, \dots, G\}$ the set of n-grams appearing in w. We associate a vector z_g to each n-gram g and represent w word by the sum of the vector representations of its n-grams.
- To learn these n-grams vector representations, we use the following scoring function :

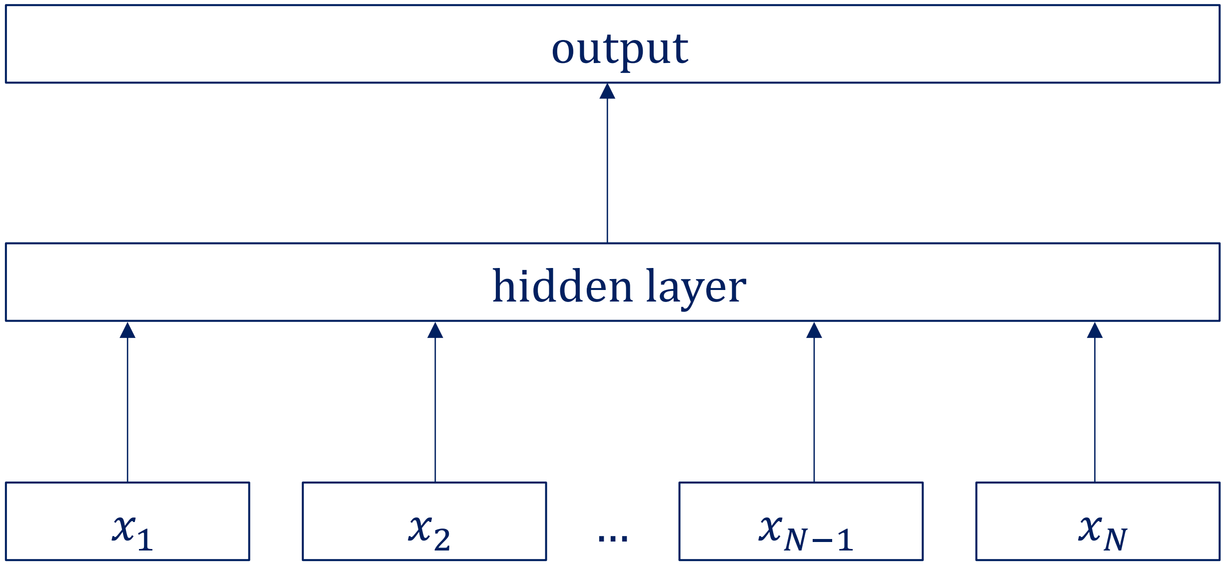
$$s(w, c) = \Sigma_{g \in \mathcal{G}_w} z_g^\top v_c = (\Sigma_{g \in \mathcal{G}_w} z_g)^\top v_c = u_w^\top v_c$$

where c is the context word.

FastText : Efficient text classification from Subword Information embeddings

With FastText

- Words $w_1, \dots, w_N$ and label y (classification)
- Not bag-of-words/vectors (BoW) $\rightarrow$ **bag of n-gram features** to capture partial words order information
- **Weight matrix A** embedding each n-gram features $x_1 \dots x_N$
- **Words representations averaging** to get the hidden text representation
- Linear classifier : **weight matrix B** for linear projection then **softmax-like function** to get probability distribution



FastText : Efficient text classification from Subword Information embeddings

Finally, for D documents we want to minimize the negative log likelihood over the classes :

$$-\frac{1}{D}\sum_{d=1}^{D} y_d \log\big(f(BAx_d)\big)$$

where x_d is the bag of features of the d-th document, y_d the label, A and B the weight matrices and f the softmax-like function.

- **Good performances**
- **Fast training on CPU** using Stochastic Gradient Descent and a linearly decaying learning rate

Another way for approximating softmax when the number of classes is large :

- Hierarchical softmax is an alternative based on Huffman Coding Tree to represent vocabulary
- Explicitly, each node (word) is associated with a probability that is the probability of the path from the root to that node. If the node is at depth $l+1$, with parents $n_1, \dots, n_l$, its probability is :

$$P(n_{l+1}) = \prod_{i=1}^{l} P(n_i)$$

Another Softmax Approximation : Hierarchical Softmax

- Uses a binary tree (Huffman Coding Tree) representation of the output layer (the entire vocabulary) with the W words as its leaves and, for each node, explicitly represents the relative probabilities of its child nodes.

- These defines a random walk that assigns probabilities to words

- Each word w can be reached by an appropriate path from the root of the tree. Let $n(w,j)$ be the j-th node on the path from the root to w and $L(w)$ the length of the path, so $n(w,1)$ = root and $n(w,L(w))$ = w.

- For any inner node n, let $ch(n)$ be an arbitrary fixed child of n and $[x]$ = 1 if x is true and -1 otherwise. Thus, hierarchical softmax defines $p(w|w_I)$ as follows :

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma\left(\left[n(w,j+1) = ch\left(n(w,j)\right)\right].{v'_{n(w,j)}}^T v_{w_I}\right)$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ and which sums to 1.

Misspelling Oblivious Word Embeddings

Misspelling Oblivious Word Embeddings

- Objective : learn word embeddings **more resilient to misspelling** and out-of-vocabulary words than FastText
- General ideas :
  - Training embedding model using the text corpus and misspellings datasets
  - Adding loss function term to push the misspelled words vectors close to the correct words vectors

We denote the FastText loss function with L_{FT} and introduce the spell correction loss L_{SC}

$$L_{SC} \coloneqq \Sigma_{(w_m, w_e) \in M} \left[\ell\big(\hat{s}(w_m, w_e)\big) + \Sigma_{w_n \in \mathcal{N}_{m,e}} \ell\big(-\hat{s}(w_m, w_n)\big)\right]$$

- where M is a set of pairs of words (w_m, w_e) such that w_e is the correctly spelled word and w_m its misspelling.
- $\mathcal{N}_{m,e}$ is a set of random negative samples from $V \backslash \{w_m, w_e\}$
- **Intuitively, optimizing LSC pushes the representation of a misspelling $\boldsymbol{w_m}$ closer to the representation of the expected word $\boldsymbol{w_e}$.**

Misspelling Oblivious Word Embeddings

- We define the scoring function $\hat{s}$ as follows :

$$\hat{s}(w_m, w_e) = \Sigma_{v_g, g \in \hat{\mathcal{G}}_{w_m}} v_g^\top v_e$$

where $\hat{\mathcal{G}}_{w_m} := \mathcal{G}_{w_m} \backslash \{w_m\}$.

- Finally, the loss function L_{MOE} is defined as following :

$$L_{MOE} := \underbrace{\boxed{(1 - \alpha) L_{FT}}}_{\text{FastText Loss}} + \underbrace{\boxed{\alpha \frac{|W|}{|M|} L_{SC}}}_{\text{Spell Correction Loss}}$$

where :

- α is an hyperparameter which sets the importance of the spell correction loss L_{SC} with respect to L_{FT}
- L_{SC} is scaled with factor $\frac{|W|}{|M|}$ for training (stochastic gradient descent)

Break

20’

See you at 15h50 !

