# “IA et langage naturel” 2

Everything else (in very short)

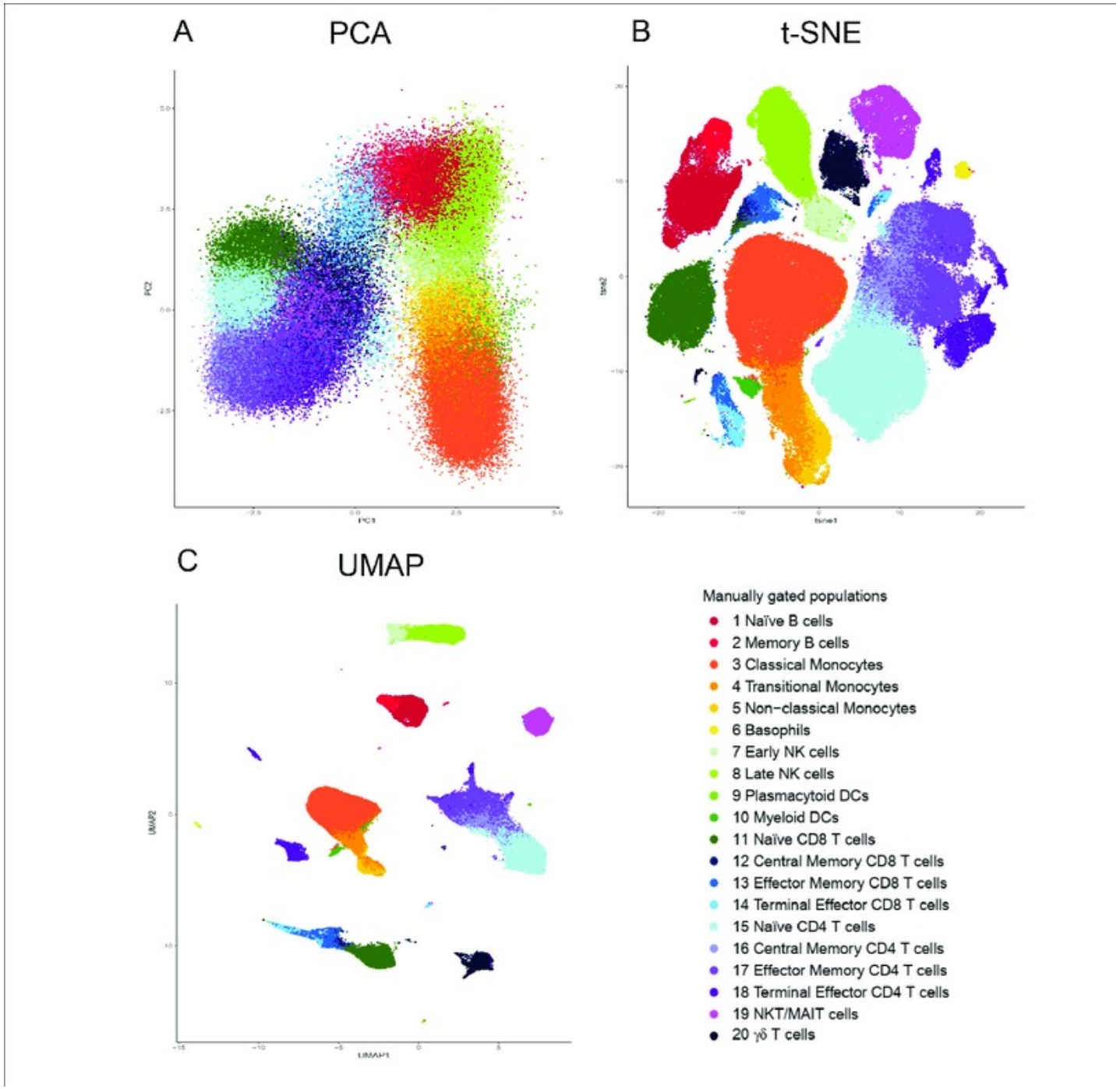
Prof. Stéphane DUPONT

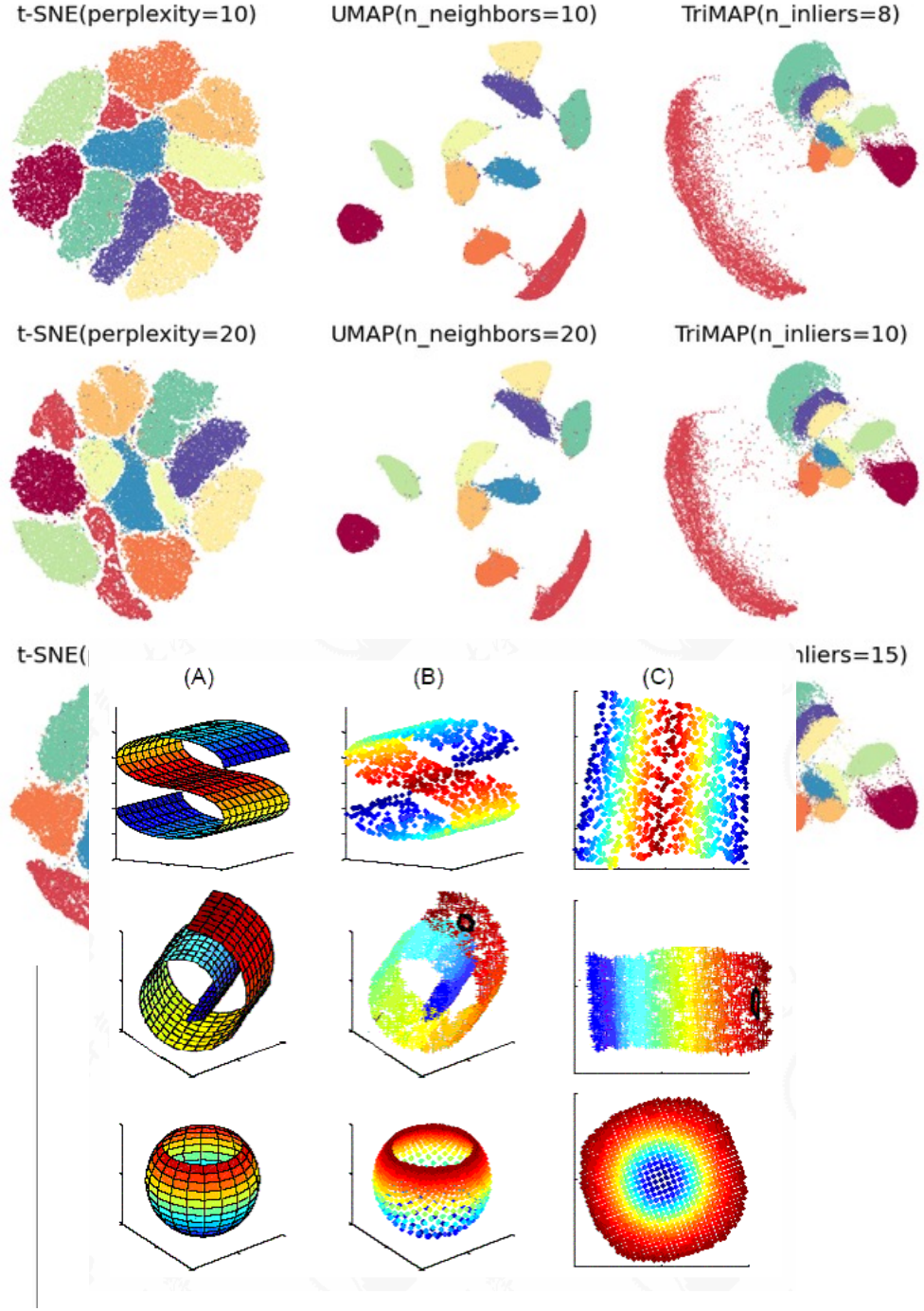
2022-2023

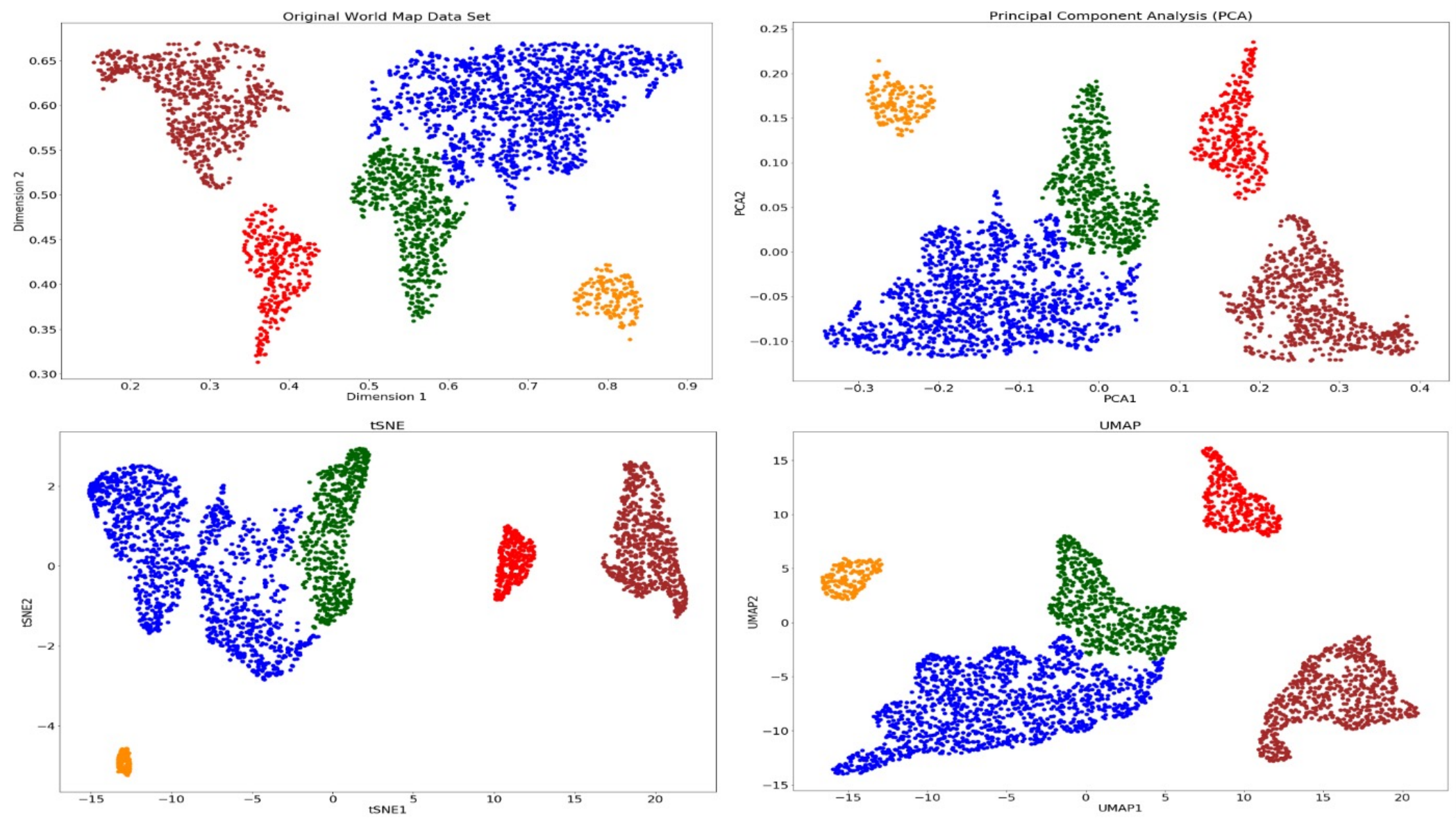
# OUTLINE

- Introduction
  - Historique
  - Catégories d'applications et exemples
  - Elements de linguistique: domaine, structure de phrase, ambiguïtés (lexicales et syntaxiques), contexte
- Algorithmes et outils
  - Descripteurs (sparse, denses, contextuels,...)
  - Réduction de dimension
  - Modèles de langage
  - Architectures des modèles (CNN, RNNs, Transformers, …)
  - Classification de texte
- Méthodologie
  - Databases, data augmentation, métriques (y compris NMT)
- Conclusions et point de vue critique

