# DATA SCIENCE CONSULTING
## Session 4

February 20th 2023

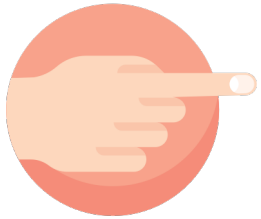
# Agenda

1. **Feedback on intermediary restitution**
2. Final presentation
3. Word Embeddings & Reccurent Neural Network
   A. Word2Vec
   B. FastText
   C. RNN, Encoder Decoder, Sequence to Sequence
4. Language Model and Transformers
   A. Attention mechanism and variants
   B. Transformers

# Feedback on intermediary restitution

## General comments

- Very satisfying level overall
- Good preparation work (good quality slides, research)
- Clear presentations' structure
- Quality of delivery (good presentation skills)



## Areas of improvement

- Some presentations lacked preliminary data analysis
- Explicit the use of data research linked to the business issue and the client customer journey
- Make sure you give enough details regarding your analysis and results
- Do not forget to mention the limits of your analysis
- Do not forget best presentation practices (progress status, source, page number…)

# Agenda

1. Feedback on intermediary restitution
2. **Final presentation**
3. Word Embeddings & Reccurent Neural Network
    A. Word2Vec
    B. FastText
    C. RNN, Encoder Decoder, Sequence to Sequence
4. Language Model and Transformers
    A. Attention mechanism and variants
    B. Transformers

# Final Restitution : 13/03

## Participants

### Project team

### Client team

- Business director
- Data Science director

### Sponsor

- Head of consumer gas & electricity (n-1 to Stéphane Michel, DG gas, renewables and power, exco member)

## Content

### Objectives

- Context
- Need
- Analysis scope

### Methodology

- Data (what, how)
- NLP methodologies used

### Analysis Results

- Explain your conclusions, and the limits associated with them

### Recommendations

- Based on the results of the analysis (including the KPIs)

### Bonus – Next Steps

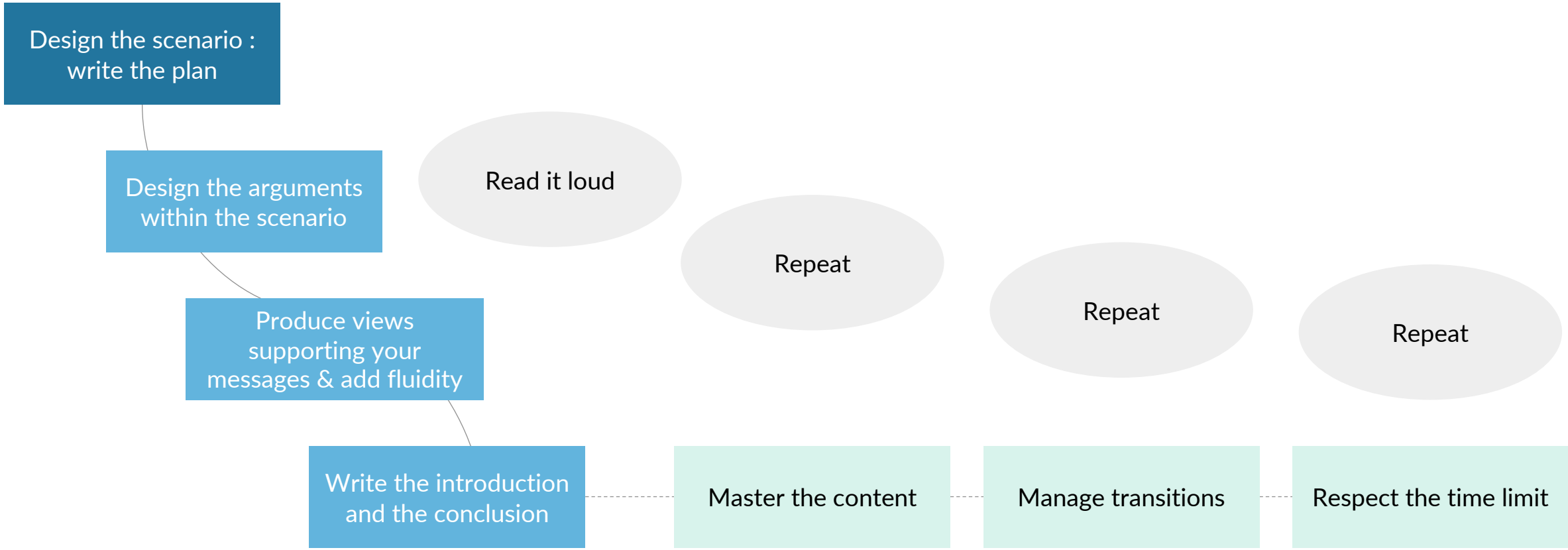
- High level roadmap
- Estimate the operational cost of implementation

## Format

**20mn presentation + 10mn questions**

# Workshop – How to build your storyboard?

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



# Brainstorm

15’

## Start to storyboard your final presentation !

# Adapting your messages to your audience

**Different types of clients/individuals** with different expectations, behaviors, reactions, and concerns that require adaptation in order to deliver the right messages and successfully involve the different stakeholders

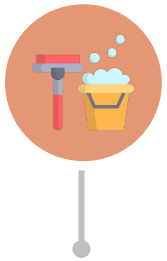
| | PROMOTING | ENABLING | ANALYSING | CONTROLING |
|---|---|---|---|---|
| Characteristics | assertive, expresses feelings, has flair, stimulating, enthusiastic, creative, talks a lot | kind, caring, friendly, cooperative, trustful, sympathetic, careful, sensitive, relaxed | patient, methodical, precise, low talker, thinker/writer, accurate, measured, result-oriented, thorough | determined, goal-oriented, impatient, demanding, efficient, direct |
| What he/she likes | To dream, to see the "big picture"<br>To be congratulated, see his/her work valued in public | To feel part of a group, to work as a team<br>To show consensus | Be provided with all the details<br>To feel secure | Short, strait to the point presentations<br>ROI-focused recommendations |
| What he/she doesn't like | To be restricted to tasks that are not very rewarding<br>Very detailed presentations | To work on his/her own, with little interaction with people<br>Conflict | To depend too much on the work of others<br>To be given unclear prospective subjects | Feeling wasting his/her time<br>Inefficient/unclear presentations |

# Agenda

1. Feedback on intermediary restitution
2. Final presentation
3. **Word Embeddings & Reccurent Neural Network**
   A. Word2Vec
   B. FastText
   C. RNN, Encoder Decoder, Sequence to Sequence
4. Language Model and Transformers
   A. Attention mechanism and variants
   B. BERT

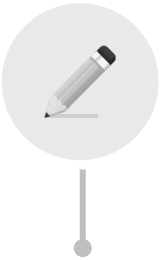
# Data pipeline

Data Collection



Data Cleaning


Word Embedding



Topic Extraction

Sentiment Analysis



# Word embedding reminder

**Definition: Word embedding is a type of word representation that allows words with similar meaning to have a similar representation.[1]**

Machine Learning and Deep Learning algorithms can only take numeric input. This means that for NLP we'll need to transform our text data  (words, tweets, full documents…) into numerical (vectors) by using embedding techniques.

**Some embedding techniques:**

- Traditional approaches:
  - Latent Semantic Indexing (LSI)
  - Word2Vec
  - FastText
- Advanced techniques: Transformers
  - GPT-3
  - BERT
  - CTRL


[1] GOLDBERG, Yoav. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 2017, vol. 10, no 1, p. 1-309.

# Agenda

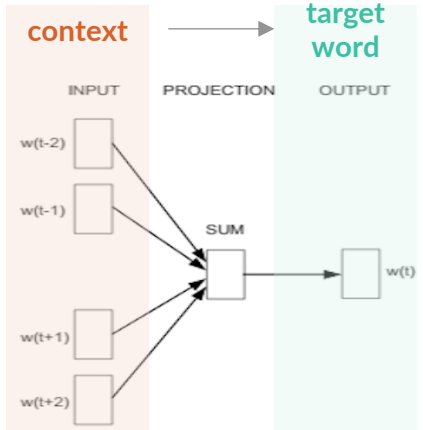
1. Feedback on intermediary restitution
2. Final presentation
3. **Word Embeddings & Reccurent Neural Network**
   A. **Word2Vec**
   B. FastText
   C. RNN, Encoder Decoder, Sequence to Sequence
4. Language Model and Transformers
   A. Attention mechanism and variants
   B. Transformers

# Word2Vec

- Word2Vec is an embedding method that which leverages the context of target words.
- It uses the surrounding words to represent target words with a Neural Network whose hidden layer encodes the word representation
- It provides a representation of each of the V words in the vocabulary by mapping context and targets in a sentence.
- Two different methods are available:

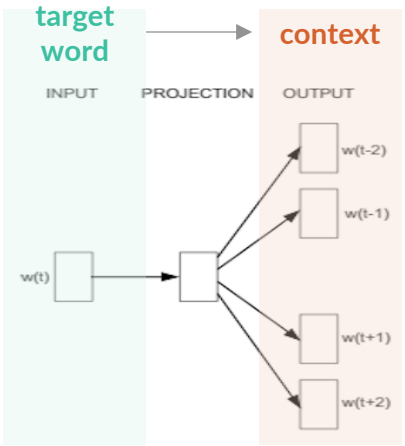
**CBOW**

Takes the **context** of each word as the input and tries to predict the word corresponding to the context being the **target**