Hands-on 1

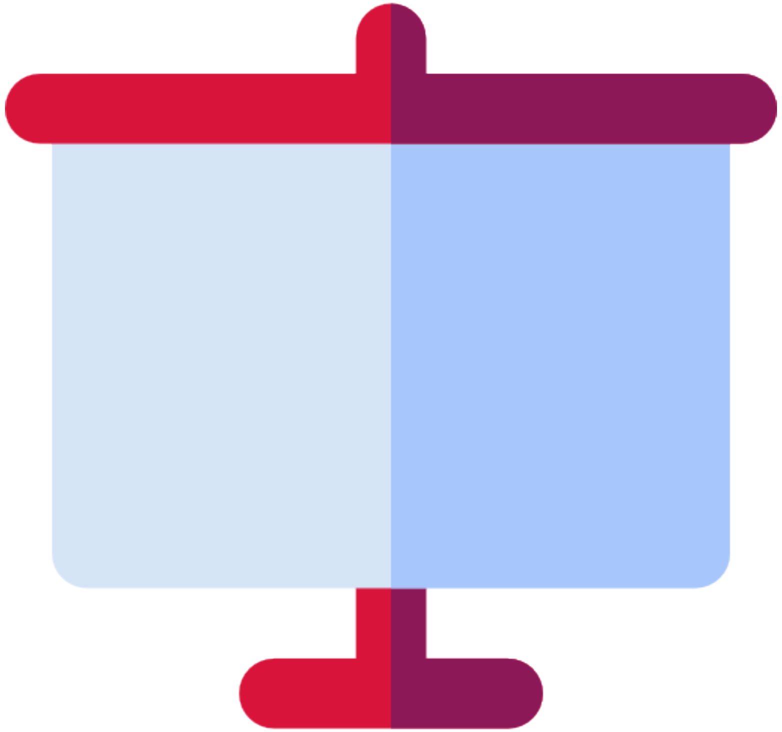
45’

Skipgram with negative sampling using TensorFlow and classification using trained word embeddings

Restitution

5’

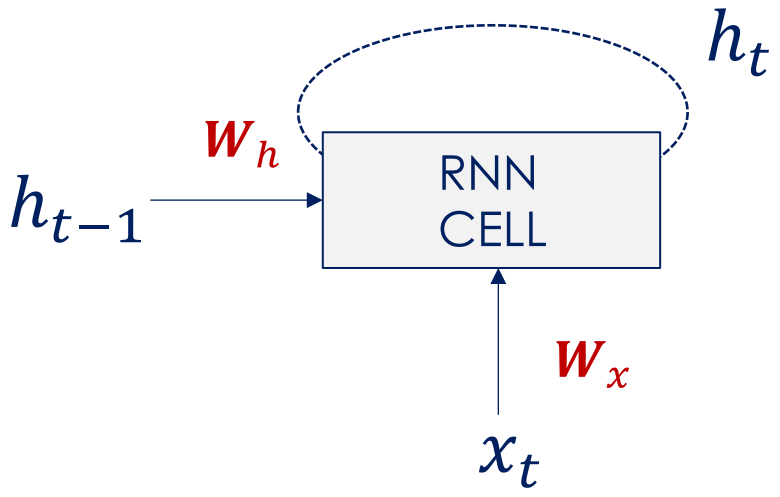
Could you implement Word2Vec ?
What results did you manage to get for the classification task ?



Reccurent Neural Networks (RNN)

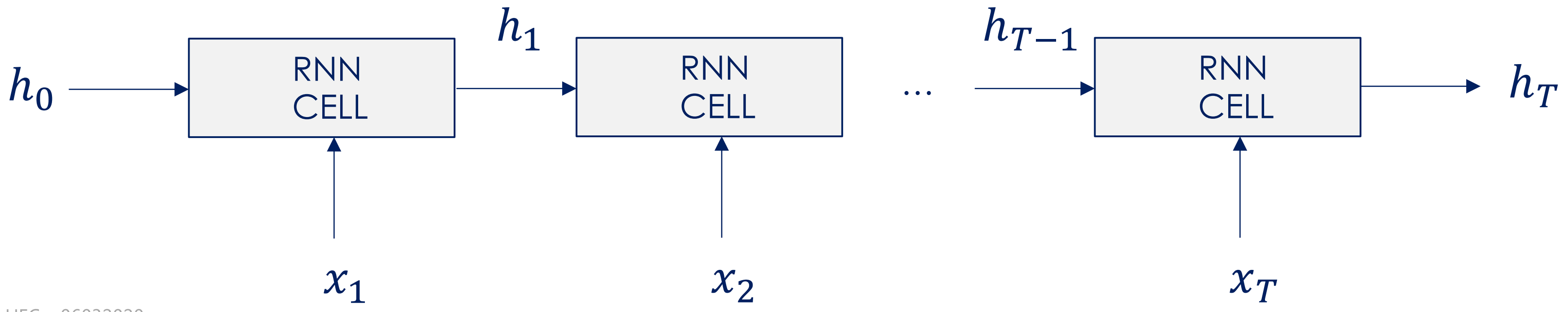
- Consider a variable-length sequence x $= (x_1, \dots, x_T)^\top$ such that $x_t \in \mathbb{R}^N$ and representing the sequence of words $(w_1, \dots, w_T)$
- A simple recurrent neural network encodes such sequences at each step t in the following way :

$$h_t = f(h_{t-1}, x_t) = \tanh(\boldsymbol{W_h}\, h_{t-1} + \boldsymbol{W_x}\, x_t)$$

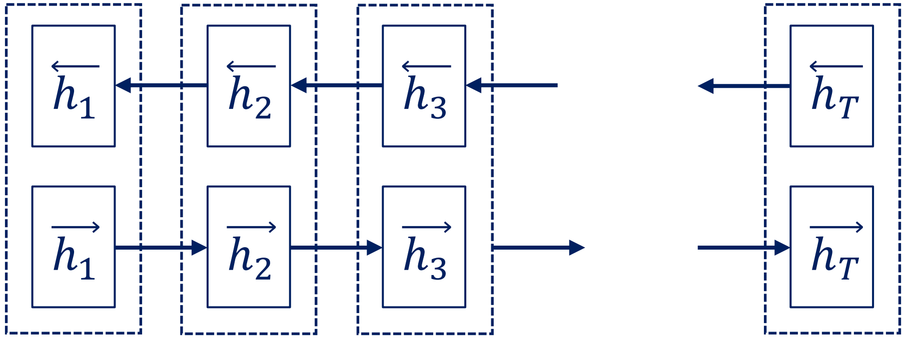


where :

- x_t is the word embedding of the word w_t
- $\boldsymbol{W_h} \in \mathbb{R}^{D\times D}, \boldsymbol{W_x} \in \mathbb{R}^{D\times N}, h_t \in \mathbb{R}^D$
- f is called the RNN Cell
- **h** $= (h_1, \dots, h_T)^\top$ are called hidden states
- Simple RNNs are struggling to capture long term dependency from the input sequence
- More sophisticated RNN cells : **Long-Short Term Memory** (LSTM), **Gated Recurrent Units** (GRU)



Reccurent Neural Networks (RNN)



Principles

- Handle variable-length sequence
- Preserve word order
- Capture sequential information
- Encode sequence/sentence information in continuous vectors

Bidirectional Recurrent Neural Networks

- Concatenation of forward and backward RNN's hidden states to build word representation containing the summaries of both the preceding words and the following words
- The forward RNN $\vec{f}$ reads the input sequence from x_1 to x_T and builds the forward hidden states $\left(\overrightarrow{h_1}, \dots, \overrightarrow{h_T}\right)$
- The backward RNN $\overleftarrow{f}$ reads the reverse input sequence from x_T to x_1 and builds the backward hidden states $\left(\overleftarrow{h_1}, \dots, \overleftarrow{h_T}\right)$