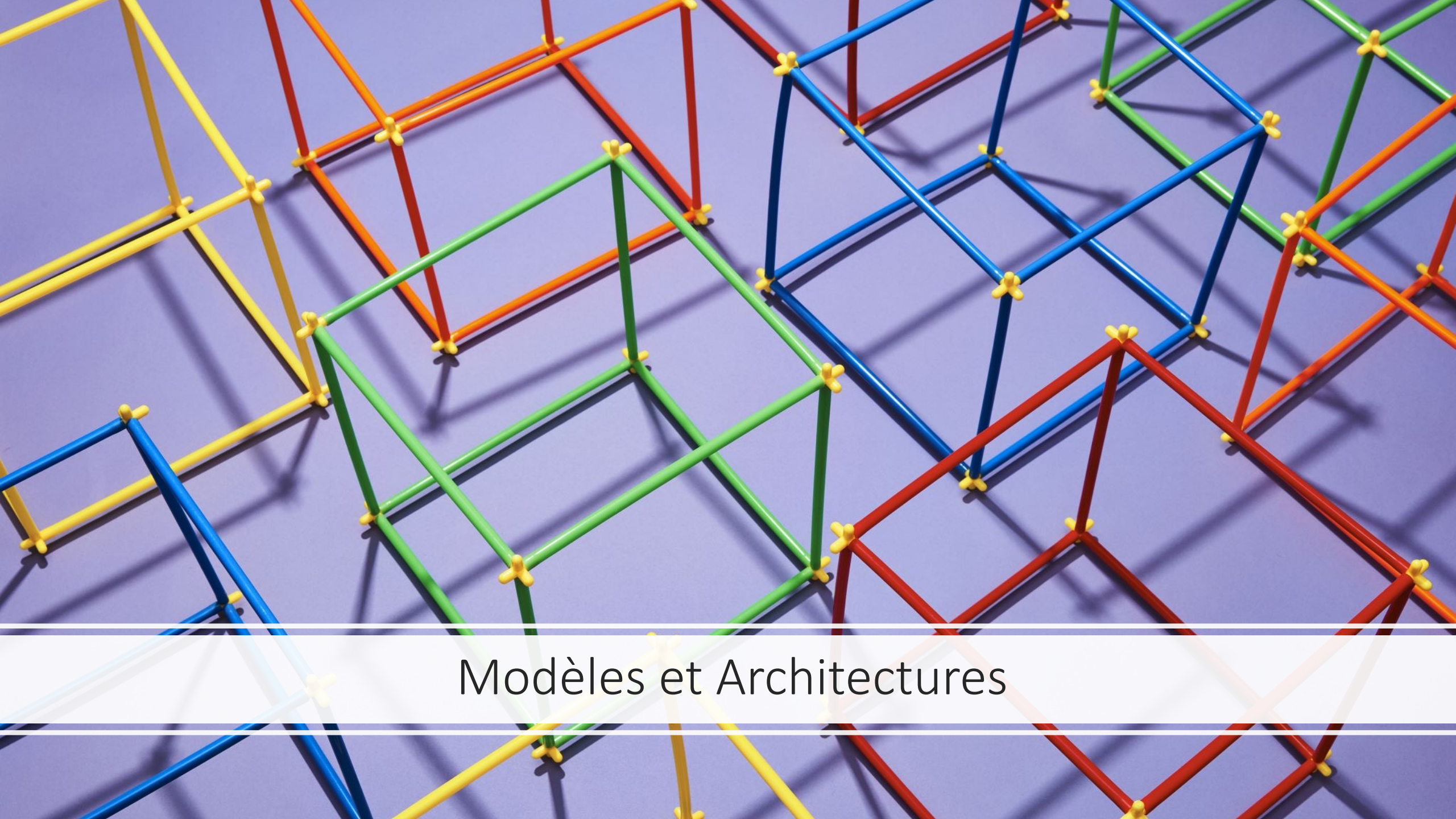
# Réduction de dimensionalité


A
PCA
B
t-SNE
C
UMAP
Manually gated populations
1 Naïve B cells
2 Memory B cells
3 Classical Monocytes
4 Transitional Monocytes
5 Non−classical Monocytes
6 Basophils
7 Early NK cells
8 Late NK cells
9 Plasmacytoid DCs
10 Myeloid DCs
11 Naïve CD8 T cells
12 Central Memory CD8 T cells
13 Effector Memory CD8 T cells
14 Terminal Effector CD8 T cells
15 Naïve CD4 T cells
16 Central Memory CD4 T cells
17 Effector Memory CD4 T cells
18 Terminal Effector CD4 T cells
19 NKT/MAIT cells
20 γδ T cells


t-SNE(perplexity=10)
UMAP(n_neighbors=10)
TriMAP(n_inliers=8)
t-SNE(perplexity=20)
UMAP(n_neighbors=20)
TriMAP(n_inliers=10)
t-SNE(
nliers=15)
(A)
(B)
(C)


Original World Map Data Set
Dimension 2
Dimension 1
Principal Component Analysis (PCA)
PCA2
PCA1
tSNE
tSNE2
tSNE1
UMAP
UMAP2
UMAP1

- https://scikit-learn.org/stable/modules/manifold.html
- https://scikit-learn.org/stable/modules/unsupervised_reduction.html



# Modèles et Architectures

# Everything else (in very short)

- S’approprier quelques méthodes
  - Gensim: **Word2Vec**
  - Sklearn: **TF & TF-IDF**
  - Spacy: **POS & NER**
  - Transformers: **Embedding**

| Type de representation/features | Outil pratiqué |
| --- | --- |
| dense non-contextuelle | sklearn |
| sparse | gensim |
| symbolique (nature des mots) | spacy |
| dense contextuelle | transformers |

# Prédire la suite de la phrase

- her ...

# Prédire la suite de la phrase

- she finally printed her ...

# Prédire la suite de la phrase

- After installing the toner into the printer, she finally printed her ...

# Prédire la suite de la phrase

- She went to the stationery store to buy more toner. After installing the toner into the printer, she finally printed her ...

# Prédire la suite de la phrase

- When she tried to print her tickets, she found the printer was out of toner. She went to the stationery store to buy more toner. After installing the toner into the printer, she finally printed her ...

https://huggingface.co/gpt2?text=When+she+tried+to+print+her+ticket%2C+she+found+the+printer+was+out+of+toner.+She+went+to+the+stationery+store+to+buy+more+toner.+After+installing+the+toner+into+the+printer%2C+she+finally+printed+her

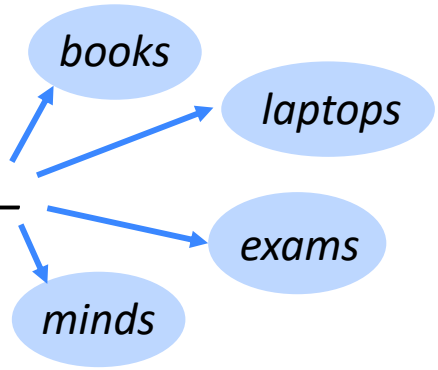
https://huggingface.co/camembert-base?text=Lorsqu%27elle+a+essayé+d%27imprimer+ses+billets%2C+elle+a+constaté+que+l%27imprimante+n%27avait+plus+de+toner.+Elle+est+allée+à+la+papeterie+pour+acheter+plus+de+toner.+Après+avoir+installé+le+toner+dans+l%27impr

# Language Models

# Language Modeling

- **Language Modeling** is the task of predicting what word comes next.

*the students opened their* _____




- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$ :

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**.

Chain rule = formule des probabilité composés

# Language Modeling

- You can also think of a Language Model as a system that assigns probability to a piece of text.

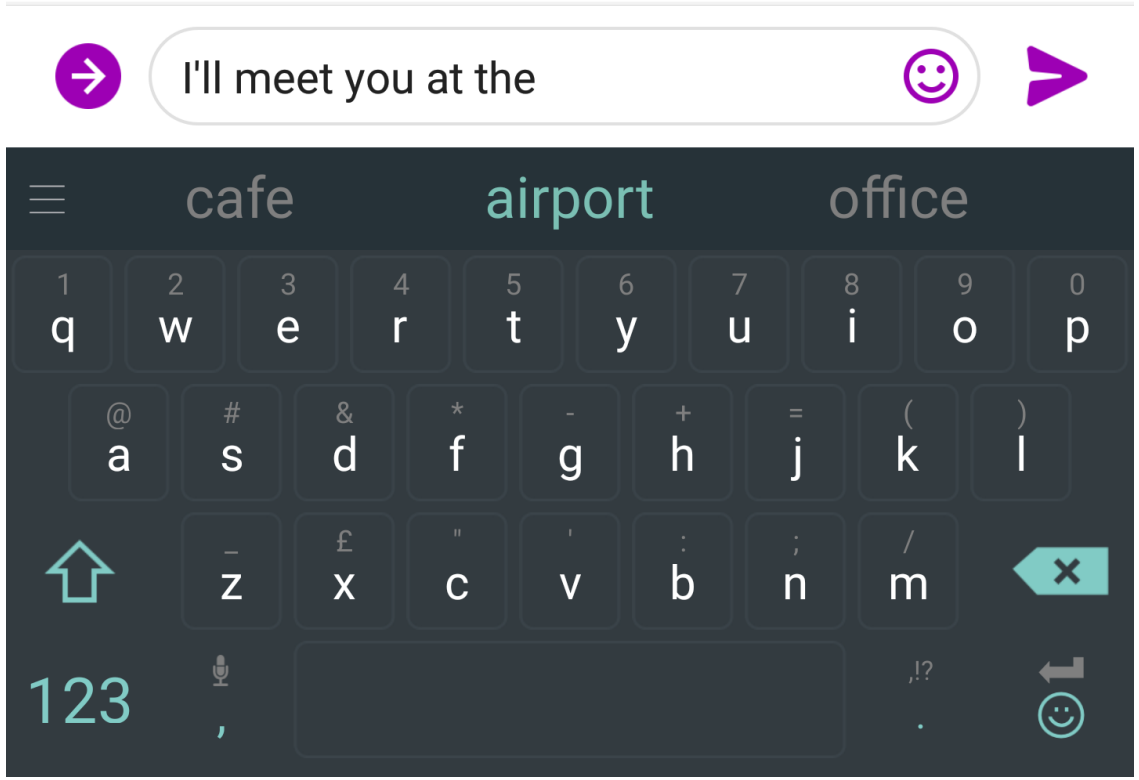
- For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