**Skipgram**

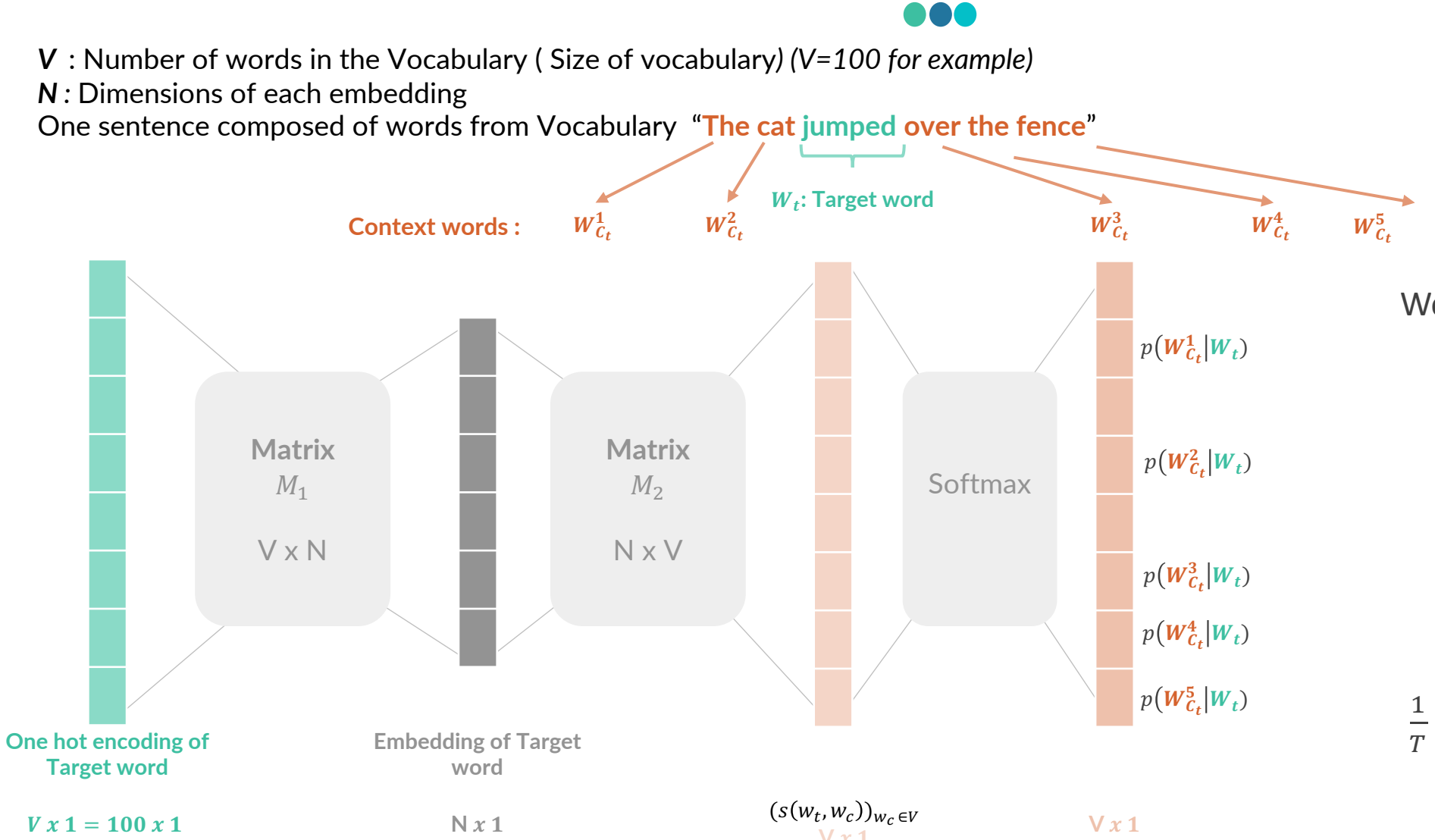
Takes the **target** word and tries to predict the **context** words of that target word and produce representations




*Corpus example*: "**The cat jumped over the fence**"
{"The", "cat", "over", "the", "fence"} => **context words**
{"jumped" } => **target word**

# Skipgram Architecture

***V*** : Number of words in the Vocabulary ( Size of vocabulary) *(V=100 for example)*
***N*** : Dimensions of each embedding
One sentence composed of words from Vocabulary "The cat jumped over the fence"




We want to maximize the log likelihood

$$\sum_{c \in C_t} \log p(w_c \mid w_t)$$

On T words

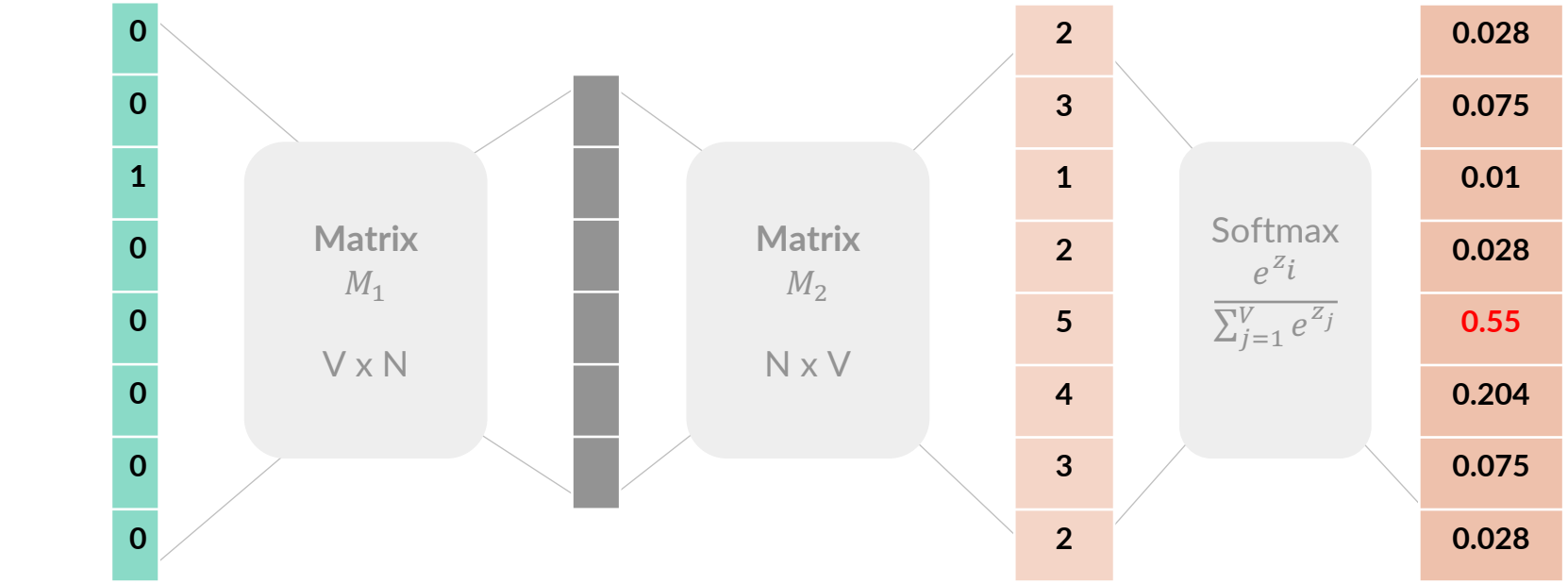
$$\frac{1}{T} \sum_{t=1}^{T} \sum_{c \in C_t} \log p(w_c \mid w_t)$$

# Zoom on softmax function

- Each word $w$, have 2 representations : $u_w$ (when w is a target word) and $v_w$ (when w is a context word) such that :

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

- s is a scoring function between a word $w_t$ and a context word $w_c$




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

# Negative Sampling Skipgram

## Problems with previous softmax formulation :

1. Implies to predict only one context word $w_c$ given a word $w_t$. ❌
2. Computing is very expensive ❌

   For example : if V = 10 000 words and N = 300, the matrix will have size = 3M.

   For each training sample, we will need to update 3M weights.

   If we have 1 billion training samples, we will need to update 3M weights 1 billion times.

## Alternatives :

1. Predict context words as a set of independant binary classification (presence or absence of a word in context) ✅
2. Limit the numbers of weights that will be updated by each training sample ✅
3. Performing subsampling ✅

For a word $w_t$ :

- New training dataset formulation : $\big((w_t, w_2)\big) with\ a\ label\ y\ , y = 1\ if\ (w_2)\ is\ in\ the\ context\ of\ w_t$ and 0 if not
- 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 :

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

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

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



*Sources : http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/*

# Negative Sampling Skipgram

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

$C_t$ : set of context words of word $w_t$
$\mathcal{N}_{t,c}$ : set of negative samples of word $w_t$

context word

center/target word

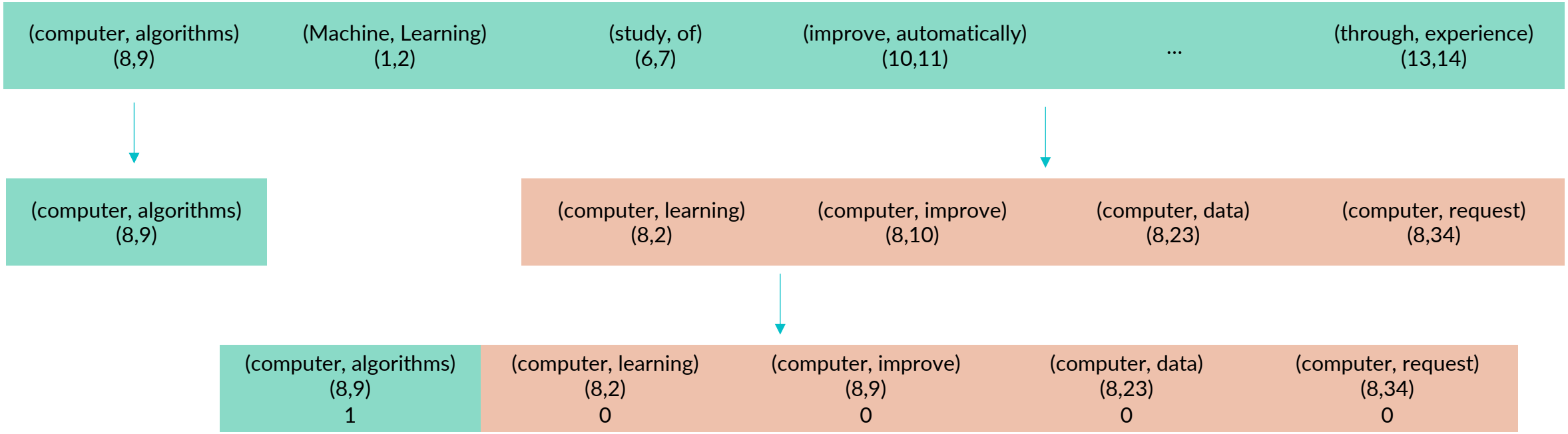
sliding window

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

It is seen as a part of artificiel 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 samples

Machine learning is the study of **computer** algorithms that improve automatically through experience




| Word | Context Words | | | | | Labels | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 9 | 2 | 10 | 23 | 34 | 1 | 0 | 0 | 0 | 0 |
| 8 | 2 | 11 | 28 | 46 | 22 | 1 | 0 | 0 | 0 | 0 |
| ⋮ | | | | | | | | | | |
| 8 | 39 | 27 | 4 | 29 | 10 | 1 | 0 | 0 | 0 | 0 |