- Finally, get $(h_1, \dots, h_T)$ such that $h_j = \left[\overrightarrow{h_j}\, ;\, \overleftarrow{h_j}\right]$

Reccurent Neural Networks (RNN)

Use RNN to learn a probability distribution over a sequence (e.g. language modeling) :

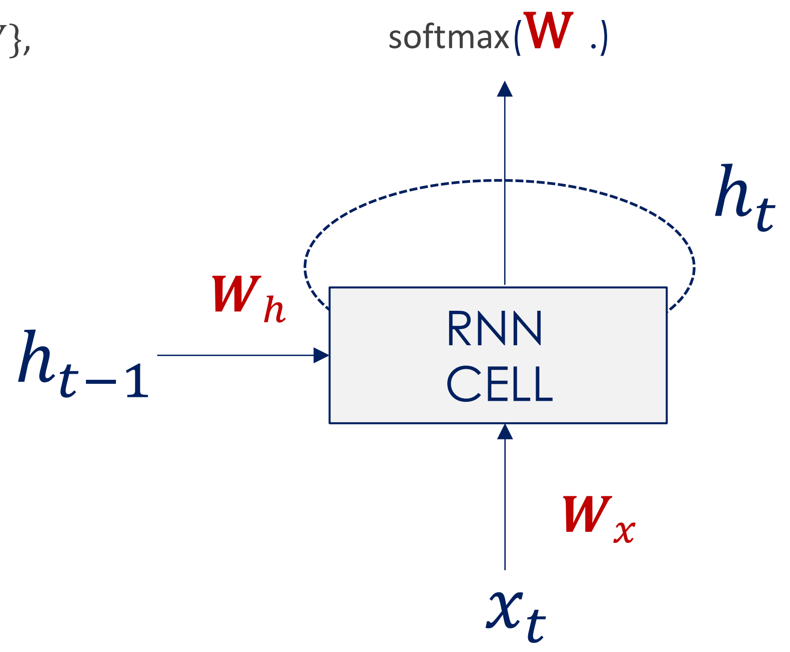
- Train the network to predict the next symbol in the sequence $p(x_t \mid x_{t-1}, \dots, x_1)$. For $j \in \{1, \dots, W\}$,

$$p(x_{t,j} \mid x_{t-1}, \dots, x_1) = \frac{\exp(\mathrm{w}_j\, \mathrm{h}_t)}{\sum_{j'=1}^{W} \exp(\mathrm{w}_{j'}, \mathrm{h}_t)}$$

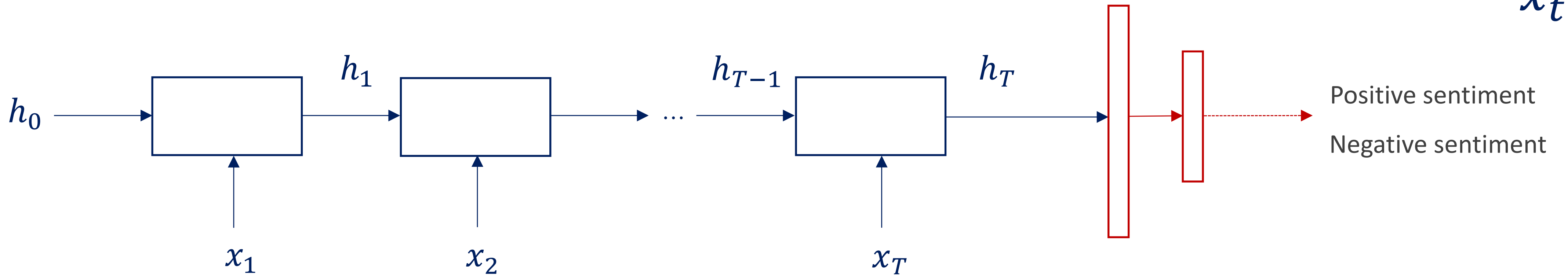
where w_j are the rows of the weight matrix **W** $\in \mathbb{R}^{W \times D}$

- By combining these probabilities, we can compute the probability of the **sequence x** :

$$p(\mathrm{x}) = \prod_{t=1}^{T} p(x_t \mid x_{t-1}, \dots, x_1)$$



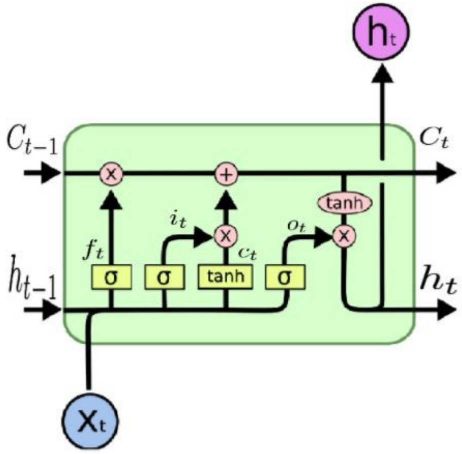
Use RNN for sequence classification (e.g. sentence classification)



Long Short Term Memory (LSTM)

Type of RNN as a solution to preserve information over many time steps.

On step t, thiere is a hidden state h, and a cell state $\boldsymbol{c}$: Both are vectors length n. The LSTM can erase, write and read information from the cell



LSTM Cell

The selection of which infromation is erased/written/reas is controlled by three corresponding gates

- The gates are also vectors length n
- On each timestep, each element of the gates can be open (1), closed (0), or somewhere in between

$\boldsymbol{f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)}$	**Forget gate**: controls what should be kept or forgotten from previous cell state
$\boldsymbol{i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)}$	**Input gate**: controls what part of the new cell content are written to cell
$\boldsymbol{o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)}$	**Output gate**: controls what parts of cell are output to hidden state
$\boldsymbol{\tilde{c}_t = \tanh(W_c h_{t-1} + U_c x_t + b_c)}$	**New cell content** :new content to be written to the cell
$\boldsymbol{c_t = f^t \circ c^{t-1} + i^t \circ \tilde{c}_t}$	**Cell state**: forget some content from last cell state, and input some new cell content
$\boldsymbol{h_t = o^t \circ \tanh c_t}$	**Hidden state**: output some content from the cell