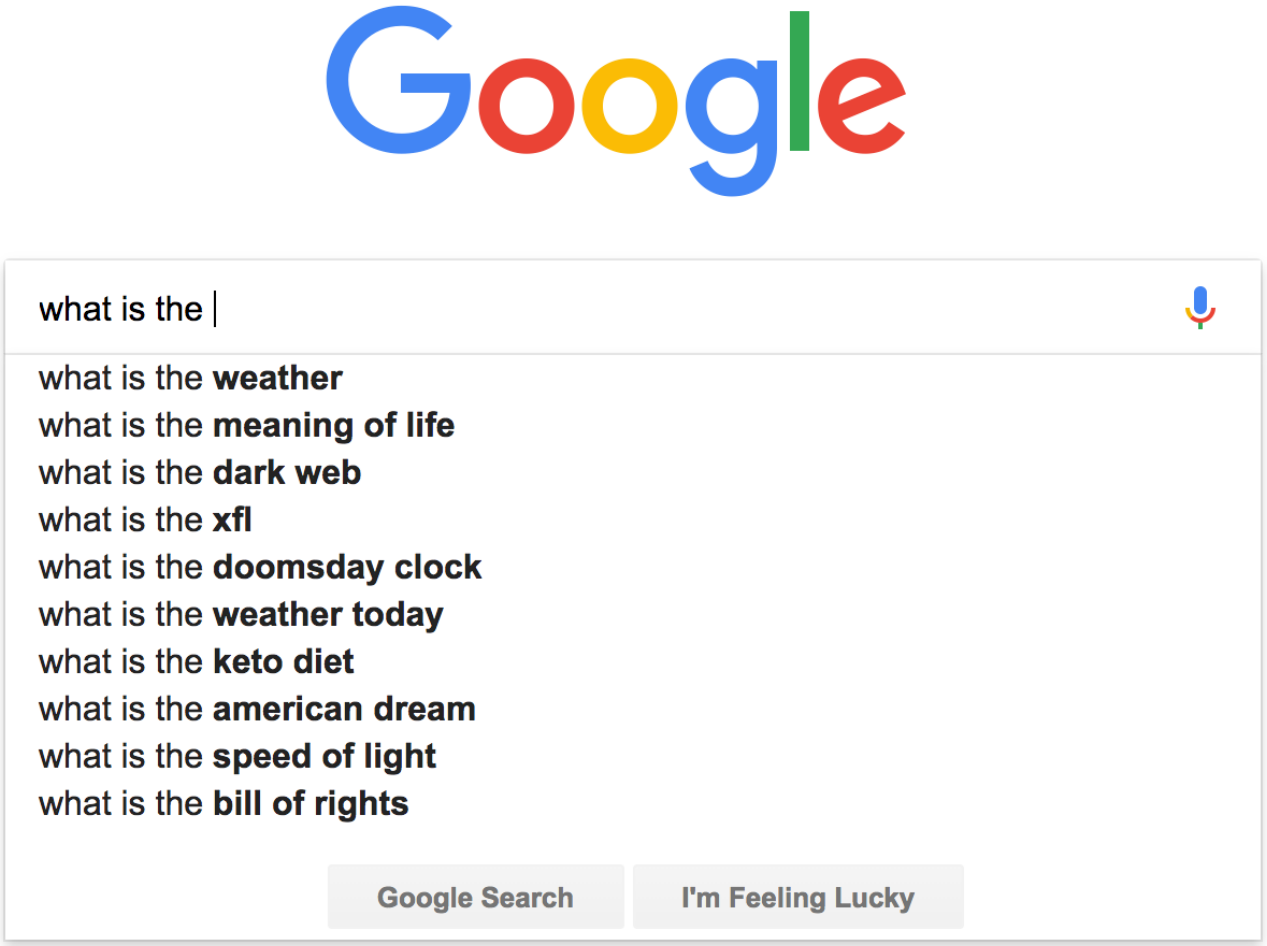
$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

# You use Language Models every day!



# You use Language Models every day!



# Generating text with a RNN Language Model

Let's have some fun!

- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Obama speeches:

*The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.*

**Source:** https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

# n-gram Language Models

*the students opened their* _____

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn an *n*-gram Language Model!

- Definition: A *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”

- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

# n-gram Language Models

- First we make a Markov assumption: $x^{(t+1)}$ depends only on the preceding *n-1* words.

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n-1 \text{ words}}) \quad \text{(assumption)}$$

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(definition of conditional prob)}$$

- **Question:** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(statistical approximation)}$$

# n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ *students opened their* _____

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

# Sparsity Problems with n-gram Language Models

Sparsity Problem 1

**Problem:** What if *"students opened their* $\boldsymbol{w}$*"* never occurred in data? Then $\boldsymbol{w}$ has probability 0!

**(Partial) Solution:** Add small $\delta$ to the count for every $\boldsymbol{w} \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Sparsity Problem 2

**Problem:** What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}$!

**(Partial) Solution:** Just condition on *"opened their"* instead. This is called *backoff*.

**Note:** Increasing *n* makes sparsity problems *worse.* Typically we can't have *n* bigger than 5.

# Modèles de langage

- When she tried to print her tickets, she found the printer was out of toner. She went to the stationery store to buy more toner. After installing the toner into the printer, she finally printed her ...

… demanderait au minimum un n-gram avec n=31, ce qui est impossible car on arriverait à une situation de ”sparsité” extrême:

- toutes les séquences de 31 mots présentes dans le corpus d’apprentissage auraient une probabilité de ~1
- toutes les autres séquence une probabilité de ~0

... et donc aucune capacité de généralisation du modèle à d’autres phrases.

# n-gram Language Models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop*

Business and financial news

*today the* _____

get probability distribution

| | |
|---|---|
| company | 0.153 |
| bank | 0.153 |
| price | 0.077 |
| italian | 0.039 |
| emirate | 0.039 |
| … | |

**Sparsity problem**: not much granularity in the probability distribution

Otherwise, seems reasonable!

* **Try for yourself:** https://nlpforhackers.io/language-models/

# Architectures de Deep Learning

Le n-gram impose une “vision tunnel”

Contexte

Vision par ordinateur

Traitement du langage naturel

”Vision tunnel”

she finally printed her ...

(5-gram)

# Architectures de Deep Learning

Le n-gram impose une “vision tunnel”

Contexte

Vision par ordinateur

Traitement du langage naturel

”Vision tunnel”

she finally printed her ...

(5-gram)

# Architectures de Deep Learning

Contexte

Vision par ordinateur

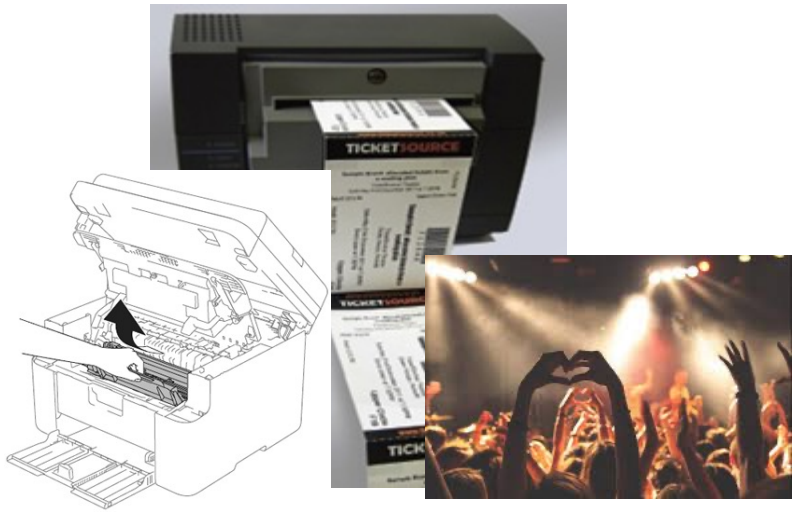
Traitement du langage naturel

”Vision tunnel”

she finally printed her ...

Contexte adéquat

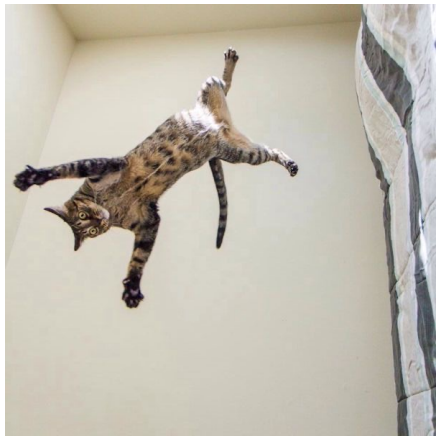
# Architectures de Deep Learning

| Contexte | Vision par ordinateur | Traitement du langage naturel |
| --- | --- | --- |
| ”Vision tunnel” |  | she finally printed her ... |
| Contexte adéquat |  | When she tried to print her tickets, she found the printer was out of toner. She went to the stationery store to buy more toner. After installing the toner into the printer, she finally printed her ... |

# Architectures de Deep Learning

| Contexte | Vision par ordinateur | Traitement du langage naturel |
| --- | --- | --- |
| ”Vision tunnel” |  | Ce chat est ... |
| Contexte adéquat | | |

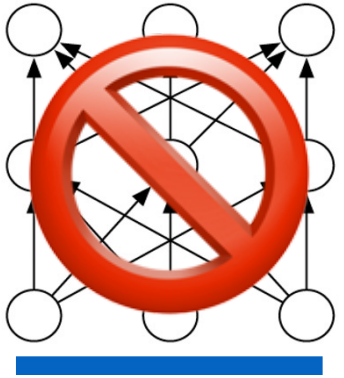
# Architectures de Deep Learning

| Contexte | Vision par ordinateur | Traitement du langage naturel |
| --- | --- | --- |
| ”Vision tunnel” |  | Ce chat est ... |
| Contexte adéquat |  | Ce chat est dingue |