# Agenda

1. Feedback on intermediary restitution
2. Final presentation
3. **Word Embeddings & Reccurent Neural Network**
   A. Word2Vec
   **B. FastText**
   C. RNN, Encoder Decoder, Sequence to Sequence
4. Language Model and Transformers
   A. Attention mechanism and variants
   B. Transformers

# FastText

**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 :

$$< wh,\ whe,\ her,\ ere,\ re >$$

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 :

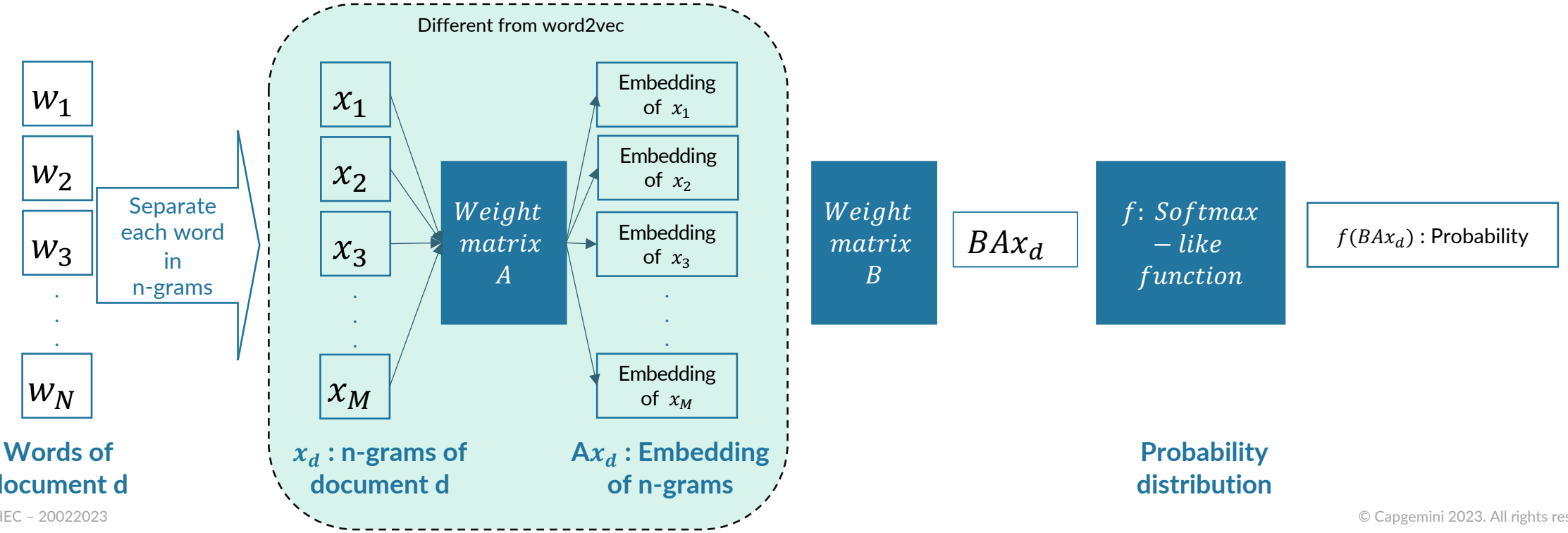
$$u_w = \Sigma_{g \in \mathcal{G}_w} z_g$$

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

where $c$ is the context word.

# FastText

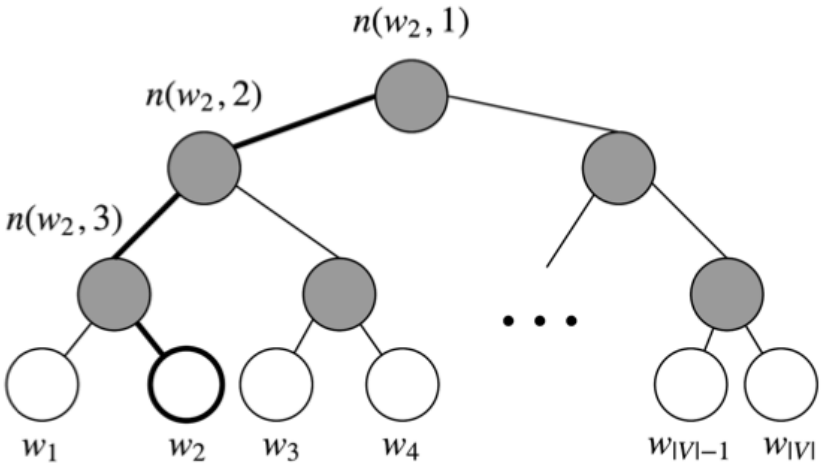
- Words $w_1, \ldots, w_N$ and label $y$ (classification)
- **bag of n-gram features** to capture partial words order information
- **Weight matrix $A$** embedding each n-gram features $x_1 \ldots x_M$
- **Words representations averaging** to get the hidden text representation
- Linear classifier : **weight matrix $B$** for linear projection then **softmax-like function** to get probability distribution



# Softmax Approximation : Hierarchical Softmax

- Hierarchical softmax uses a binary tree to represent all words in the vocabulary. Each leaf of the tree is a word, and there is a unique path from root to leaf.
- In this model, the probability of a word *w* given a vector *wi*, *P*(*w*|*wi*), is equal to the probability of a random walk starting in the root and ending in the leaf node corresponding to *w*.
- The main advantage in computing the probability this way is that the cost is only $O(\log_2(V))$, corresponding to the length of the path.
- 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 the left child of $n$ and $[x] = 1$ if $x$ is true and $-1$ otherwise. Thus, hierarchical softmax defines $p(w|w_I)$ as follows :

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




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

Source : https://flavien-vidal.medium.com/

# To retain from Word2Vec and Fastext

Two embedding approaches  that have one thing in common, they have exactly one embedding for one word or n-gram of a word. They do not look at the contextual relationship of a specific word, or only in a very small range and limited way (skipgram model).

| Word2Vec | FastText |
| --- | --- |
| • Provides a representation for each word in the vocabulary.<br>• Limited when dealing with misspellings and complex languages.<br>• Does not consider morphology of words. | • Representation of every n-gram.<br>• Size of the dictionary can cause a high memory requirement. |

Two main problems that present these methods are:

- Always the same representation for a word type, regardless of the context in which a word token occurs.
- Only one representation for each word, but each one has different aspects, including semantics, syntatic behaviour, and register/connotations.

In order to fix them, we should consider some more advanced techniques that will use the overall context of a word in a given sentence. A same word will have different representation depending on the sentence they are in. Some examples of these methods are:

- ELMO (Embeddings from Language Models).
- Transformers techniques such as GPT-3, BERT or CTRL.

# Hands-on 1

45’

17: 20

## Skipgram with negative sampling using TensorFlow



# Break

20’

**See you after 20 min!**

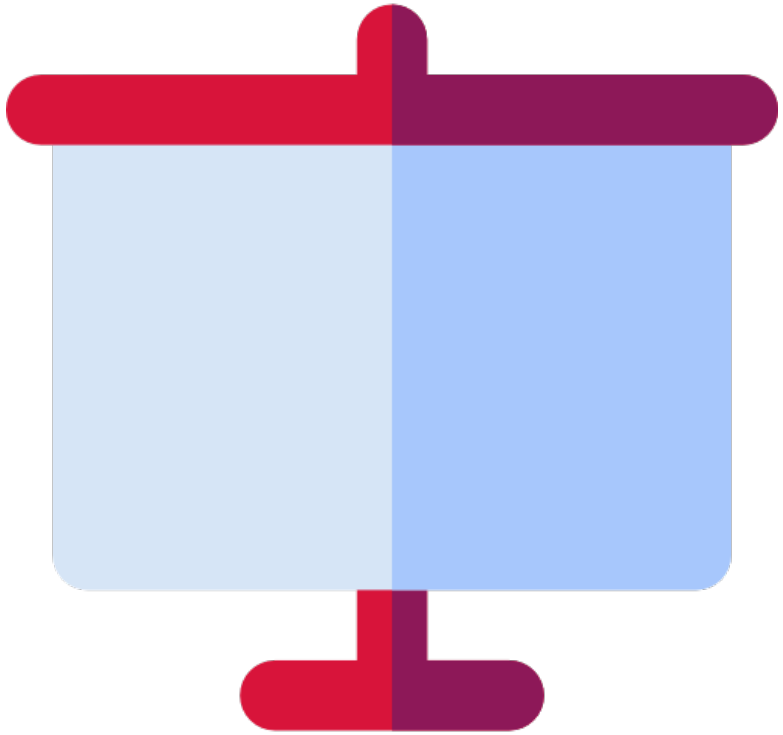


# Restitution

5’

## Could you implement Word2Vec ?





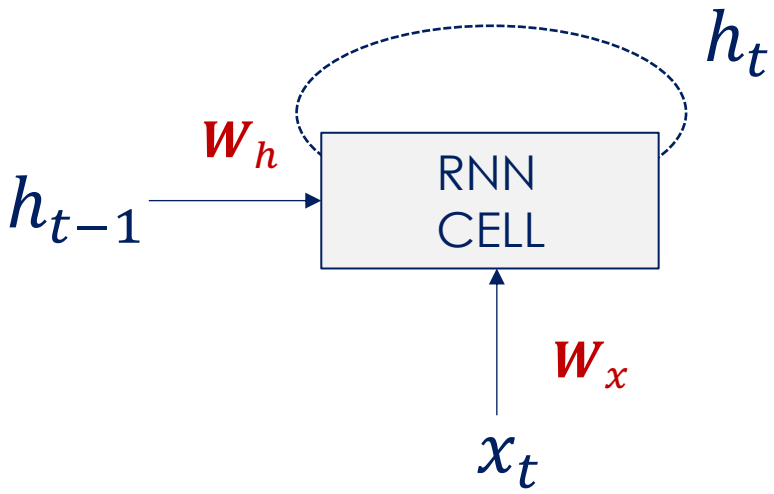
# Agenda

1. Feedback on intermediary restitution
2. Final presentation
3. **Word Embeddings & Reccurent Neural Network**
   A. Word2Vec
   B. FastText
   **C. RNN, Encoder Decoder, Sequence to Sequence**
4. Language Model and Transformers
   A. Attention mechanism and variants
   B. Transformers