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

Agenda

1. Feedback on intermediary restitution

2. Final presentation

3. Advanced word representations
    A. Word2Vec
    B. FastText
    C. Misspelling Oblivious Word Embeddings

4. **Attention Mechanism**

    A. RNN, Encoder Decoder, Sequence to Sequence

    B. Attention mechanism and variants

    C. HAN : example of Attention models for Document classification

    D. Openings: Self Attention, Transformer, Language Models

RNN Encoder-Decoder

- Decoder : the RNN Decoder is trained to predict the next word y_t given the context c and all the previously predicted words $\{y_1, \dots, y_{t-1}\}$.
- The decoder defines a probability over the translation y by decomposing the joint probability into the ordered conditionals :

$$p(\mathrm{y}) = \Pi_{t=1}^{T'} p(y_t \mid \{y_1, \dots, y_{t-1}\}, c)$$

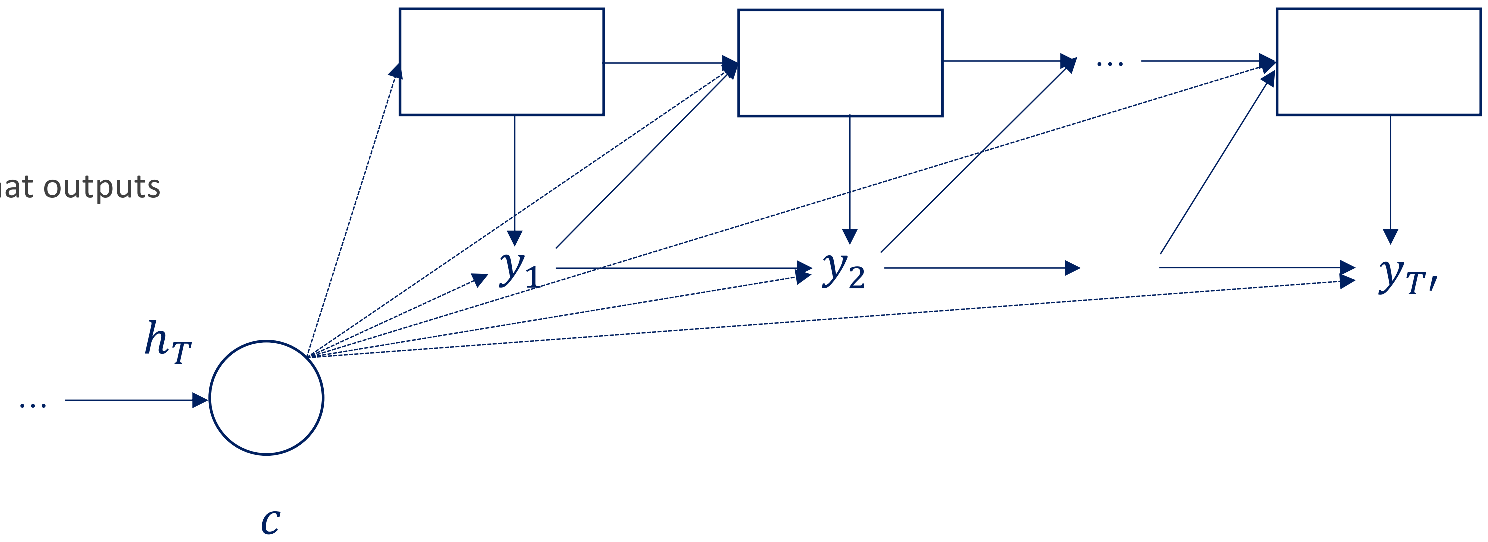
such that :

$$p(y_t \mid \{y_1, \dots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

$$s_t = f_{Decoder}(s_{t-1}, y_{t-1}, c)$$

where :

- $\mathbf{y} = (y_1, \dots, y_{T'})$
- s_t the hidden state of the RNN Decoder
- $g(.)$ a nonlinear function, potentially multi-layered, that outputs
- the probability of y_t



Attention Mechanism

- In the RNN Decoder, attention mechanism allows to build for each time step t a more relevant context vector c_t

$$p(y_t \mid \{y_1, \dots, y_{t-1}\}, \mathrm{x}) = g(y_{t-1}, s_t, c_t)$$

where s_i is the RNN hidden state for time i, computed by :

$$s_t = f_{Decoder}(s_{t-1}, y_{t-1}, c_t)$$

- Attention scores, can be computed with several variants, here this is a simple dot product between each hidden state h_i and decoder hidden state at t, s_t

$$e_t = [s_t^T h_1, \dots, s_t^T h_N]$$

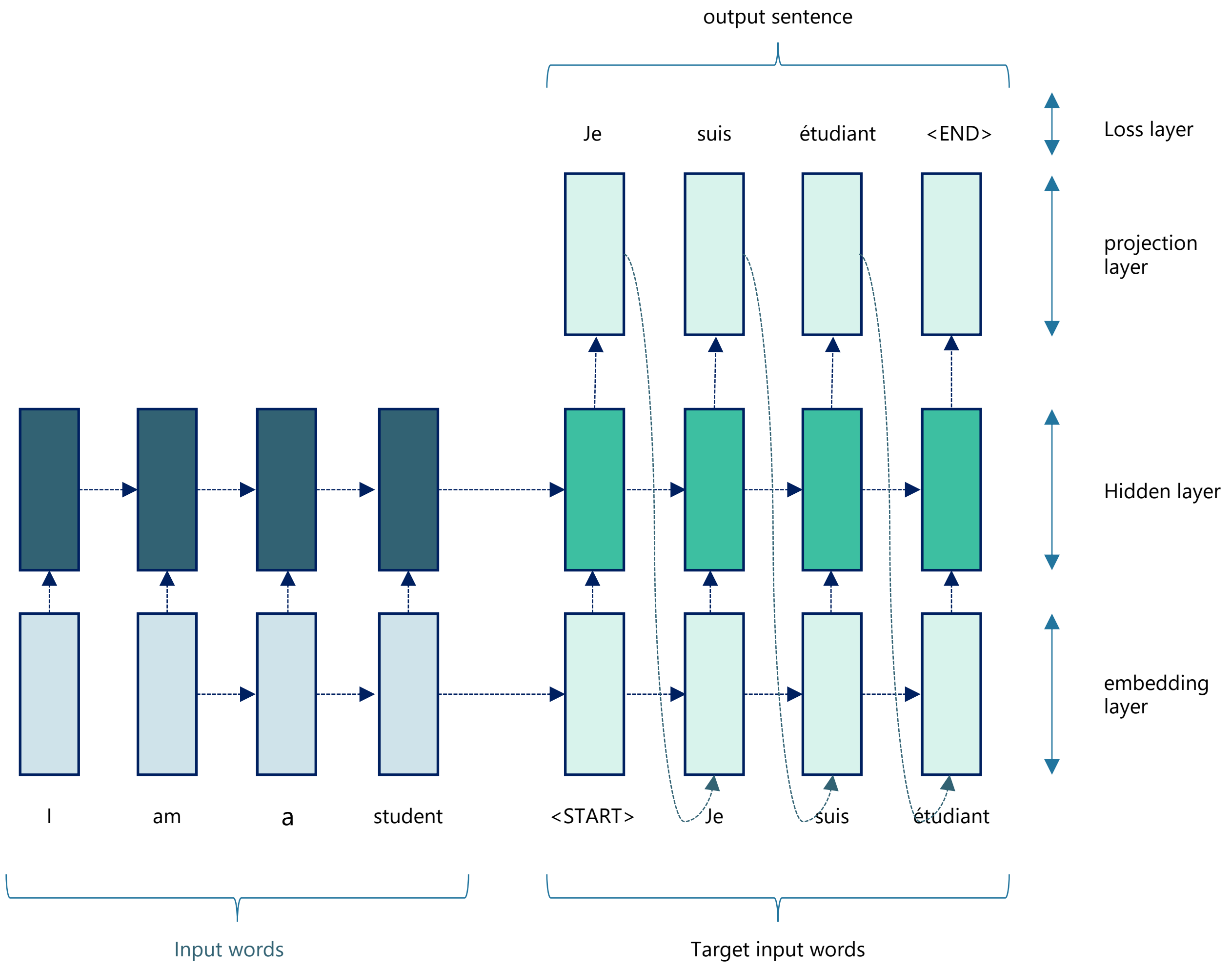
- Unlike previous approach, the probability is conditioned on a distinct context vector c_i for each target word y_i
- c_i depends on the RNN encoder hidden states sequence $(h_1, \dots, h_T)$ and is weighted according to each encoded input vector :