# NLP – Word Prediction = Language Model

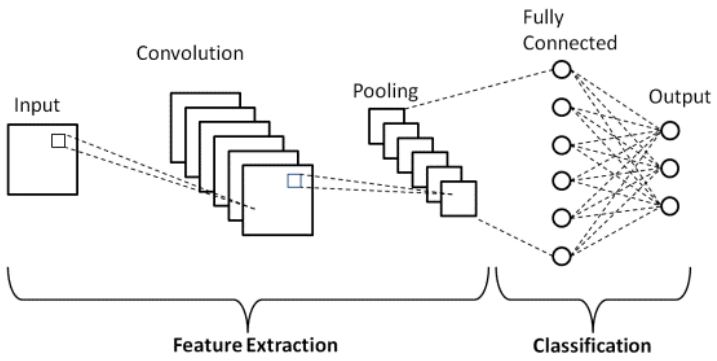
- Challenge
  - Designing better predictors requires modeling the long-term dependencies in the data: a word at position n in a sentence can influence words at positions n+1, but also n+L (L can be a large number)
    - ConvNets (especially using dilated convolutions)
    - RNN
    - LSTM
    - LSTM + Attention
    - DNN + Attention (Transformer)
- Useful Tasks – by fine tuning those models
    - NLG – Natural Language Generation
      - Summarization, Dialog (goal driven / open social), Conversational QA, Storytelling
    - NMT – Neural Machine Translation
    - Question answering
    - Classification

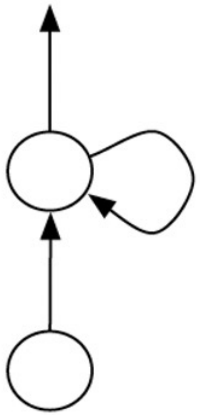
# Architectures de Deep Learning



- Les approches NLP doivent donc chercher à modéliser statistiquement des séquences de texte très longues en # de mots (tout comme les approches de CV le font sur des images avec un très grand nombre de pixels):
  - Quelles sont les solutions proposées?

CNNs | Recurrent Neural Network, and Gating Units (LSTMs) | Attention Models | "Transformers"



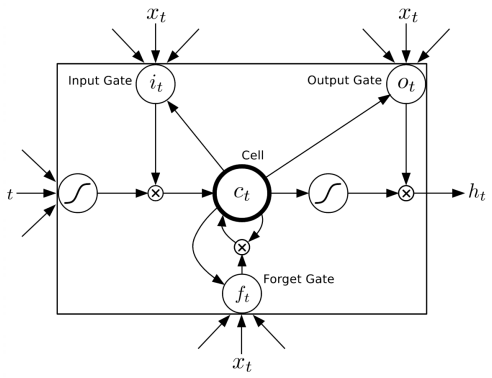
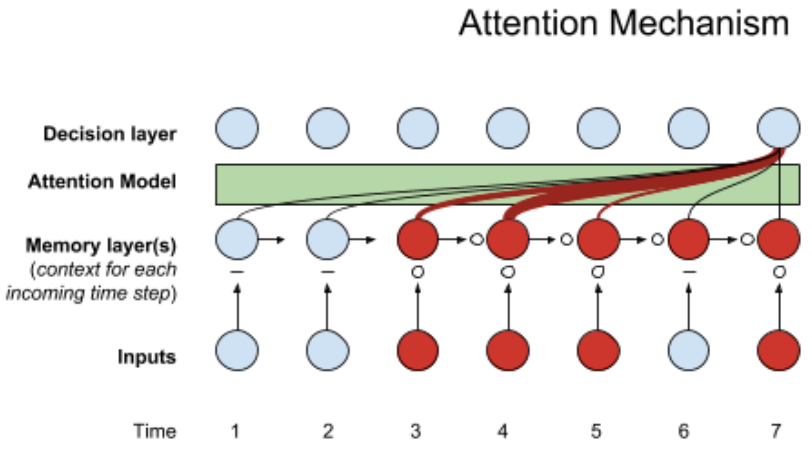



Figure 2: **Long Short-term Memory Cell**

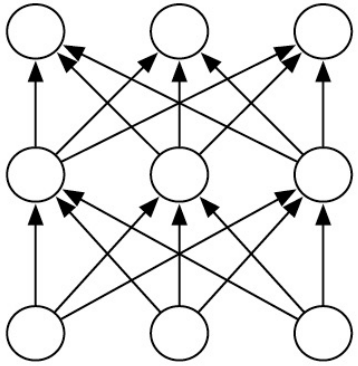


Cascaded processing

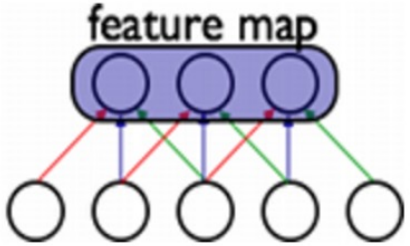
+ later in the course for more about some of those architectures

# Different neuron/layer types

## Feed-forward

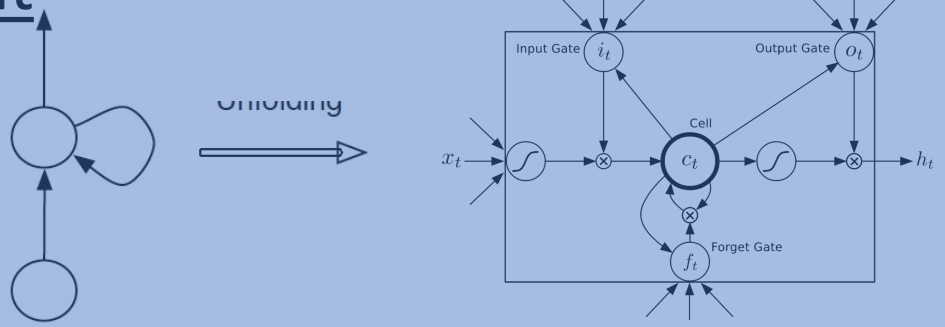


Fully-connected
Feed-forward neural network




Convolutional
Convolutional neural network

## Recurrent




Vanilla LSTM

Recurrent neural network

## + cascading those

## Nowadays, basically not much has changed

- Other **activation** functions (sigmoid, tanh, RELU, …) so output can be other than binary (and f.i. express a probability)
- Make weights **shared** between different neurons
- Make network **sparse**: neurons not receiving every inputs

# MLP in NLP?

# A fixed-window neural Language Model

Modèle Feedforward MLP + couche d'embedding en entrée

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

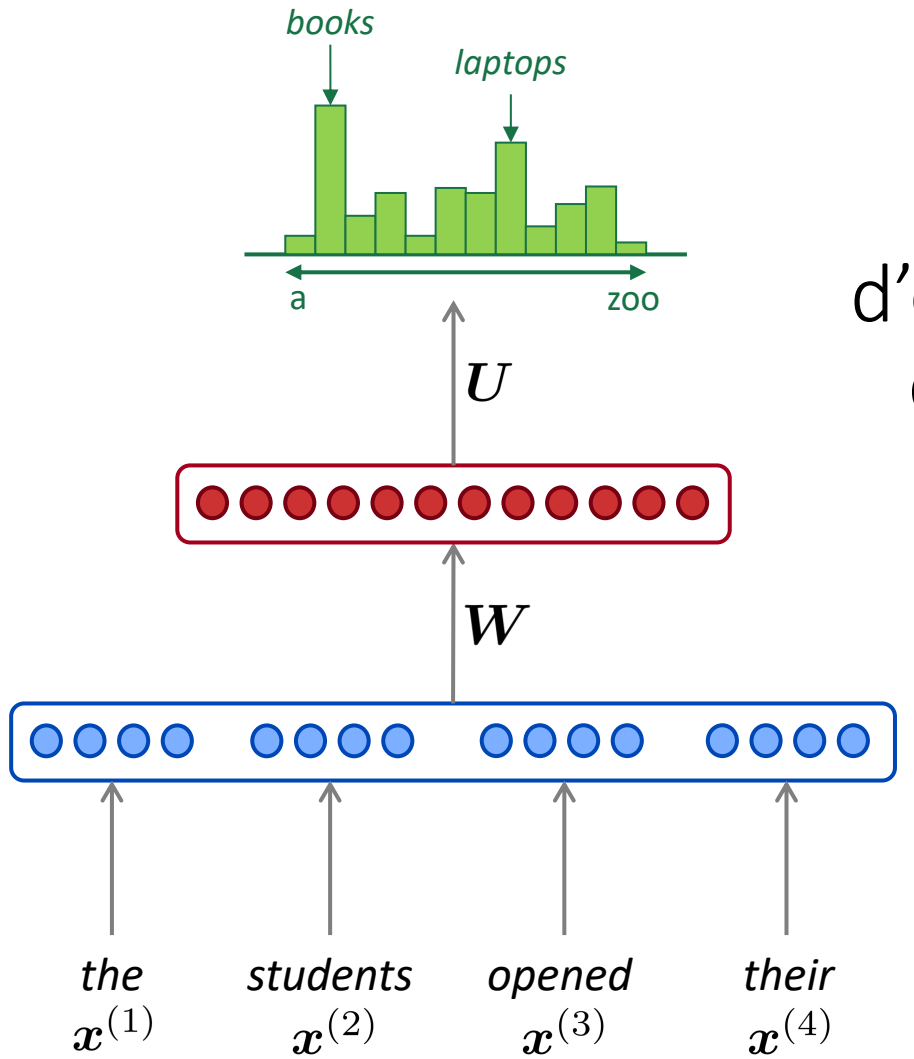
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



# A fixed-window neural Language Model

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

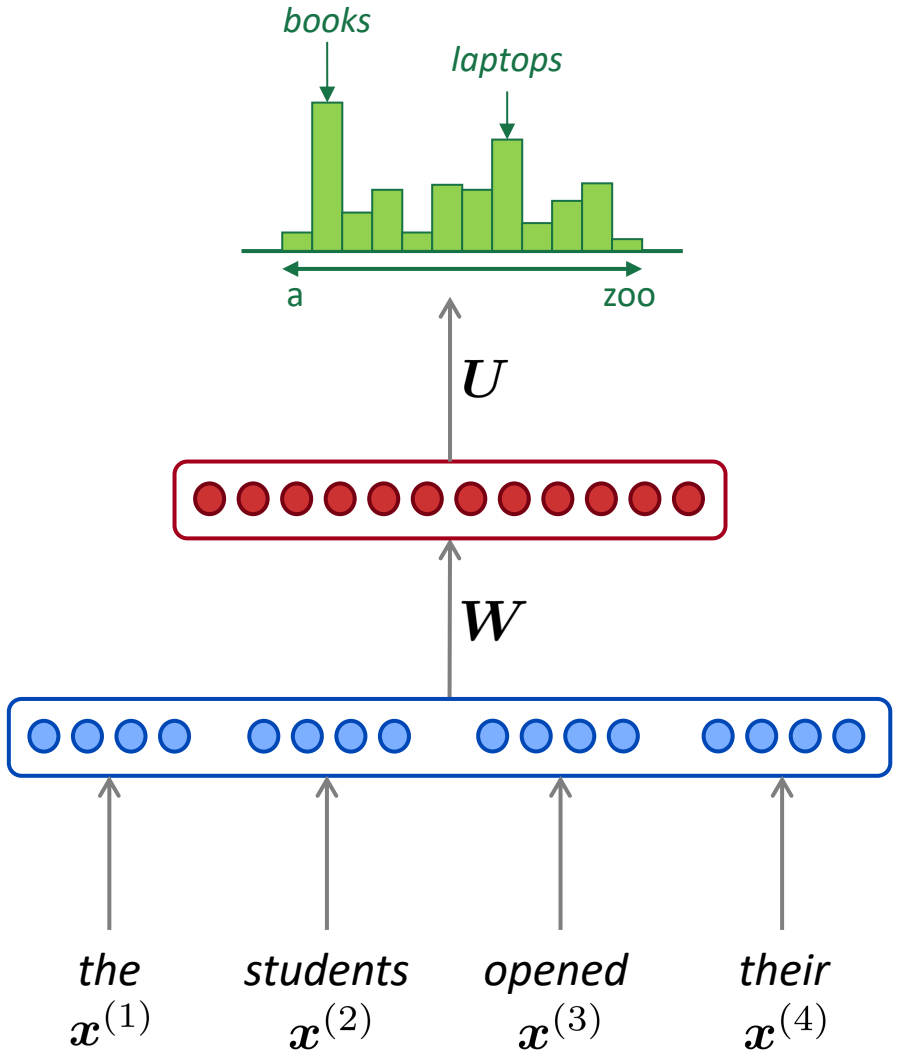
**Improvements** over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$
- Window can never be large enough!
- $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ are multiplied by completely different weights in $\boldsymbol{W}$. No symmetry in how the inputs are processed.