# Reccurent Neural Networks (RNN)

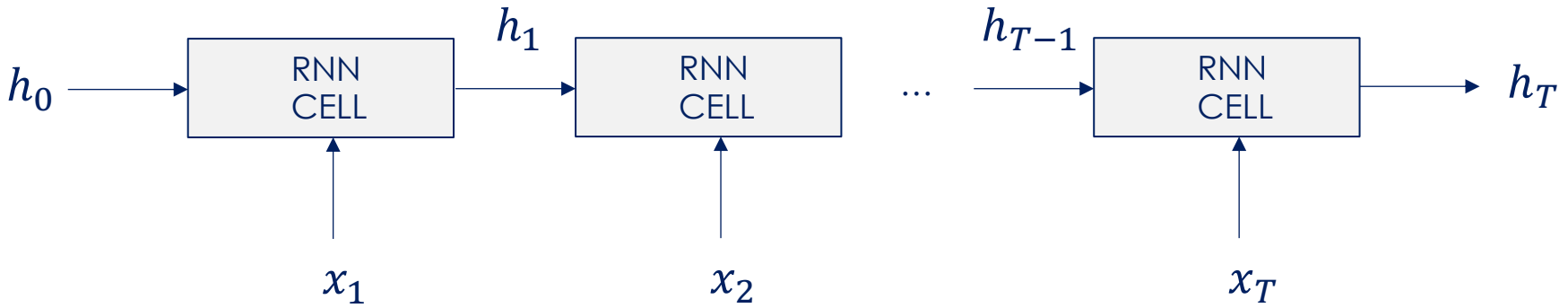
- Consider a variable-length sequence x $= (x_1, \dots, x_T)^\top$ such that $x_t \in \mathbb{R}^N$ and representing the sequence of words $(w_1, \dots, w_T)$
- A simple recurrent neural network encodes such sequences at each step $t$ in the following way :

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




where :

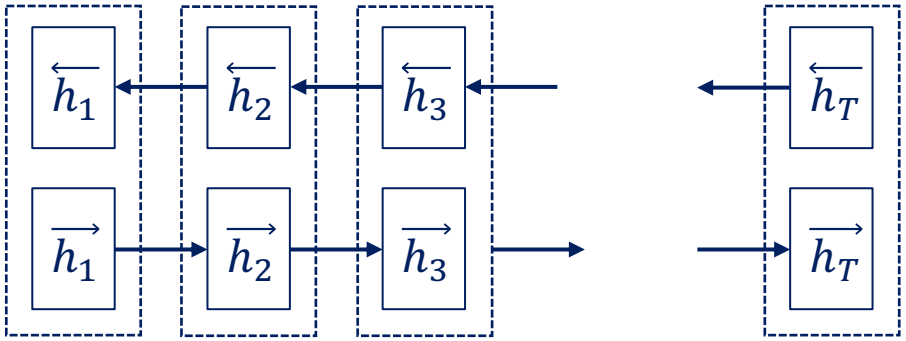
- $x_t$ is the word embedding of the word $w_t$
- $\boldsymbol{W_h} \in \mathbb{R}^{D \times D}, \boldsymbol{W_x} \in \mathbb{R}^{D \times N}, h_t \in \mathbb{R}^D$
- $f$ is called the RNN Cell
- $\mathbf{h} = (h_1, \dots, h_T)^\top$ are called hidden states
- Simple RNNs are struggling to capture long 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 $(\overrightarrow{h_1}, \dots, \overrightarrow{h_T})$
- The backward RNN $\overleftarrow{f}$ reads the reverse input sequence from $x_T$ to $x_1$ and builds the backward hidden states $(\overleftarrow{h_1}, \dots, \overleftarrow{h_T})$
- Finally, get $(h_1, \dots, h_T)$ such that $h_j = [\overrightarrow{h_j}\,;\,\overleftarrow{h_j}]$

# Reccurent Neural Networks (RNN)

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

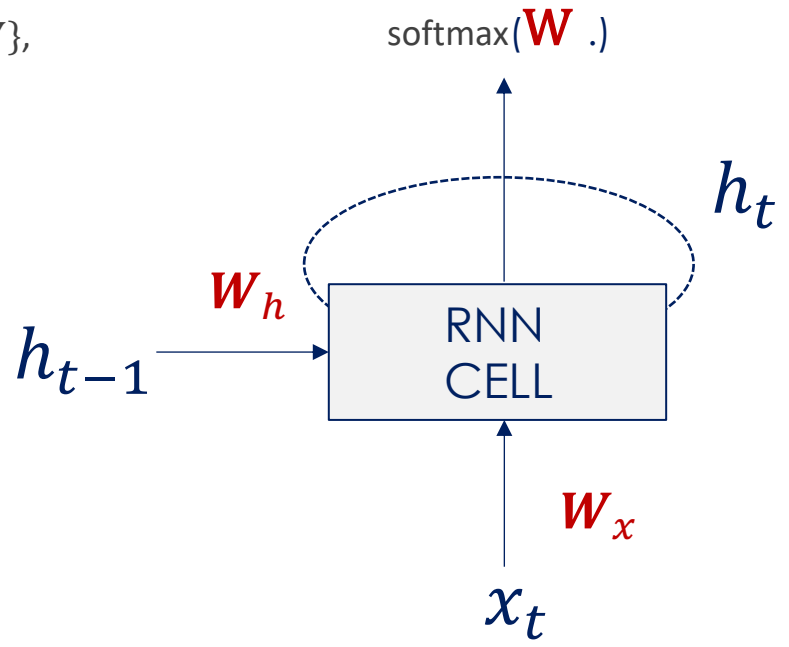
- Train the network to predict the next symbol in the sequence $p(x_t \mid x_{t-1}, \dots, x_1)$. For $j \in \{1, \dots, W\}$,

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

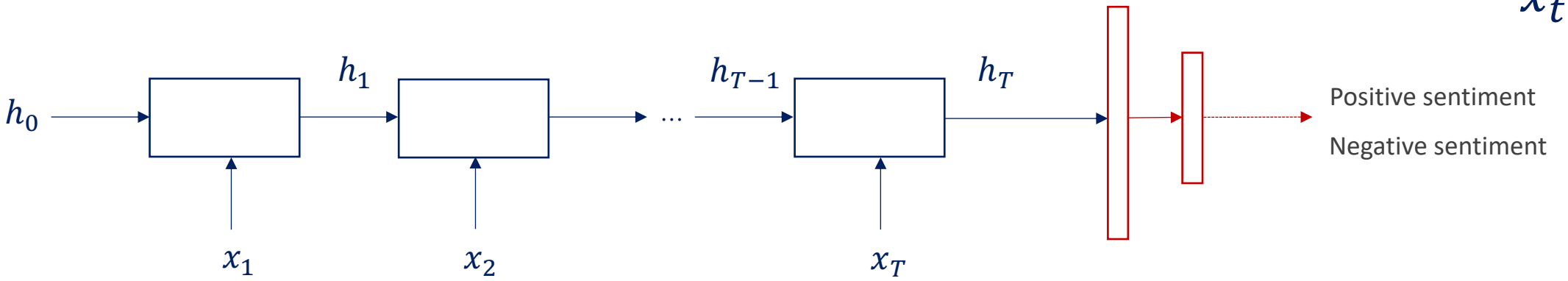
where $\mathrm{w}_j$ are the rows of the weight matrix $\mathbf{W} \in \mathbb{R}^{W \times D}$

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

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



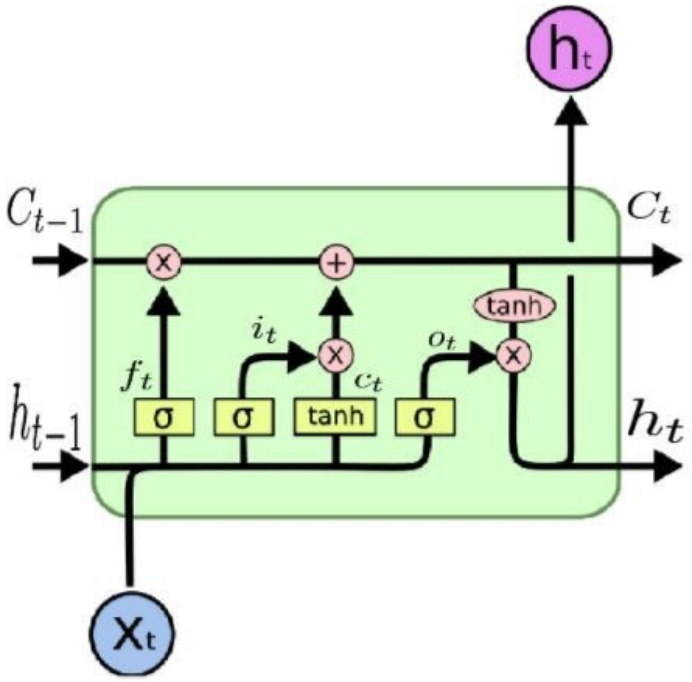

**Use RNN for sequence classification (e.g. sentence classification)**



# Long Short Term Memory (LSTM)

**Type of RNN as a solution to preserve information over many time steps (Problem of Vanishing gradients)**

On step $t$, there 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

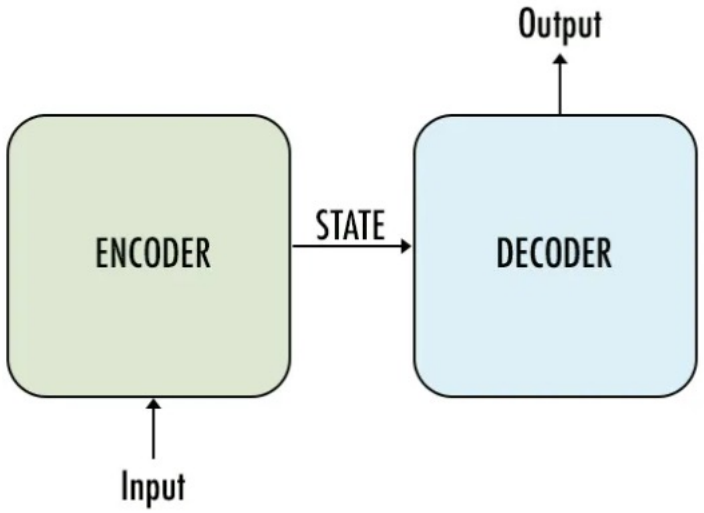
| | |
|---|---|
| $\boldsymbol{f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)}$ | **Forget gate**: controls what should be kept or forgotten from previous cell state |
| $\boldsymbol{i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)}$ | **Input gate**: controls what part of the new cell content are written to cell |
| $\boldsymbol{o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)}$ | **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/