$$\alpha_t = \mathrm{softmax}(e_t)$$
$$c_i = \Sigma_{i=1}^{N} \alpha_{ti} h_i$$

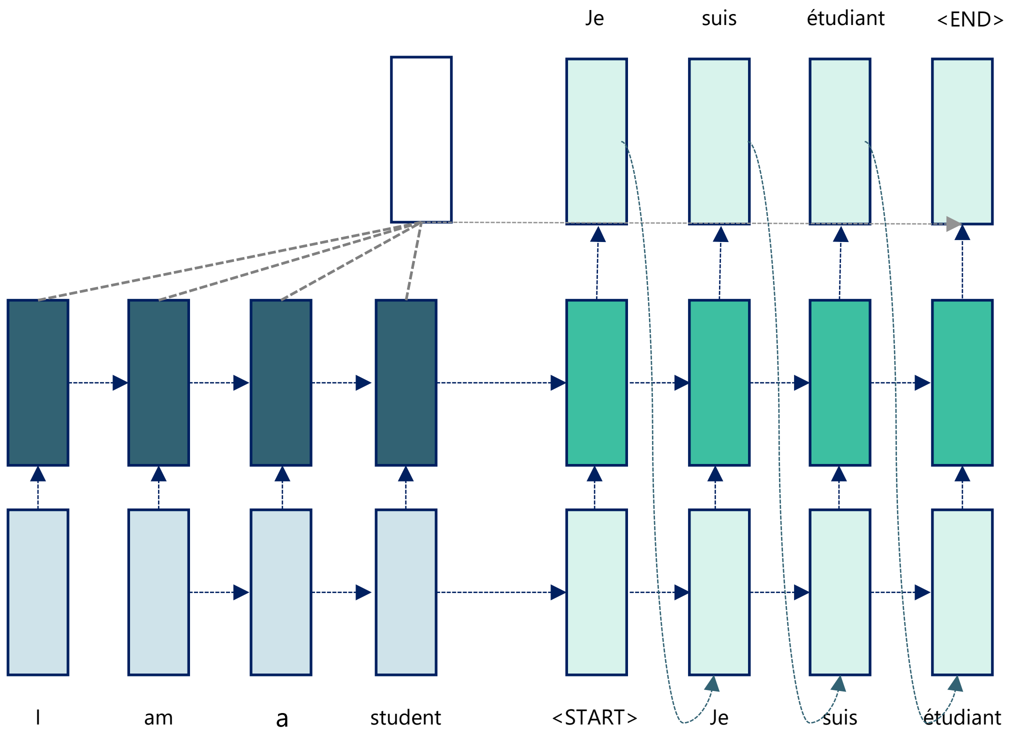
- The context vector is then concatenated to the decoder hidden state to produce the prediction at time step t

Sequence to Sequence

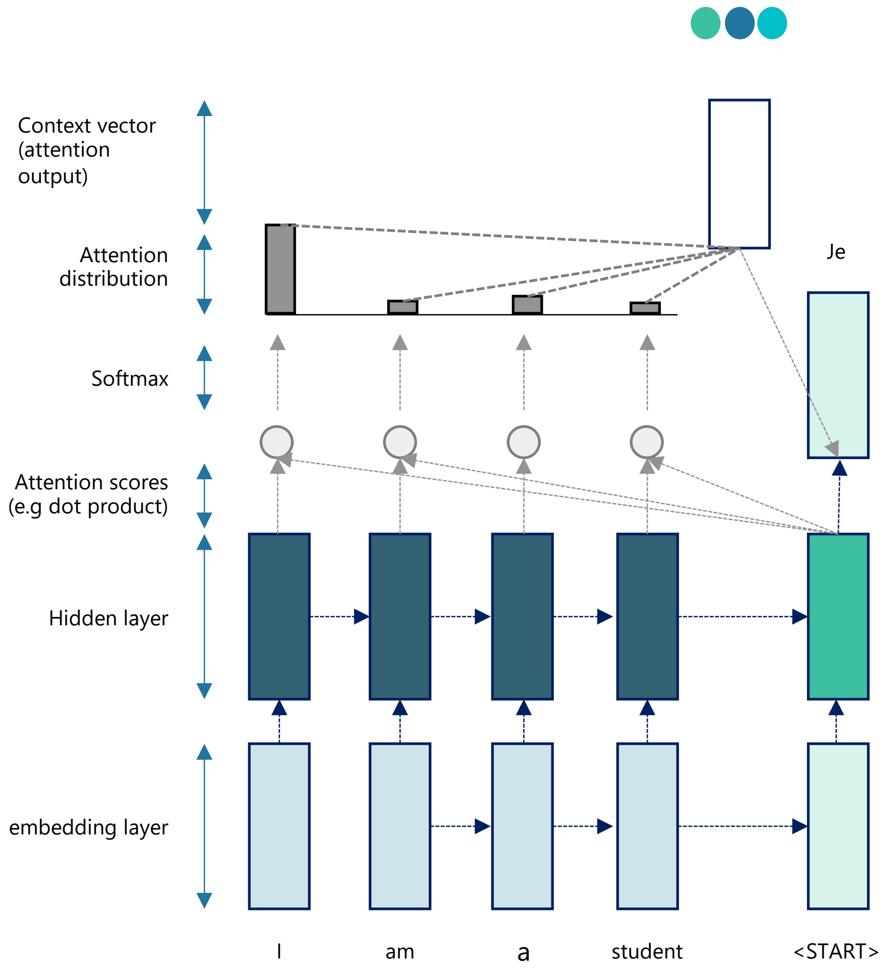
Sequence to Sequence (seq2seq)