We need a neural architecture that can process *any length input*




Modèle Feedforward MLP

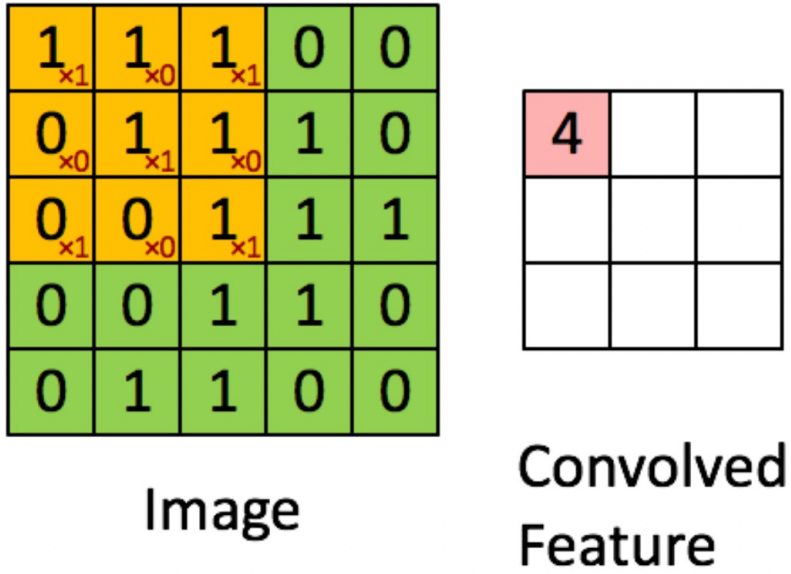
# CNNs in NLP?

# What is a convolution anyway?

- 1d discrete convolution generally: $(f * g)[n] = \sum_{m=-M}^{M} f[n-m]g[m].$

- Convolution is classically used to extract features from images
  - Models position-invariant identification
  - Go to cs231n!

- 2d example →
- Yellow color and red numbers show filter (=kernel) weights
- Green shows input
- Pink shows output




From Stanford UFLDL wiki

# CNNs on text

One layer with kernel of size 3

| | | | | |
|---|---|---|---|---|
| **tentative** | 0.2 | 0.1 | −0.3 | 0.4 |
| **deal** | 0.5 | 0.2 | −0.3 | −0.1 |
| **reached** | −0.1 | −0.3 | −0.2 | 0.4 |
| **to** | 0.3 | −0.3 | 0.1 | 0.1 |
| **keep** | 0.2 | −0.3 | 0.4 | 0.2 |
| **government** | 0.1 | 0.2 | −0.1 | −0.1 |
| **open** | −0.4 | −0.4 | 0.2 | 0.3 |

Apply a **filter** (or **kernel**) of size 3

| | | | |
|---|---|---|---|
| 3 | 1 | 2 | −3 |
| −1 | 2 | 1 | −3 |
| 1 | 1 | −1 | 1 |

| | | | |
|---|---|---|---|
| **t,d,r** | −1.0 | 0.0 | 0.50 |
| **d,r,t** | −0.5 | 0.5 | 0.38 |
| **r,t,k** | −3.6 | -2.6 | 0.93 |
| **t,k,g** | −0.2 | 0.8 | 0.31 |
| **k,g,o** | 0.3 | 1.3 | 0.21 |

+ bias

➔ non-linearity

# CNNs on text

One layer with kernel of size 3 folowed by max-pooling of size 2

| | | | | |
|---|---|---|---|---|
| **Ø** | **0.0** | **0.0** | **0.0** | **0.0** |
| **tentative** | 0.2 | 0.1 | −0.3 | 0.4 |
| **deal** | 0.5 | 0.2 | −0.3 | −0.1 |
| **reached** | −0.1 | −0.3 | −0.2 | 0.4 |
| **to** | 0.3 | −0.3 | 0.1 | 0.1 |
| **keep** | 0.2 | −0.3 | 0.4 | 0.2 |
| **government** | 0.1 | 0.2 | −0.1 | −0.1 |
| **open** | −0.4 | −0.4 | 0.2 | 0.3 |
| **Ø** | **0.0** | **0.0** | **0.0** | **0.0** |

Apply 3 **filters** of size 3

| | | | |
|---|---|---|---|
| 3 | 1 | 2 | −3 |
| −1 | 2 | 1 | −3 |
| 1 | 1 | −1 | 1 |

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 1 | 0 | −1 | −1 |
| 0 | 1 | 0 | 1 |

| | | | |
|---|---|---|---|
| 1 | −1 | 2 | −1 |
| 1 | 0 | −1 | 3 |
| 0 | 2 | 2 | 1 |

| | | | |
|---|---|---|---|
| **Ø,t,d** | −0.6 | 0.2 | 1.4 |
| **t,d,r** | −1.0 | 1.6 | −1.0 |
| **d,r,t** | −0.5 | −0.1 | 0.8 |
| **r,t,k** | −3.6 | 0.3 | 0.3 |
| **t,k,g** | −0.2 | 0.1 | 1.2 |
| **k,g,o** | 0.3 | 0.6 | 0.9 |
| **g,o,Ø** | −0.5 | −0.9 | 0.1 |
| **Ø** | −Inf | −Inf | −Inf |

| | | | |
|---|---|---|---|
| **Ø,t,d,r** | −0.6 | 1.6 | 1.4 |
| **d,r,t,k** | −0.5 | 0.3 | 0.8 |
| **t,k,g,o** | 0.3 | 0.6 | 1.2 |
| **g,o,Ø,Ø** | −0.5 | −0.9 | 0.1 |

Several layers in sequence (deep learning)

# CNNs on text

- Yoon Kim (2014): Convolutional Neural Networks for Sentence Classification. EMNLP 2014. https://arxiv.org/pdf/1408.5882.pdf
- A variant of convolutional NNs of Collobert, Weston et al. (2011)
- Natural Language Processing (almost) from Scratch
  - Goal: Sentence classification
  - Mainly positive or negative sentiment of a sentence
  - Other tasks like:
    - Subjective or objective language sentence
    - Question classification: about person, location, number, …