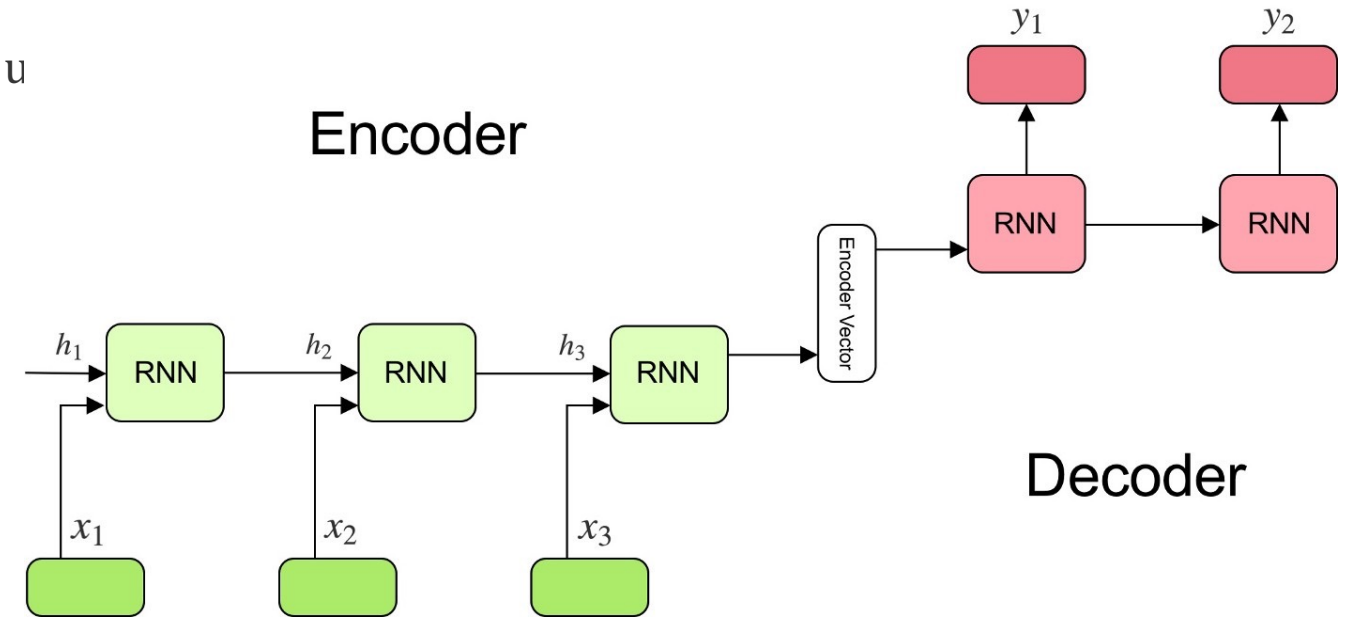
# Encoder-Decoder Architecture

**Output of decoder =>** $y_t = softmax(W^S h_t)$

Using the hidden state at the current time step together with the respective weight W(S)




pez u



# Encoder-Decoder Architecture

I. **Encoder**:
   - Processes the sequence and then returns at the end an encoding vector of the whole series
   - Several LSTM cells, for each cell it accepts a single element of the input sequence and propagates forward
   - Generates a context vector that summarizes the hidden layers and encapsulates all the information

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

II. **Decoder**:

   - The context vector is then taken as the input to the decoder in order to make the predictions.
   - Each recurrent unit accepts a hidden state from the previous unit and produces and output as well as its own hidden state.

$$h_t = f(W^{(hh)} h_{t-1})$$

# Encoder-Decoder Architecture

**Main Drawbacks of the Encoder-Decoder Model:**

- ✓ **Architecture with Limited memory**: The final state is where all the information of the sentence is -> the more the length is fixed of the W matrix, the lossier the neural network will be
- ✓ **Deeper neural network => hard to train** i.e. for RNN, the length of the neural network is dependant on the sequence's length

    => **Problem of** Vanishing Gradients: gradients that are used to update the weights shrink

For more Robust and lengthy sentences > **Attention Models**

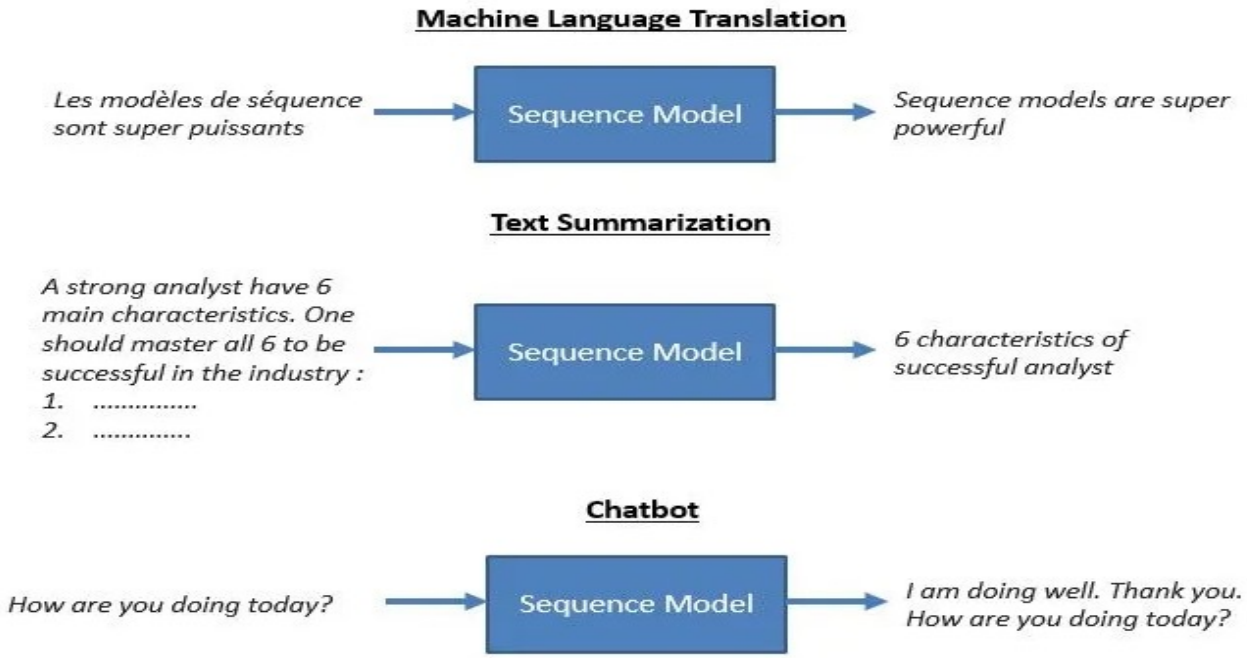
# Sequence to Sequence Models

Use Cases of the **Sequence to Sequence** Models (seq2seq):

- Google Translate
- Chatbots

- The applications are related to any sequence-based problem when the input and outputs have different sizes (Machine Translation and Speech Recognition)

- Most Common architecture to build Seq2Seq models is Encoder-Decoder Architecture



# Agenda

1. Feedback on intermediary restitution
2. Final presentation
3. Word Embeddings & Reccurent Neural Network
    A. Word2Vec
    B. FastText
    C. RNN, Encoder Decoder, Sequence to Sequence
4. **Language Model and Transformers**
    A. **Attention mechanism and variants**
    B. Transformers

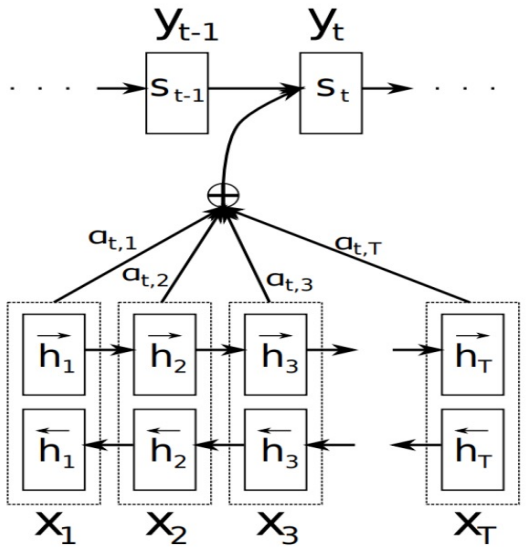
# Attention Model

The fixed size context-vector bottleneck was one of the main motivations by [Bahdanau et al. 2015], which proposed a similar architecture but with a crucial improvement**:**

✓ **Encoder**: Bi-Directionnal RNN with a forward and backward hidden states
=> Concatenation of the two hidden states= encoder state at a given position

***Why?*** Include both preceding and following words in the representation of an input word

✓ **Decoder:** Search mechanism at the decoder level => Look at the whole input sentence



# Attention Mechanism

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

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

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 $t$, $s_t$**

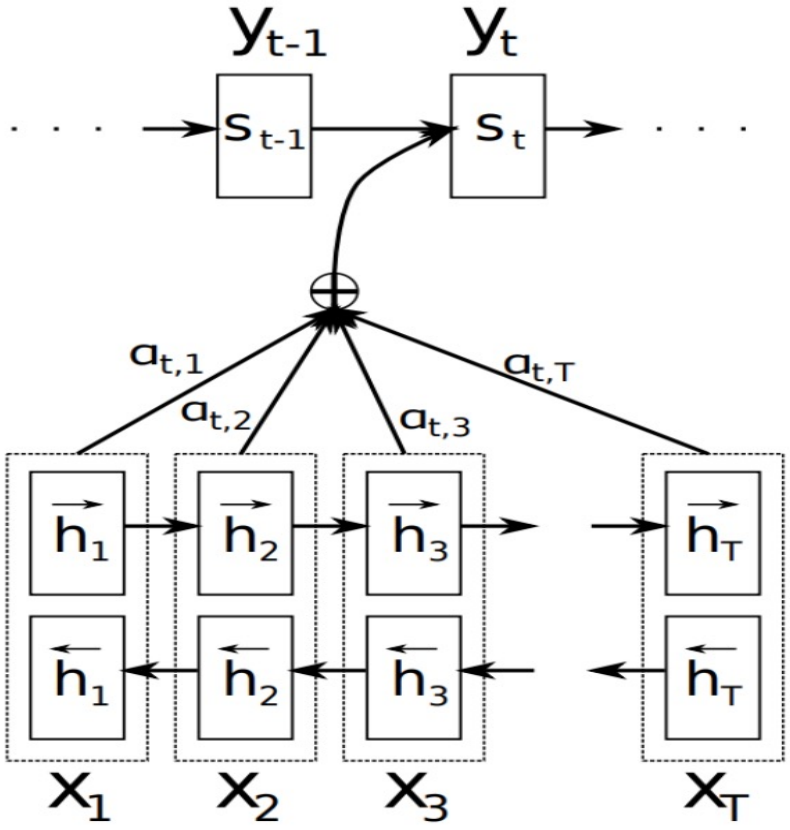
$$e_t = [s_t^T h_1, \ldots, s_t^T h_N]$$