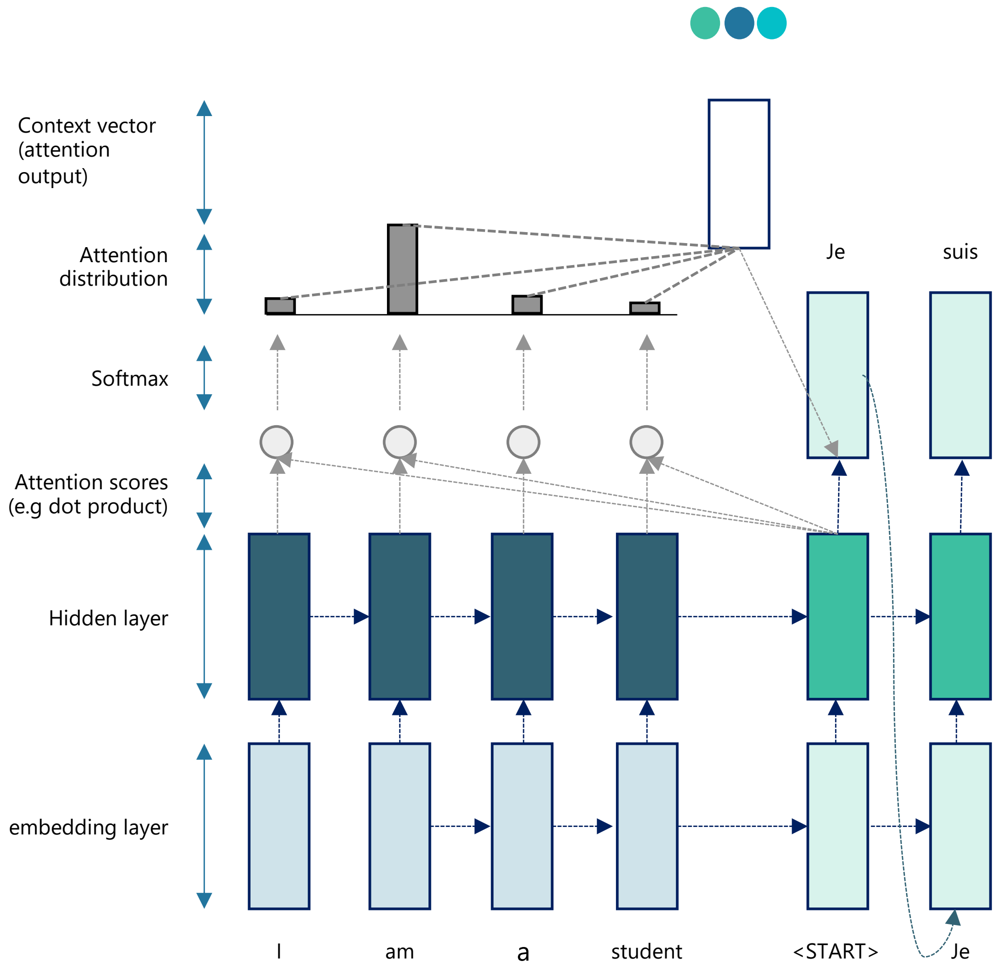
seq2seq with attention



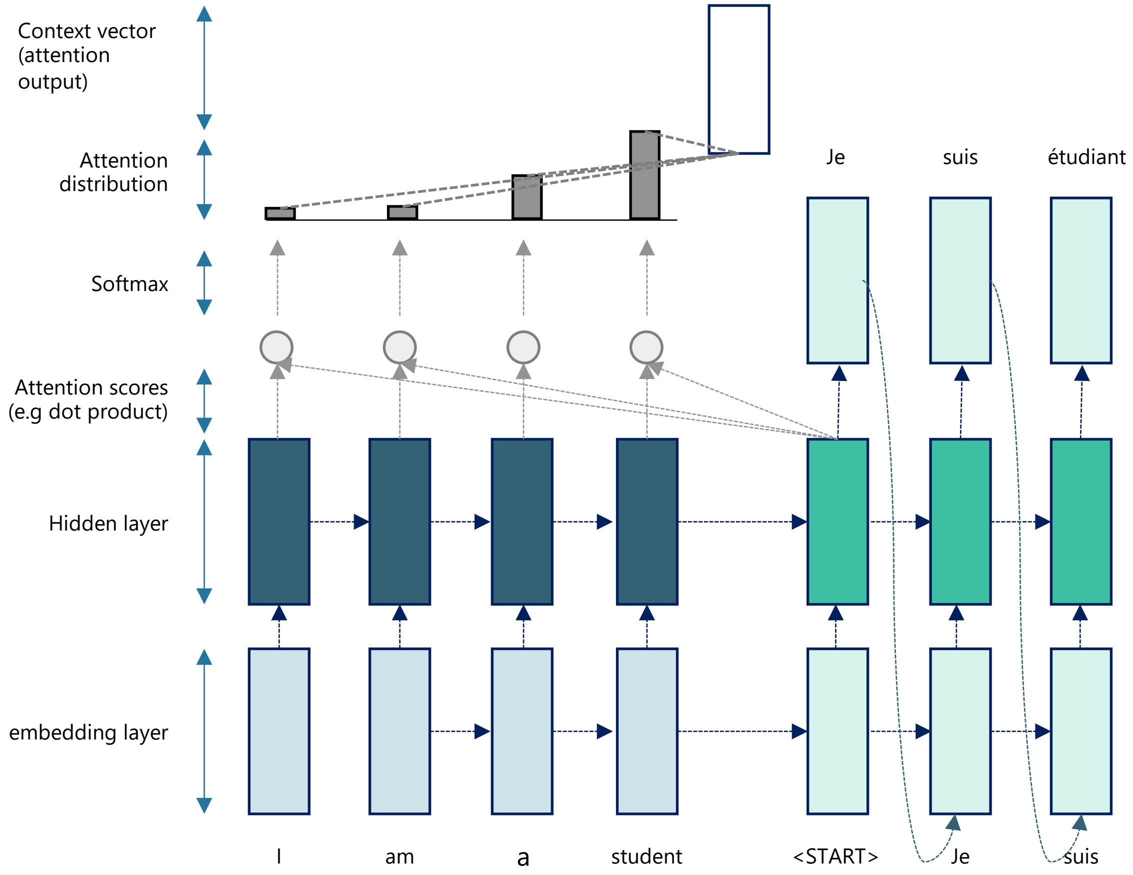
Seq2seq with Attention (1/4)



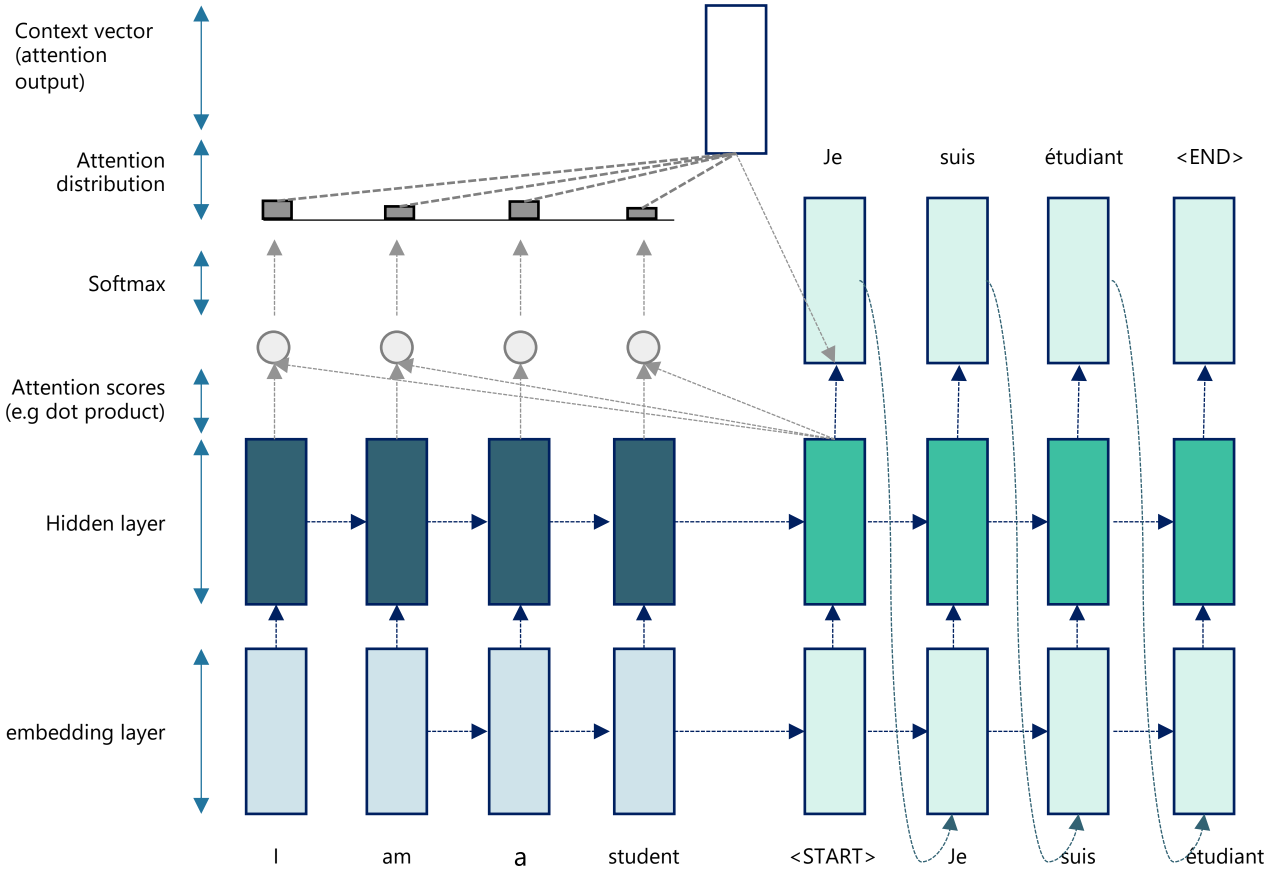
Seq2seq with Attention (2/4)