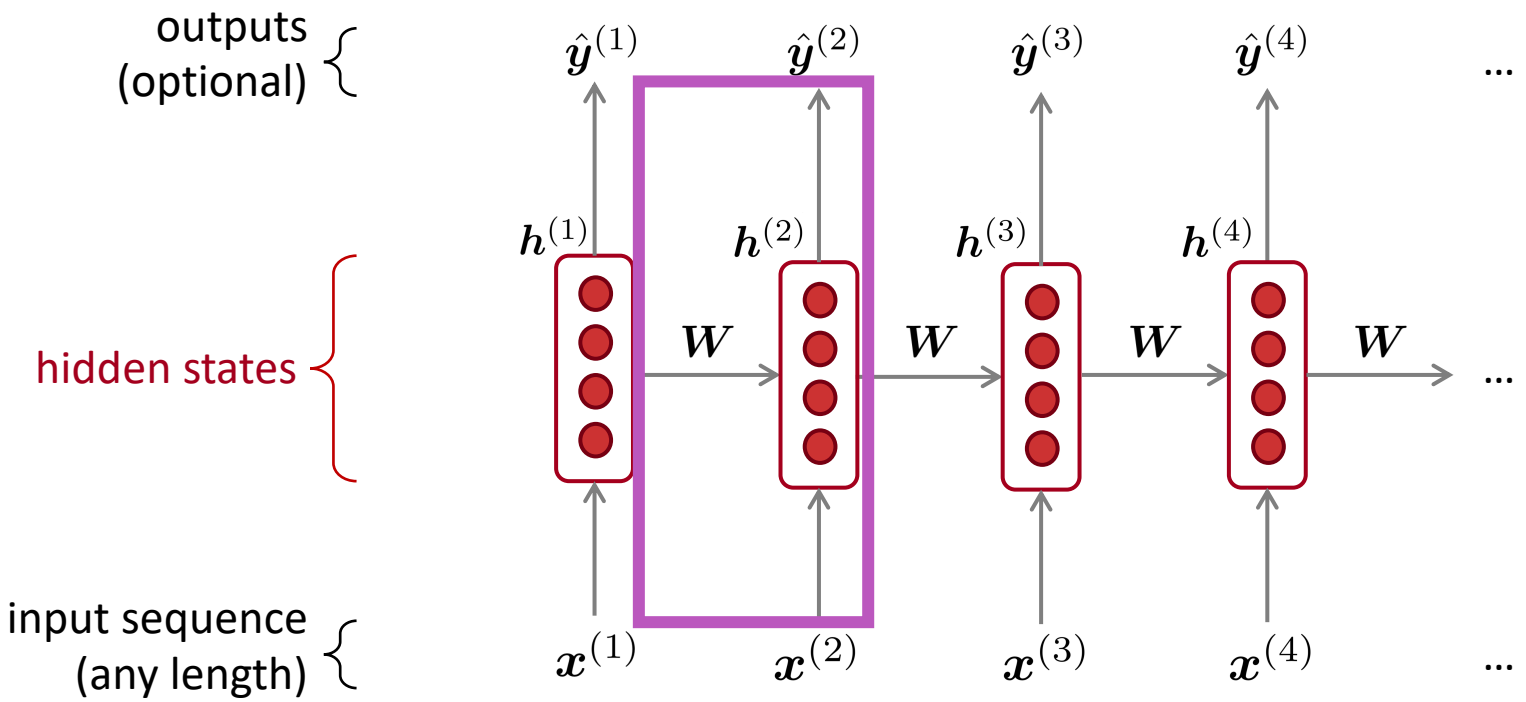
# RNNs in NLP

# Recurrent Neural Networks (RNN)

A family of neural architectures

**Core idea:** Apply the same weights $\boldsymbol{W}$ *repeatedly*

Modèle Récurrent RNN



# A Simple RNN Language Model

Modèle Récurrent RNN + couche d'embedding en entrée

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

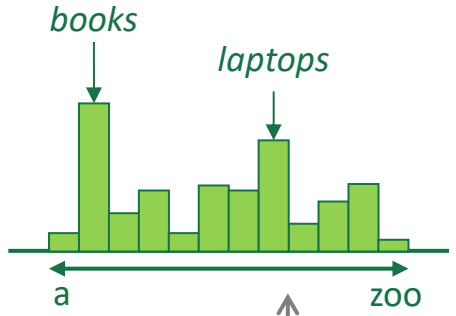
word embeddings

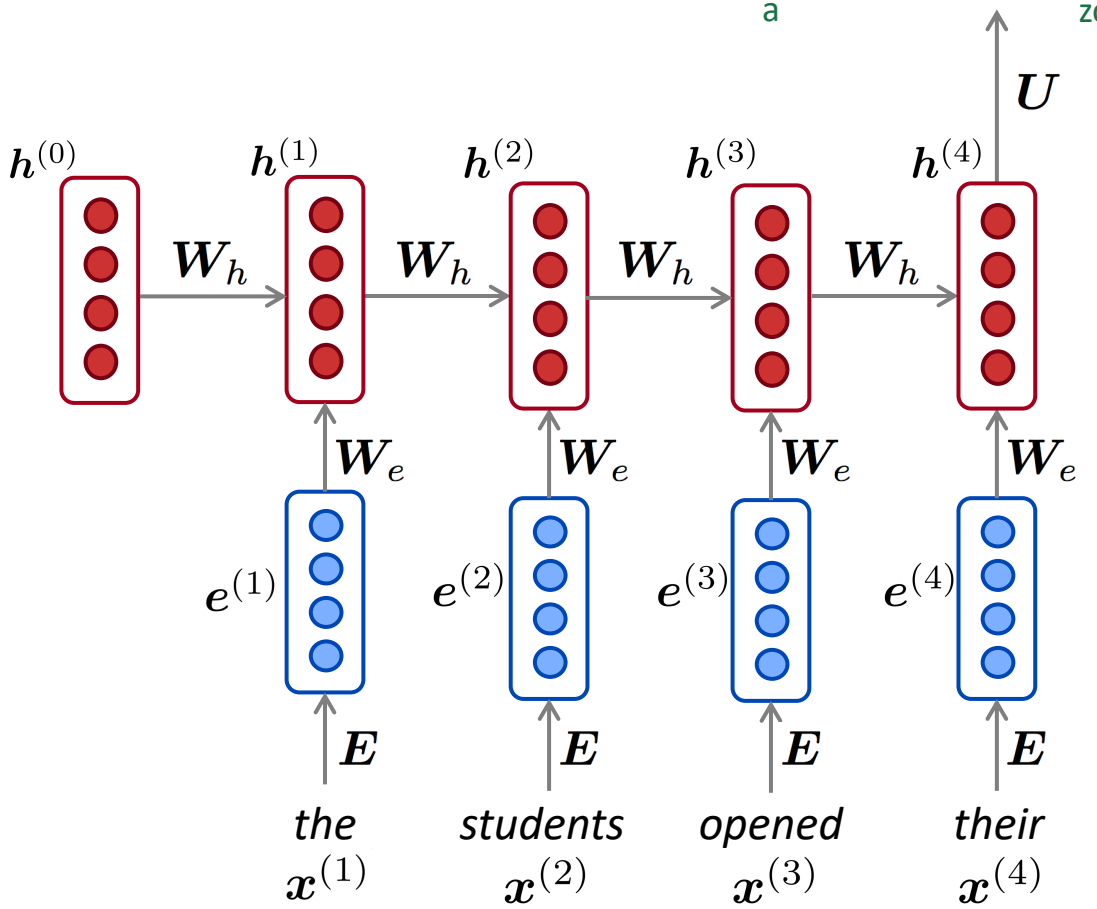
$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$

$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$




***Note**: this input sequence could be much longer, but this slide doesn't have space!*

# RNN Language Models

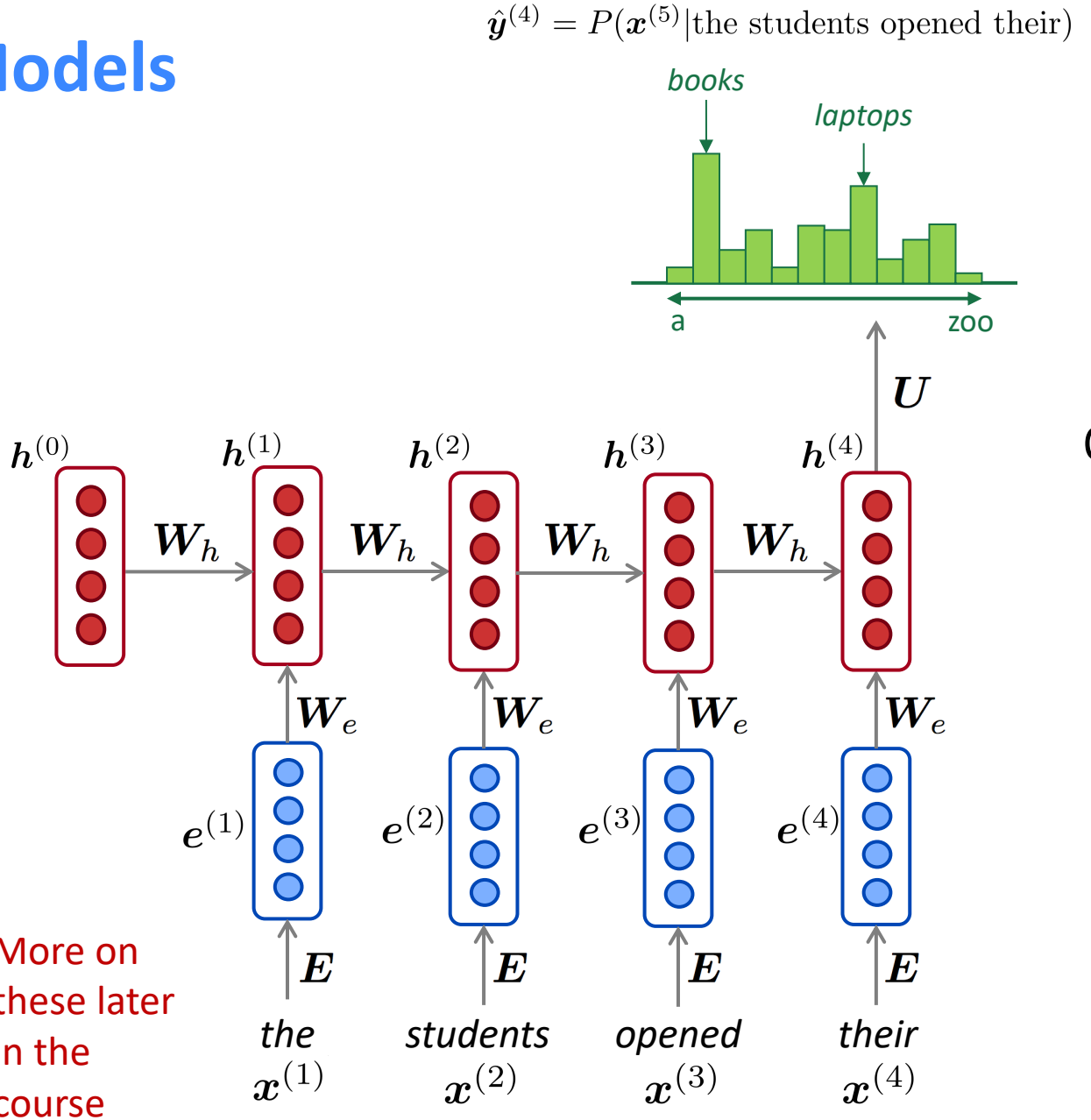
RNN **Advantages**:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later in the course