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 $(h_1, \ldots, h_T)$ and is weighted according to each encoded input vector :

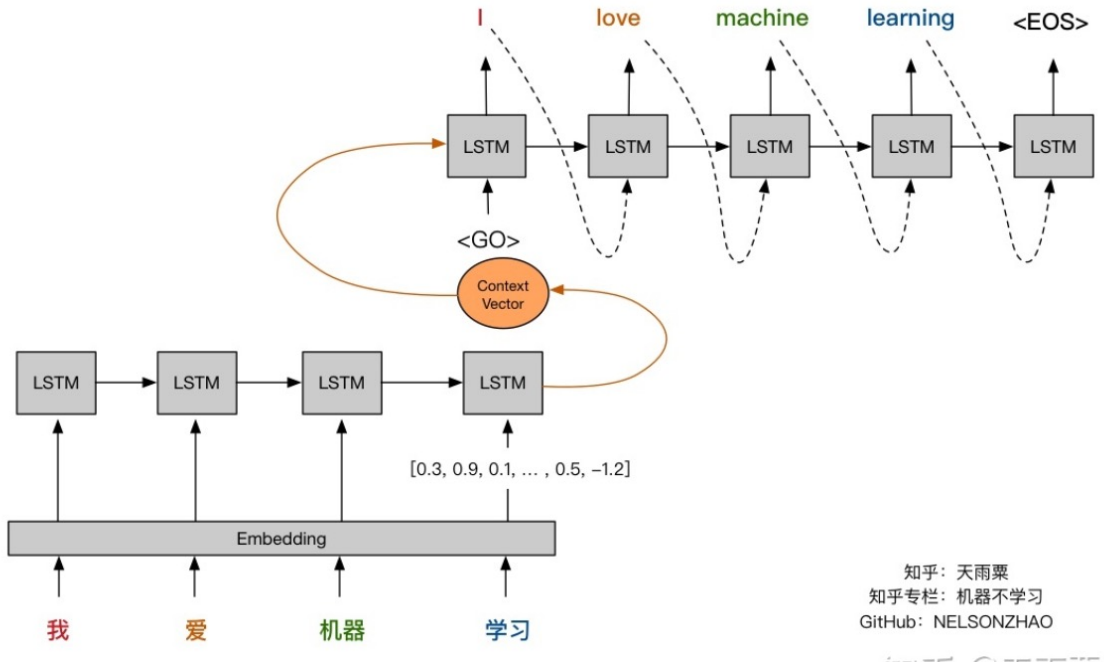
$$\alpha_t = \mathrm{softmax}(e_t)$$
$$c_i = \Sigma_{i=1}^{N} \alpha_{ti} h_i$$

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



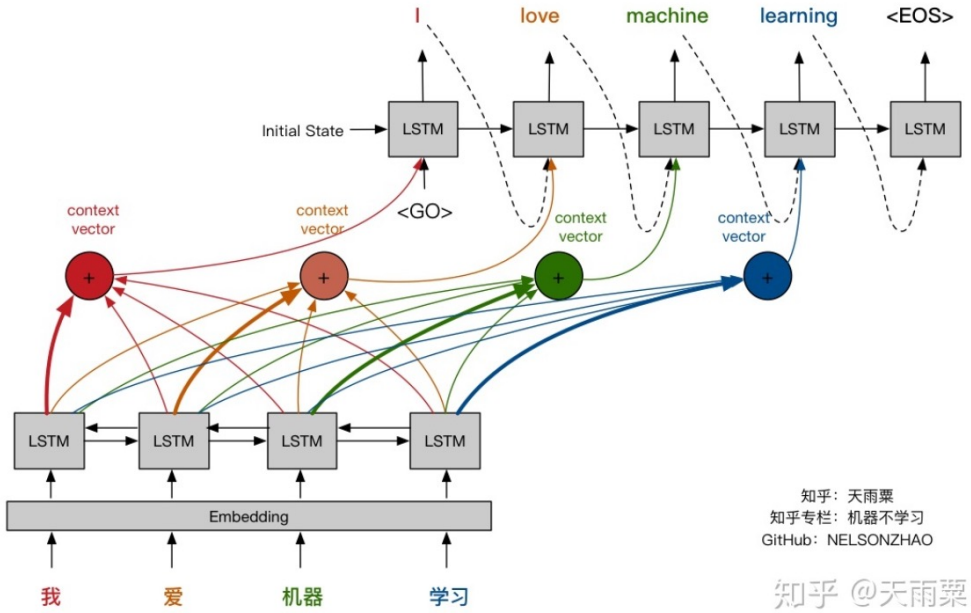
# Attention Mechanism

## Seq2seq with fixed-length




https://zhuanlan.zhihu.com/p/37290775

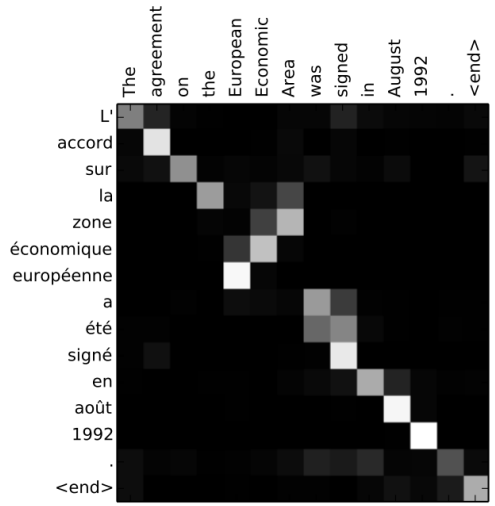
## Seq2seq with attention




https://zhuanlan.zhihu.com/p/37290775

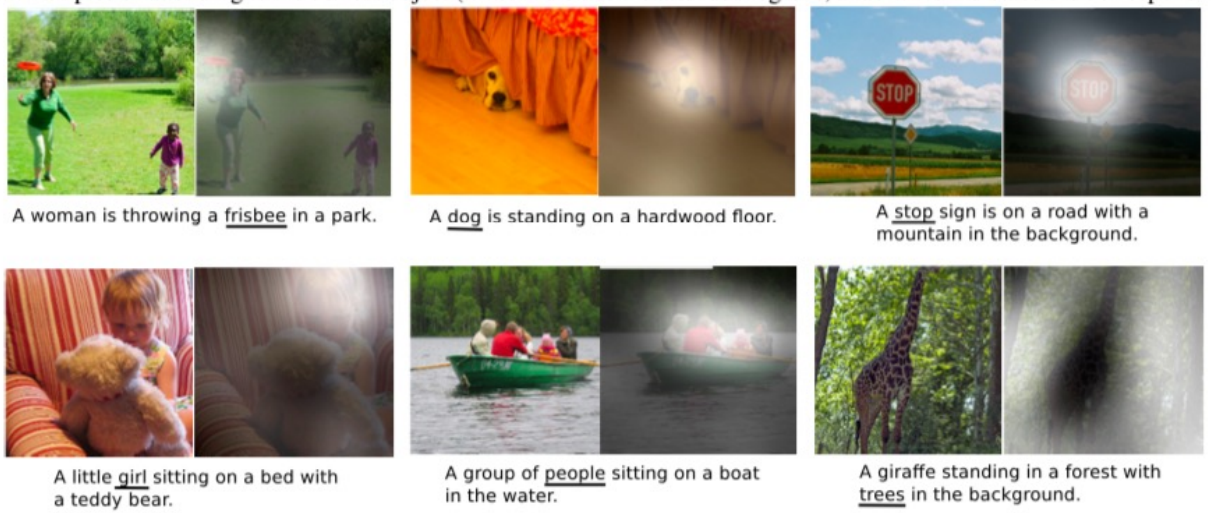
# Attention Mechanism is everywhere

- 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

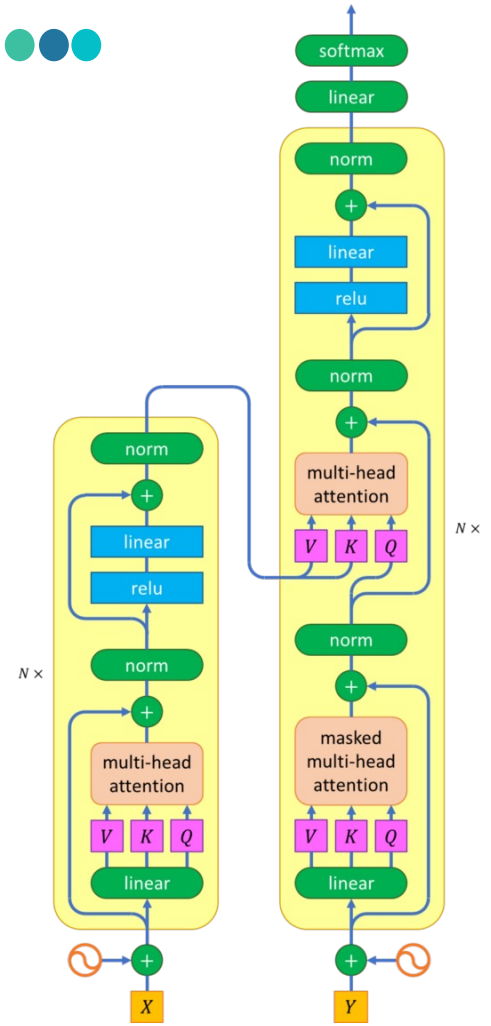
*Figure 3.* Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)