Seq2seq with Attention (3/4)

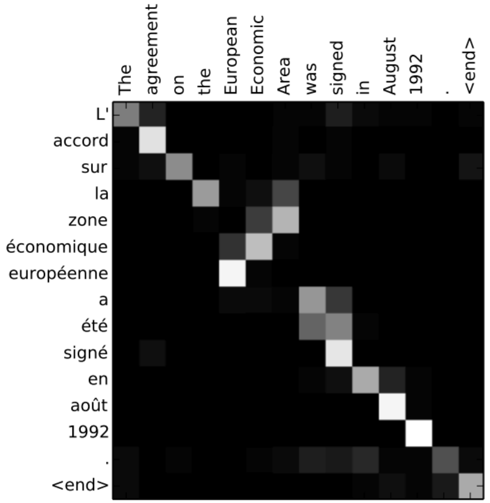


Seq2seq with Attention (4/4)



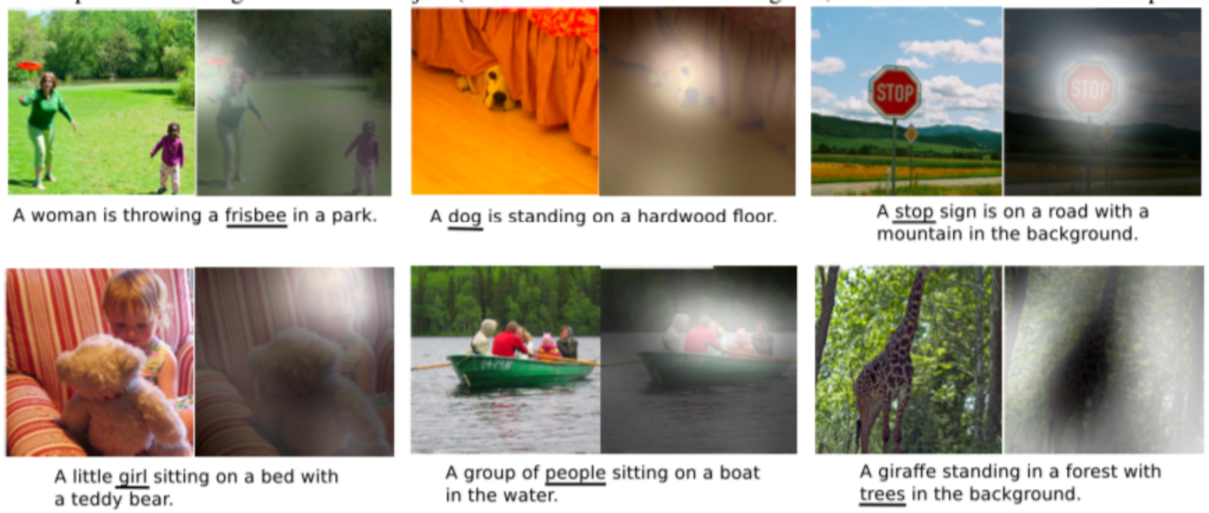
Attention Mechanism is everywhere

- Significantly **improves performances**
- solves the bottleneck problem
  - allows decoder to **look directly at source**; bypass bottleneck
- helps with vanishing gradient problem
- **Provides some interpretability**
- Attention is a general Deep Learning technique that can be leverage for a lot of tasks
  - Machine Translation
  - Sentence classification
  - Sequence tagging
  - Question Answering
  - Image captioning
  - …



Source: [Neural Machine Translation By Jointly Learning To Align And Translate]()

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



Source: [Show, Attend and Tell: Neural Image Caption Generation with Visual Attention]()

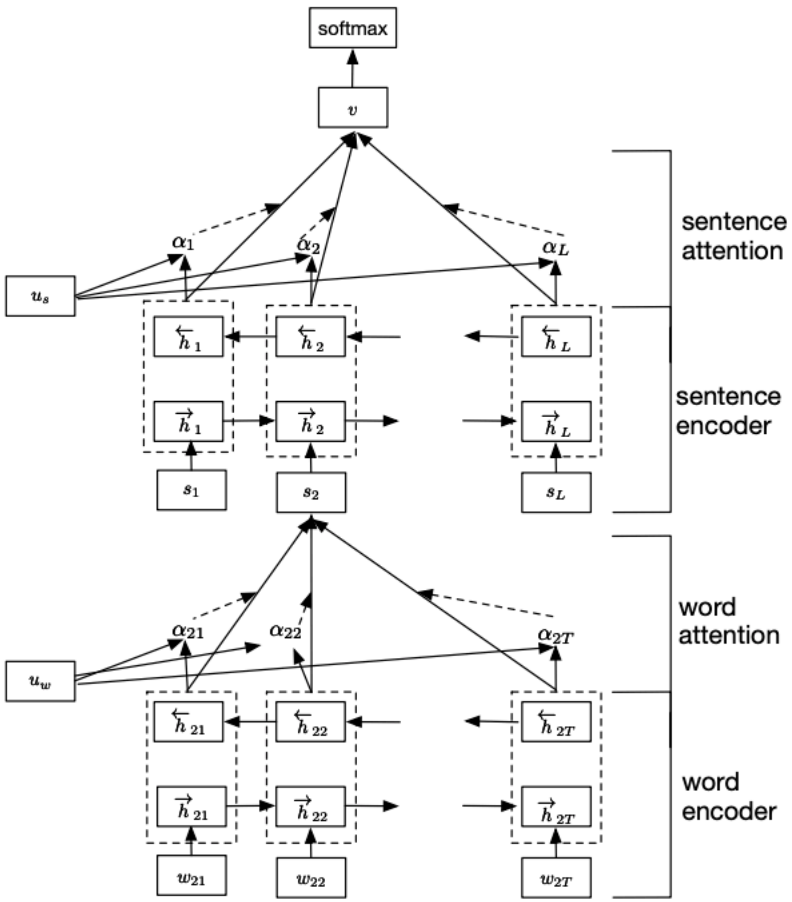
Hands-on 3

30’