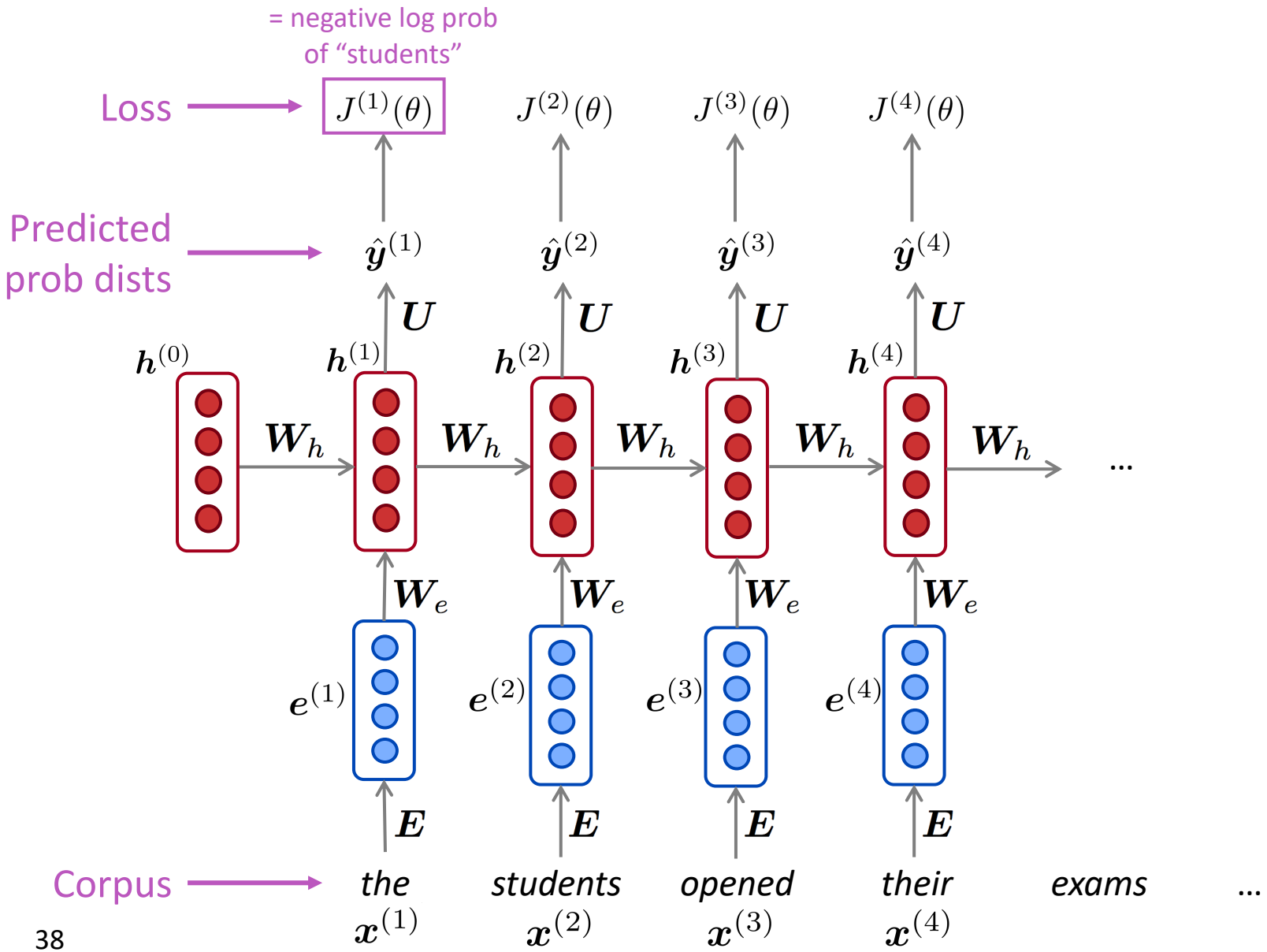



Modèle Récurrent RNN + couche d'embedding en entrée

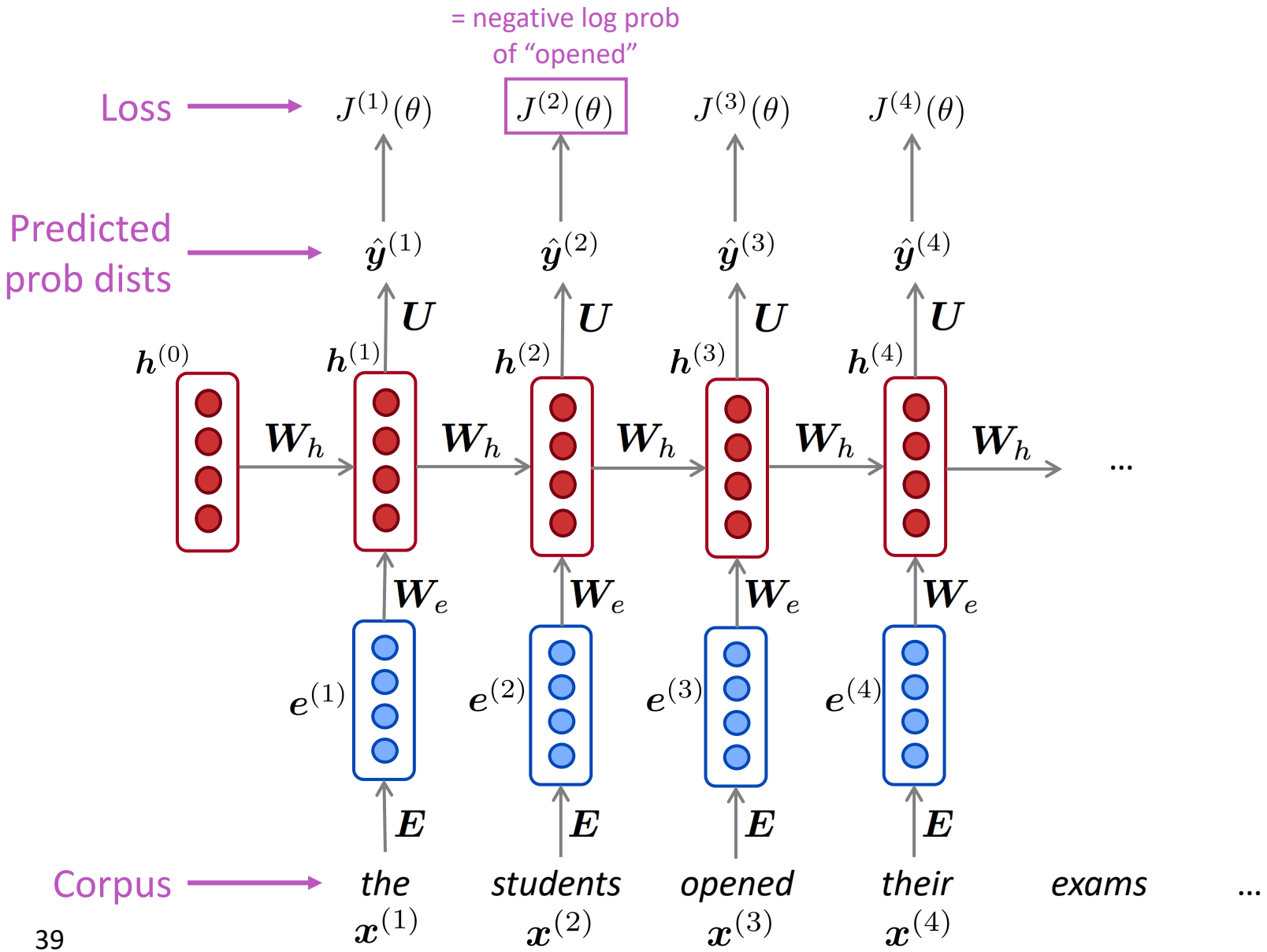
# Méthodologies d'Apprentissage

- L'algorithme d'apprentissage
  - Descente de gradient
  - Backpropagation = permet de calculer les gradient du "Loss" par rapport à tous les paramètres du modème
- Les autres éléments de méthodologie et autres bonnes pratiques ou "astuces" sont recommandées ici:
  - Dataset:
    - train / validation / test
  - Eviter le surentraînement (overfitting):
    - Régularisation: dropout, batch norm, …
    - Arrêt précoce (early stopping)
  - Metaparamètres de l'apprentissage:
    - Epochs, iteration, batch size, …
  - ….