Source: Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

# Transformers

- Transformer: stack of 6 layers
- 6 layer encoder stack on the left
- 6 layer decoder stack on the right
- No RNN, LSTM et CNN




$$P_{i,2j} = \sin\left(i/10000^{2j/d_x}\right)$$

$$P_{i,2j+1} = \cos\left(i/10000^{2j/d_x}\right)$$

$$X = [x_1 \quad x_2 \quad \cdots \quad x_t]^T + [P_{ij}]$$

$$Y = [<\text{bos}> \quad y_1 \quad \cdots \quad y_m]^T + [P_{ij}]$$

For $r = 1,2,\ldots,N$

$$X := \text{LayerNorm}\left(X + \text{MultiHead}(XW_Q, XW_K, XW_V)\right)$$

$$X := \text{LayerNorm}\left(X + \max\left\{0, XW_1^{(r)} + b_1^{(r)}\right\}W_2^{(r)} + b_2^{(r)}\right)$$

For $r = 1,2,\ldots,N$

$$Y := \text{LayerNorm}\left(Y + \text{MaskedMultiHead}(YW_Q, YW_K, YW_V)\right)$$

$$Y := \text{LayerNorm}\left(Y + \text{MultiHead}(YW_Q, XW_K, XW_V)\right)$$

$$Y := \text{LayerNorm}\left(Y + \max\left\{0, YW_3^{(r)} + b_3^{(r)}\right\}W_4^{(r)} + b_4^{(r)}\right)$$

$$\text{proba} = \text{softmax}(YW_O)$$

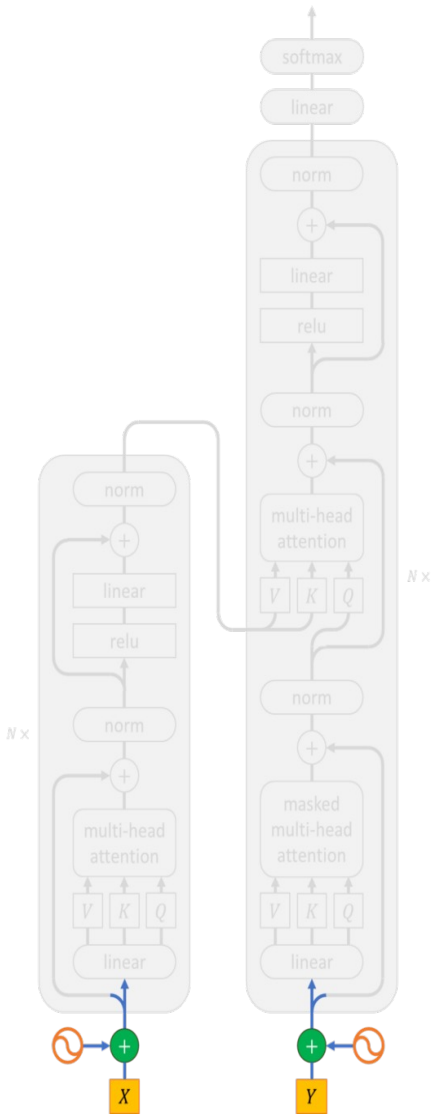
# Transformers

Positional encoding added to word embeddings

- Provide sine and cosine functions for the positional encoding for each dimension I of the word embedding vector

$$P_{i,2j} = \sin\left(i/10000^{2j/d_x}\right)$$
$$P_{i,2j+1} = \cos\left(i/10000^{2j/d_x}\right),$$

- Add positional vector to embedding vector before fed to the first encoder




$P_{i,2j} = \sin\left(i/10000^{2j/d_x}\right)$

$P_{i,2j+1} = \cos\left(i/10000^{2j/d_x}\right)$

$X = [x_1 \quad x_2 \quad \cdots \quad x_t]^T + [P_{ij}]$

$Y = [<\text{bos}> \quad y_1 \quad \cdots \quad y_m]^T + [P_{ij}]$

For $r = 1, 2, \ldots, N$

$X := \text{LayerNorm}\left(X + \text{MultiHead}\left(XW_Q^{(r)}, XW_K^{(r)}, XW_V^{(r)}\right)\right)$

$X := \text{LayerNorm}\left(X + \max\left\{0, XW_1^{(r)} + b_1^{(r)}\right\}W_2^{(r)} + b_2^{(r)}\right)$

For $r = 1, 2, \ldots, N$

$Y := \text{LayerNorm}\left(Y + \text{MaskedMultiHead}\left(YW_Q^{(r)}, YW_K^{(r)}, YW_V^{(r)}\right)\right)$

$Y := \text{LayerNorm}\left(Y + \text{MultiHead}\left(YW_Q^{(r)}, XW_K^{(r)}, XW_V^{(r)}\right)\right)$

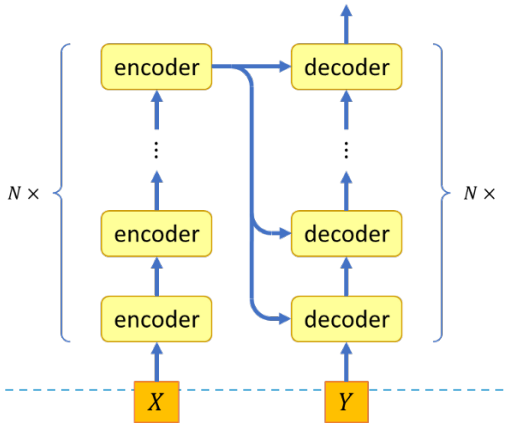
$Y := \text{LayerNorm}\left(Y + \max\left\{0, YW_3^{(r)} + b_3^{(r)}\right\}W_4^{(r)} + b_4^{(r)}\right)$

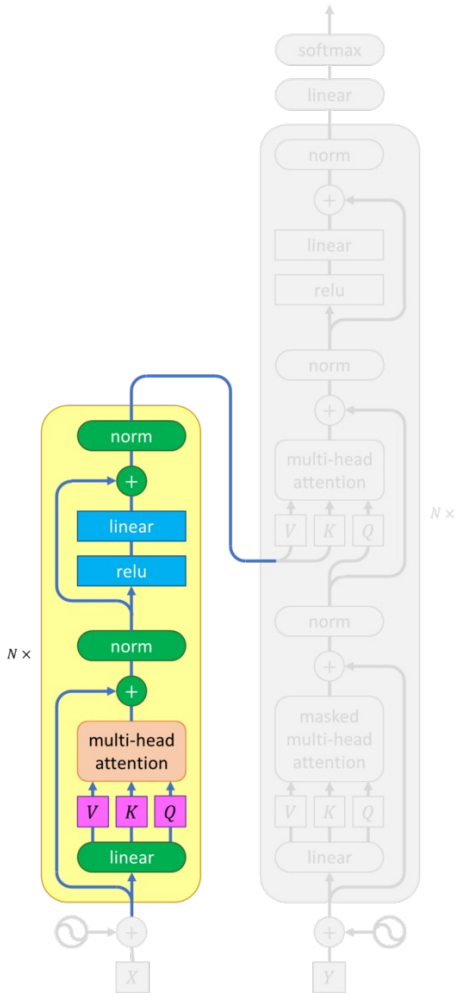
$\text{proba} = \text{softmax}(YW_O)$

# Transformers

The **encoder** consists of the following components:

- Input embeddings ( including the positional embedding => multi-head attention
- Add word embeddings to the output of the attention and normalize
- After normalizations, feed forward neural network




$$P_{i,2j} = \sin\left(i/10000^{2j/d_x}\right)$$

$$P_{i,2j+1} = \cos\left(i/10000^{2j/d_x}\right)$$

$$X = [x_1 \quad x_2 \quad \cdots \quad x_t]^T + [P_{ij}]$$

$$Y = [<\text{bos}> \quad y_1 \quad \cdots \quad y_m]^T + [P_{ij}]$$

For $r = 1,2,\ldots,N$

$$X := \text{LayerNorm}\left(X + \text{MultiHead}(XW_Q, XW_K, XW_V)\right)$$

$$X := \text{LayerNorm}\left(X + \max\left\{0, XW_1^{(r)} + b_1^{(r)}\right\}W_2^{(r)} + b_2^{(r)}\right)$$

For $r = 1,2,\ldots,N$

$$Y := \text{LayerNorm}\left(Y + \text{MaskedMultiHead}(YW_Q, YW_K, YW_V)\right)$$

$$Y := \text{LayerNorm}\left(Y + \text{MultiHead}(YW_Q, XW_K, XW_V)\right)$$

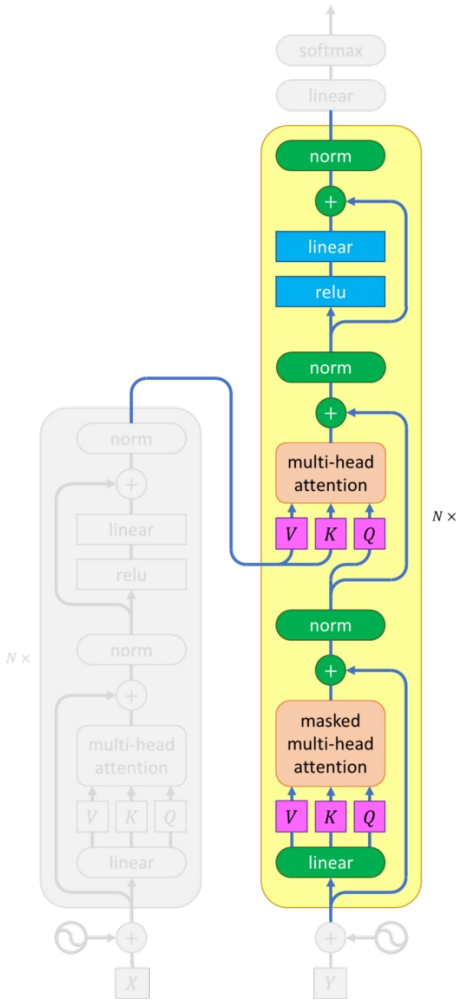
$$Y := \text{LayerNorm}\left(Y + \max\left\{0, YW_3^{(r)} + b_3^{(r)}\right\}W_4^{(r)} + b_4^{(r)}\right)$$

$$\text{proba} = \text{softmax}(YW_O)$$

# Transformers

**Decoder** consists of the following components:

- Output embeddings Y => fed into multi-head attention
- Information is passed to another multi-head attention (every position in the decoder to see all the positions in the input sequence)
- Result is passed through a feed forward neural network




$$P_{i,2j} = \sin\left(i/10000^{2j/d_x}\right)$$

$$P_{i,2j+1} = \cos\left(i/10000^{2j/d_x}\right)$$

$$X = [x_1 \quad x_2 \quad \cdots \quad x_t]^T + [P_{ij}]$$

$$Y = [<\text{bos}> \quad y_1 \quad \cdots \quad y_m]^T + [P_{ij}]$$

For $r = 1,2,\ldots,N$

$$X := \text{LayerNorm}\left(X + \text{MultiHead}(XW_Q, XW_K, XW_V)\right)$$

$$X := \text{LayerNorm}\left(X + \max\left\{0, XW_1^{(r)} + b_1^{(r)}\right\}W_2^{(r)} + b_2^{(r)}\right)$$