Document classification with Hierarchical Attention Network (in Tensorflow)

Hierarchical Attention Network (HAN) for Document Classification

$$Loss = -\sum \log p_{dj}$$



Hierarchical structure that mirrors the hierarchical structure of documents; (ii) it has two levels of attention mechanisms applied at the wordand sentence-level, enabling it to attend differentially to more and less important content when constructing the document representation.

pork belly = delicious . || scallops? || I don't even like scallops, and these were a-m-a-z-i-n-g . || fun and tasty cocktails. || next time I in Phoenix, I will go back here. || Highly recommend.

$$p = \text{softmax}(W_c v + b_c)$$

The document vector $\boldsymbol{v}$ is a high level representation of the document and can be used as features for document classification
We use the negative log likelihood of the correct labels as training loss where $\boldsymbol{j}$ is the label of document $\boldsymbol{d}$.

$$u_i = \tanh(W_s h_i + b_s)$$
$$\alpha_i = \frac{e^{u_i^T u_s}}{\sum_i e^{u_i^T u_s}}$$
$$v = \sum_i \alpha_i h_i$$

$\boldsymbol{\alpha_i}$ the sentence attention vector
$\boldsymbol{v}$ is the document vector that summarizes all the information of sentences in a document.

$$\overrightarrow{h_i} = \overrightarrow{GRU}(s_i), i \in [L, 1]$$
$$\overleftarrow{h_i} = \overleftarrow{GRU}(s_t), t \in [L, 1]$$
$$h_i = [\overrightarrow{h_i}; \overleftarrow{h_i}]$$

h_i summarizes the neighbor sentences around sentence $\boldsymbol{s_i}$ but still focus on i-th sentence.

$$u_{it} = \tanh(W_w h_{it} + b_w)$$
$$\alpha_{it} = \frac{e^{u_{it}^T u_w}}{\sum_t e^{u_{it}^T u_w}}$$
$$s_i = \sum_i \alpha_{it} h_{it}$$

introducing attention mechanism to extract words that are important to the meaning of the sentence and aggregate the representation of those informative words to form a sentence vector $\boldsymbol{s_i}$

$$x_{it} = W_e w_{it}, t \in [1, T],$$
$$\overrightarrow{h_{it}} = \overrightarrow{GRU}(x_{it}), t \in [1, T],$$
$$\overleftarrow{h_{it}} = \overleftarrow{GRU}(x_{it}), t \in [1, T].$$
$$h_{it} = [\overrightarrow{h_{it}}; \overleftarrow{h_{it}}]$$

bidirectional GRU to get annotations of words by summarizing information from both directions for words at time step t of the sentence i, and therefore incorporate the contextual information in the annotation.

 |

Summary of the session

Learning word embeddings

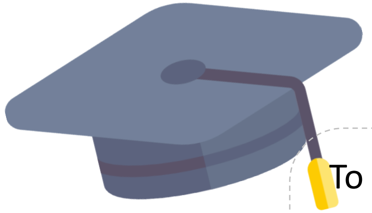
- Negative Sampling and hierarchical softmax are two methods to tackle the computation complexity of Softmax function
- Word embeddings can be pre-trained and use as initialization weights for any NLP tasks
- FastText can create infer embeddings of words that are not in the training set, thanks to character level representations
- Both Word2vec and FastText fail at effectively capturing polysemy

Learning contextual representations of text

- RNN is a family of flexible architectures to model variable length sequences, hence are efficient for NLP problems
- LSTMs (and GRUs) are powerful at capturing long term dependencies compares to simple RNN
- Using bidirectional RNN is preferable (when applicable) to capture the context at each time step
- Seq2seq architecture is particulary suited for Machine Translation or any NLG tasks
- Attention mechanism is a general deep learning technique that helps the model to focus on particular parts of the input
- Attention weights can be used to provide some interpretability

Work for next sessions

to be shared by Sunday March 7th, 2021



To practice what we learned today, for next session, you'll have to :

Questions : Implement some of these suggestions to improve HAN performances for rating classification:

- Apply regularization for generalization : dropout, recurrent dropout, L2/L1 regularization
- Address unbalacanced data problems for training : oversampling, subsampling, loss weighting, new loss function ?
- For evaluation : consider other metrics than accuracy : precision, recall, f1-score, confusion matrix…
- Iterate with more data, including more validation data
- Optional:
  - Interpret your model prediction by extracting and visualizing attention weights at word and sentence level
  - Less aggressive padding strategy by introducing masking ?

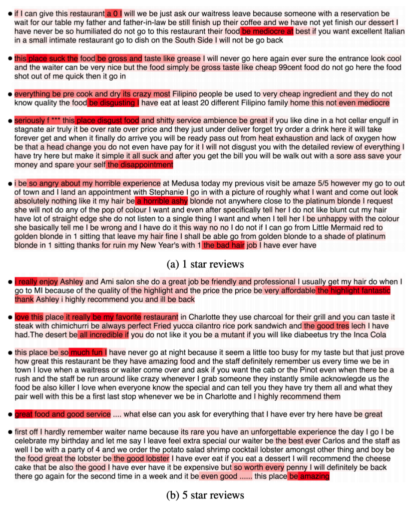
We expect you to send your notebook file by **Sunday 7th evening** to *khemon.beh@capgemini.com* and *johan.attia@capgemini.com*

If you have any questions, feel free to contact us through the slack channel or by email.

References

Word Embeddings

- Efficient Estimation of Word Representations in Vector Space (2013)
- Distributed Representations of Words and Phrases and their Compositionality (2013)
- Bag of Tricks for Efficient Text Classification (2016)
- Enriching Word Vectors with Subword Information (2017)
- Advances in Pre-Training Distributed Word Representations (2017)
- Misspelling Oblivious Word Embeddings (2019)



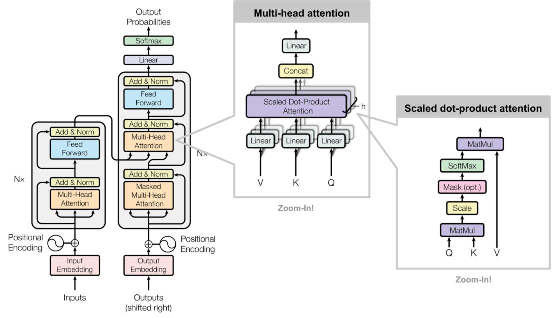
(a) 1 star reviews

(b) 5 star reviews

Figure 2: Heatmap of Yelp reviews with the two extreme score.

Embeddings, Sequence-to-Sequence, Neural Machine Translation & Language Models

- Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation (2014)
- Sequence to Sequence Learning with Neural Networks (2014)
- Neural Machine Translation by Jointly Learning to Align and Translate (2014)
- Effective Approaches to Attention-based Neural Machine Translation (2015)
- A Structured Self-attentive Sentence Embedding (2017)
- Transformer: Attention is All You Need (2017)
- Deep Contextualized Word Representations (2018) (**ELMo**)
- **BERT** : Pre-training of Deep Bidirectional Transformers for Language Understanding (2018)
- GPT-2: Language Models Unsupervised Multitask learners (2019)
- **GPT-3** : Language Models are Few-Shot Learners (2020)



Course evaluation

Did you like that course ? It’s time to share your feedbacks !

Capgemini

Thank you for your attention

GOODBYE !

March 6th, 2020