For $r = 1,2,\ldots,N$

$$Y := \text{LayerNorm}\left(Y + \text{MaskedMultiHead}(YW_Q, YW_K, YW_V)\right)$$

$$Y := \text{LayerNorm}\left(Y + \text{MultiHead}(YW_Q, XW_K, XW_V)\right)$$

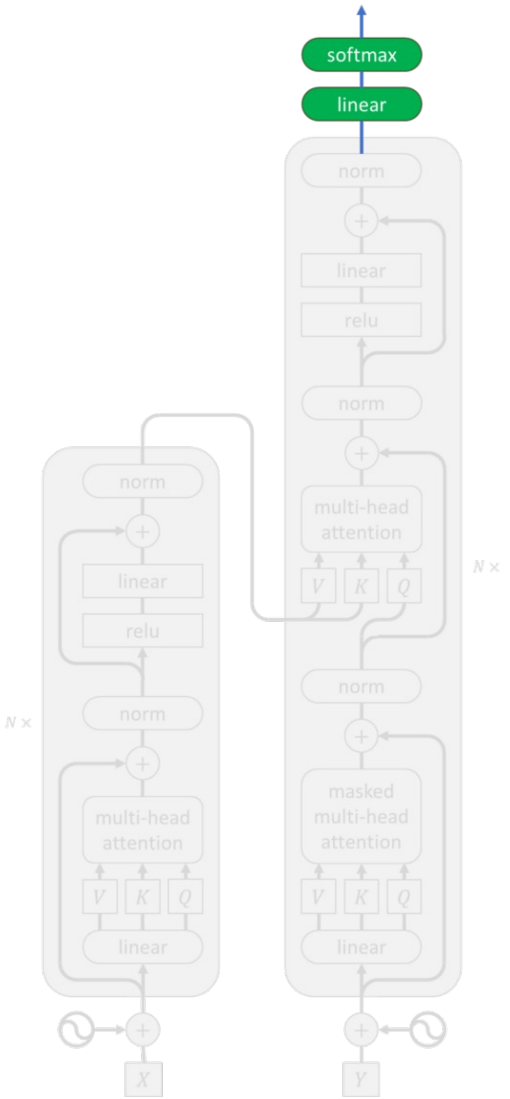
$$Y := \text{LayerNorm}\left(Y + \max\left\{0, YW_3^{(r)} + b_3^{(r)}\right\}W_4^{(r)} + b_4^{(r)}\right)$$

$$\text{proba} = \text{softmax}(YW_O)$$

# Transformers

**Classifier:**

- Last step of the transformers model
- Linear transformation and softmax to convert decoder output to predicted next token probabilities




$$P_{i,2j} = \sin\left(i/10000^{2j/d_x}\right)$$

$$P_{i,2j+1} = \cos\left(i/10000^{2j/d_x}\right)$$

$$X = [x_1 \quad x_2 \quad \cdots \quad x_t]^T + [P_{ij}]$$

$$Y = [<\text{bos}> \quad y_1 \quad \cdots \quad y_m]^T + [P_{ij}]$$

For $r = 1,2,\ldots,N$

$$X := \text{LayerNorm}\left(X + \text{MultiHead}(XW_Q, XW_K, XW_V)\right)$$

$$X := \text{LayerNorm}\left(X + \max\left\{0, XW_1^{(r)} + b_1^{(r)}\right\}W_2^{(r)} + b\right.$$

For $r = 1,2,\ldots,N$

$$Y := \text{LayerNorm}\left(Y + \text{MaskedMultiHead}(YW_Q, YW_K,\right.$$

$$Y := \text{LayerNorm}\left(Y + \text{MultiHead}(YW_Q, XW_K, XW_V)\right)$$

$$Y := \text{LayerNorm}\left(Y + \max\left\{0, YW_3^{(r)} + b_3^{(r)}\right\}W_4^{(r)} + b\right.$$

$$\text{proba} = \text{softmax}(YW_O)$$

# 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

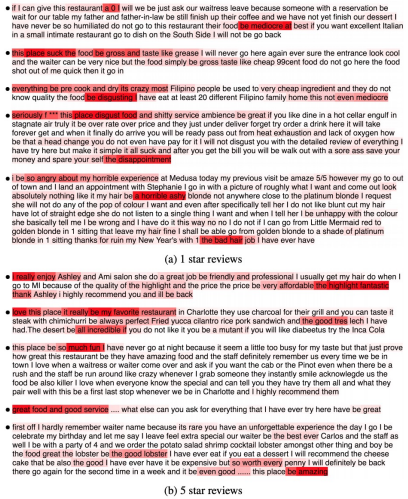
## Learning contextual representations of text

- 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
- The transformer has had great success in NLP. Many pre-trained models such as GPT-2, GPT-3, BERT, XLNet, and RoBERTa demonstrate the ability of transformers to perform a wide variety of NLP-related tasks such as machine translation, document summarization, document generation, named entity recognition, and video understanding.

# 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)



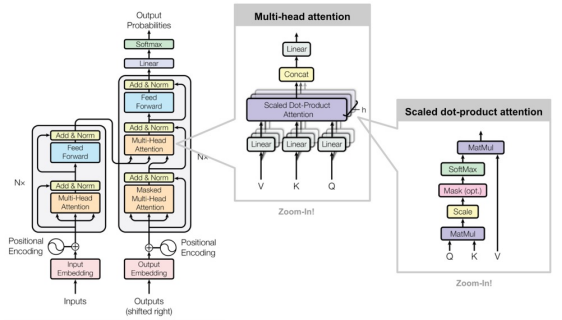
(a) 1 star reviews

(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 !**

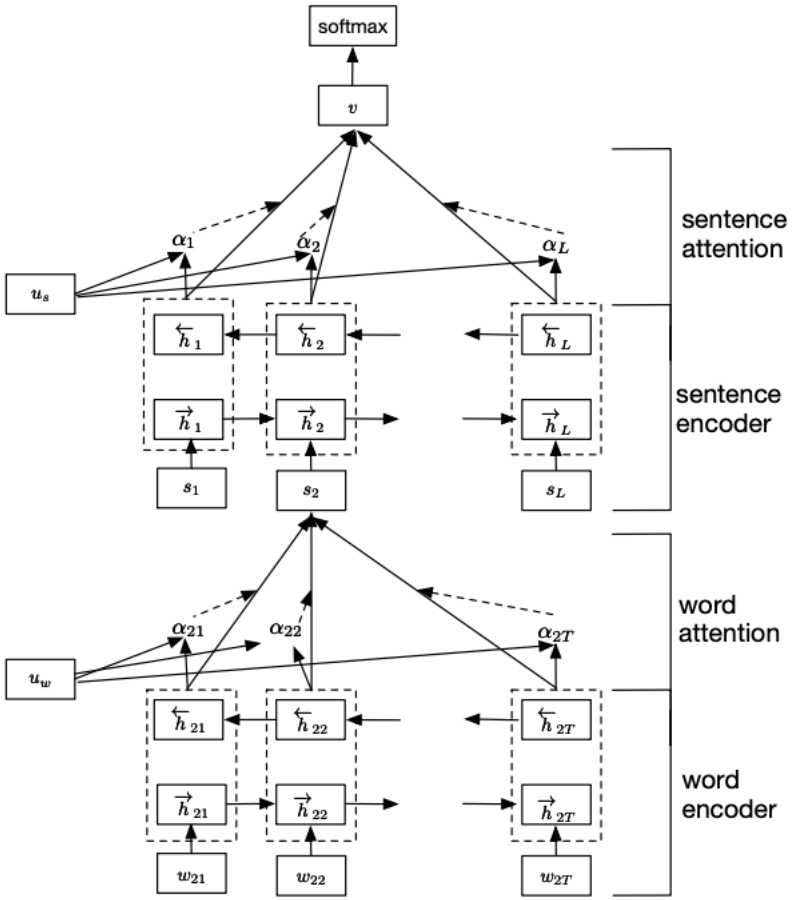
# Thank you for your attention

# GOODBYE !

February 20th, 2023

# 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<br>We use the negative log likelihood of the correct labels as training loss where $\boldsymbol{j}$ is the label of document $\boldsymbol{d}$. |
| $u_i = \tanh(W_s h_i + b_s)$<br>$\alpha_i = \frac{e^{u_i^T u_s}}{\sum_i e^{u_i^T u_s}}$<br>$v = \sum_i \alpha_i h_i$ | $\boldsymbol{\alpha_i}$ the sentence attention vector<br>$\boldsymbol{v}$ is the document vector that summarizes all the information of sentences in a document. |
| $\overrightarrow{h_i} = \overrightarrow{GRU}(s_i), i \in [L, 1]$<br>$\overleftarrow{h_i} = \overleftarrow{GRU}(s_t), t \in [L, 1]$<br>$h_i = [\overrightarrow{h_i}; \overleftarrow{h_i}]$ | $h_i$ summarizes the neighbor sentences around sentence $\boldsymbol{s_i}$ but still focus on i-th sentence. |
| $u_{it} = \tanh(W_w h_{it} + b_w)$<br>$\alpha_{it} = \frac{e^{u_{it}^T u_w}}{\sum_t e^{u_{it}^T u_w}}$<br>$s_i = \sum_i \alpha_{it} h_{it}$ | 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 $\boldsymbol{s_i}$ |
| $x_{it} = W_e, w_{it}, t \in [1, T],$<br>$\overrightarrow{h_{it}} = \overrightarrow{GRU}(x_{it}), t \in [1, T],$<br>$\overleftarrow{h_{it}} = \overleftarrow{GRU}(x_{it}), t \in [1, T].$<br>$h_{it} = [\overrightarrow{h_{it}}; \overleftarrow{h_{it}}]$ | 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. |