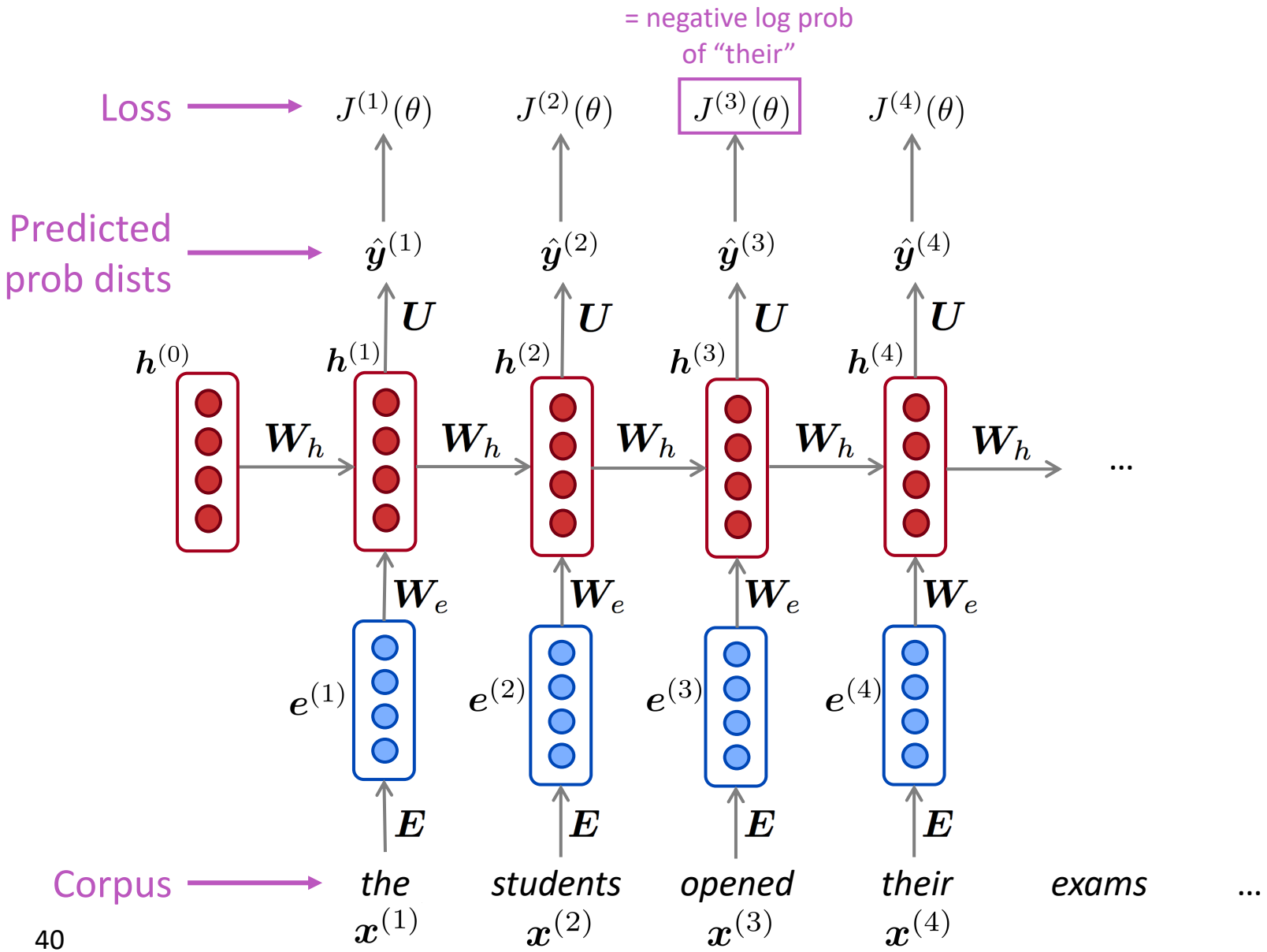
# Training an RNN Language Model



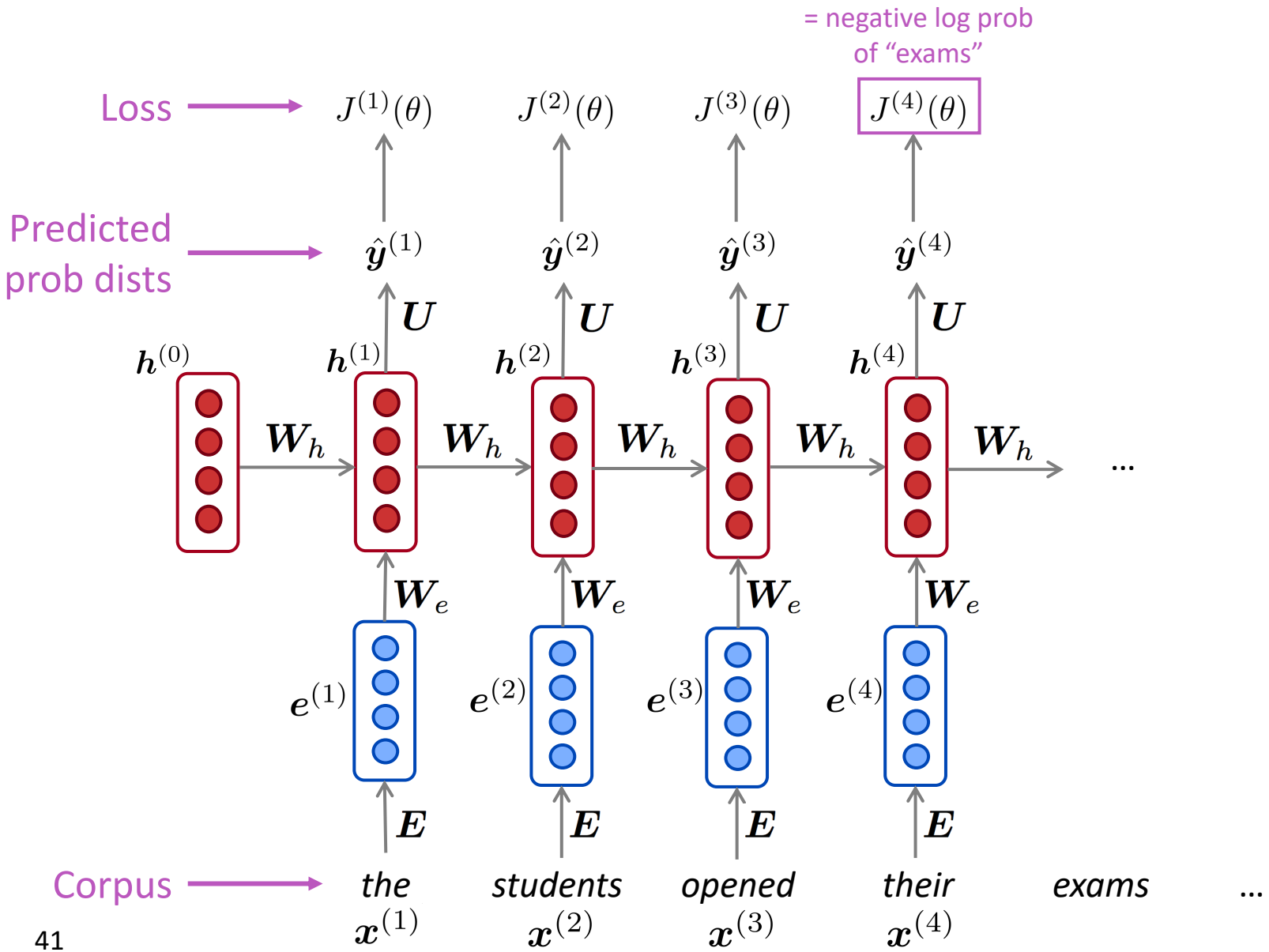
# Training an RNN Language Model



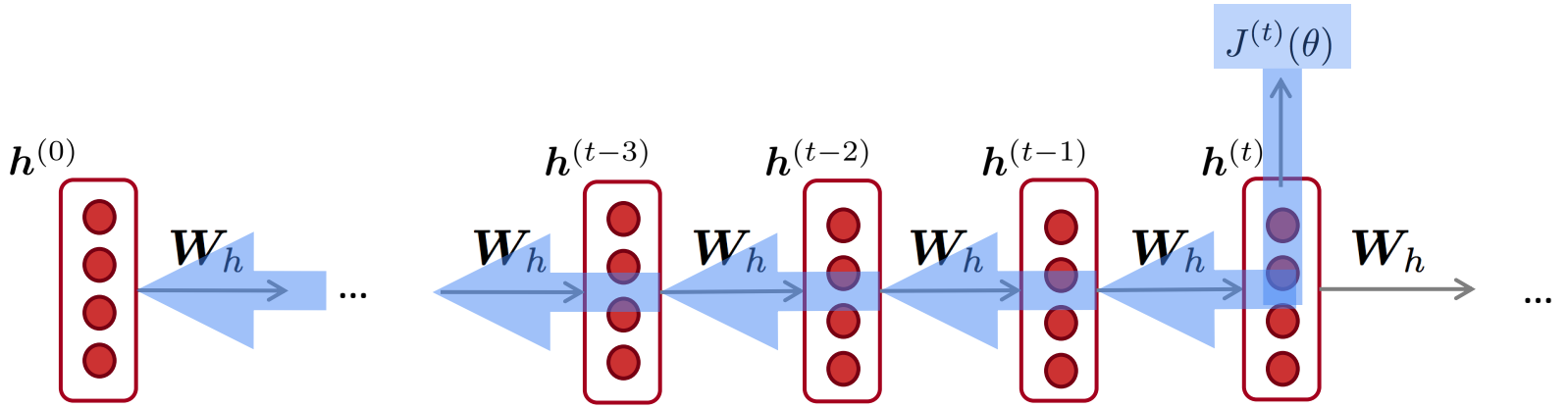
# Training an RNN Language Model



# Training an RNN Language Model



# Backpropagation for RNNs




$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} \right|_{(i)}$$

**Question:** How do we calculate this?

**Answer:** Backpropagate over timesteps *i*=*t*,…,0, summing gradients as you go.
This algorithm is called **"backpropagation through time"**
[Werbos, P.G., 1988, *Neural Networks* **1**, and others]

# Méthodologies d'Apprentissage

- <u>Pourquoi les modèles de langage ont tant d'importance?</u>
  - Au-delà de leur usage déjà utile de prédiction/suggestion de texte

# Examples

- The course on Selected Topics in AI is ...

=> continue the sentence [”enlightening”, “boring”, …]

- The latest Pixar movie deserves some Oscars

=> classify sentence as [“positive”, “negative”, “neutral”]

- Did you read the latest **provocative** tweet from President **Trump**?

=> classify each word (such as provocative and Trump) in a category, such as [“noun”, “adjective”, ”proper noun”,…]

**Generation** of words or whole sentences

**Classification** of complete sentences

(Undertanding - NLU)

**Classification** of individual words

(Undertanding - NLU)

Functions, fact(oïd)s, …

# Examples

- A soccer game with multiple males playing

⇒find other documents such as “Some men are playing a sport.”

- Bienvenue à tous

⇒translate sentence as [“Welcome to everybody”]

- Who is wearing glasses?

=>reply by Man or Woman

**Retrieval** of similar expressions or documents

Understanding and Generation **Combined**

**Multimodal** tasks: vision, language and intraction combined

# Examples

- A man is drinking a beer...

=> Determine that “man” is the subject, and “beer” the object and both are linked to the verb “drinking”

- A language model (LM) is a model that is can predict future words from previous ones. LMs have a particular status in NLP. They are pervasive and very often used as the foundational models that are then fine tuned for downstream tasks.

Ex: Resnet trained on ImageNet => Bert trained on Wikipedia

Generation of **Graphs**

Language Models

# Méthodologies d'Apprentissage

- Pourquoi les modèles de langage ont tant d'importance?
  - Au-delà de leur usage déjà utile de prédiction/suggestion de texte

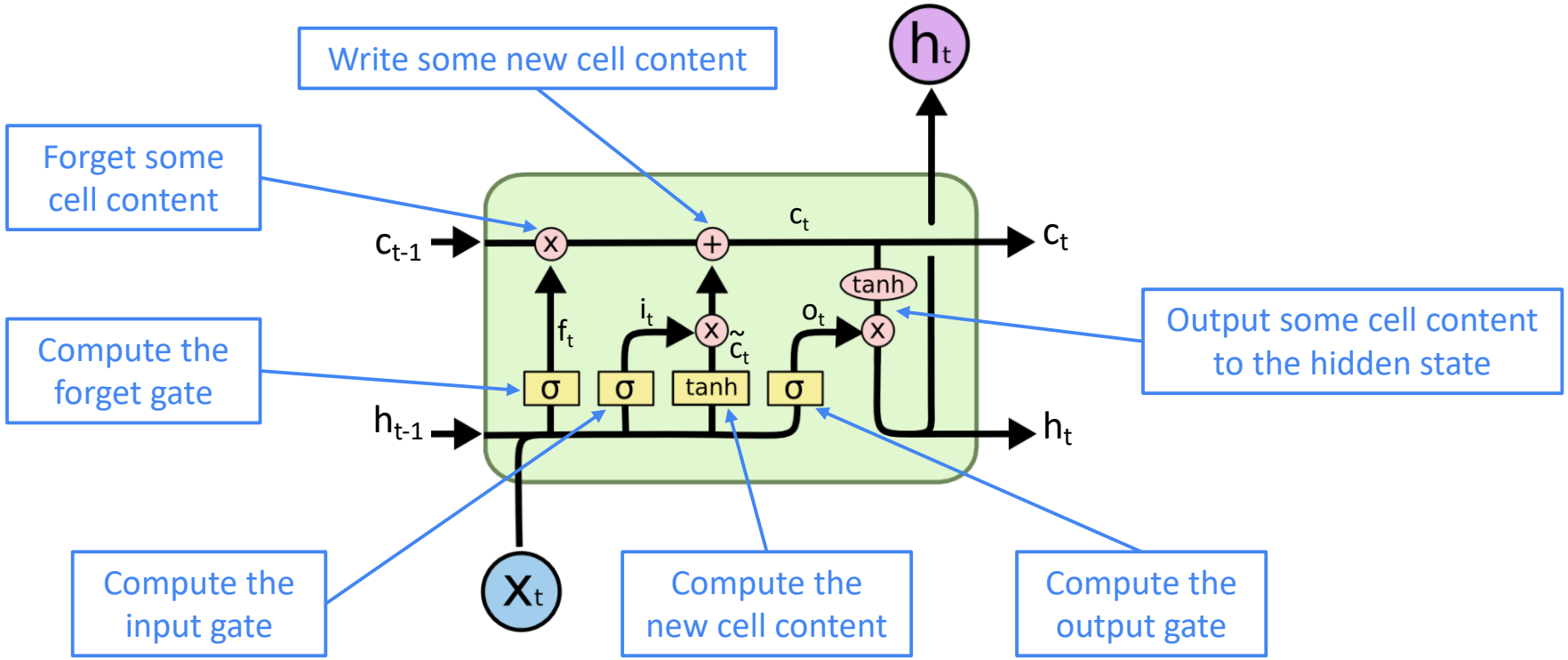
- Types d'apprentissage
  - Supervisé
  - Non-supervisé
  - (Renforcement)

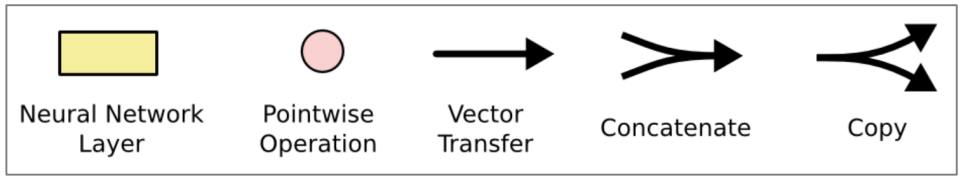
- Paradigmes
  - Distillation
  - Transfert d'apprentissage

# RNNs with Gates in NLP

# Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



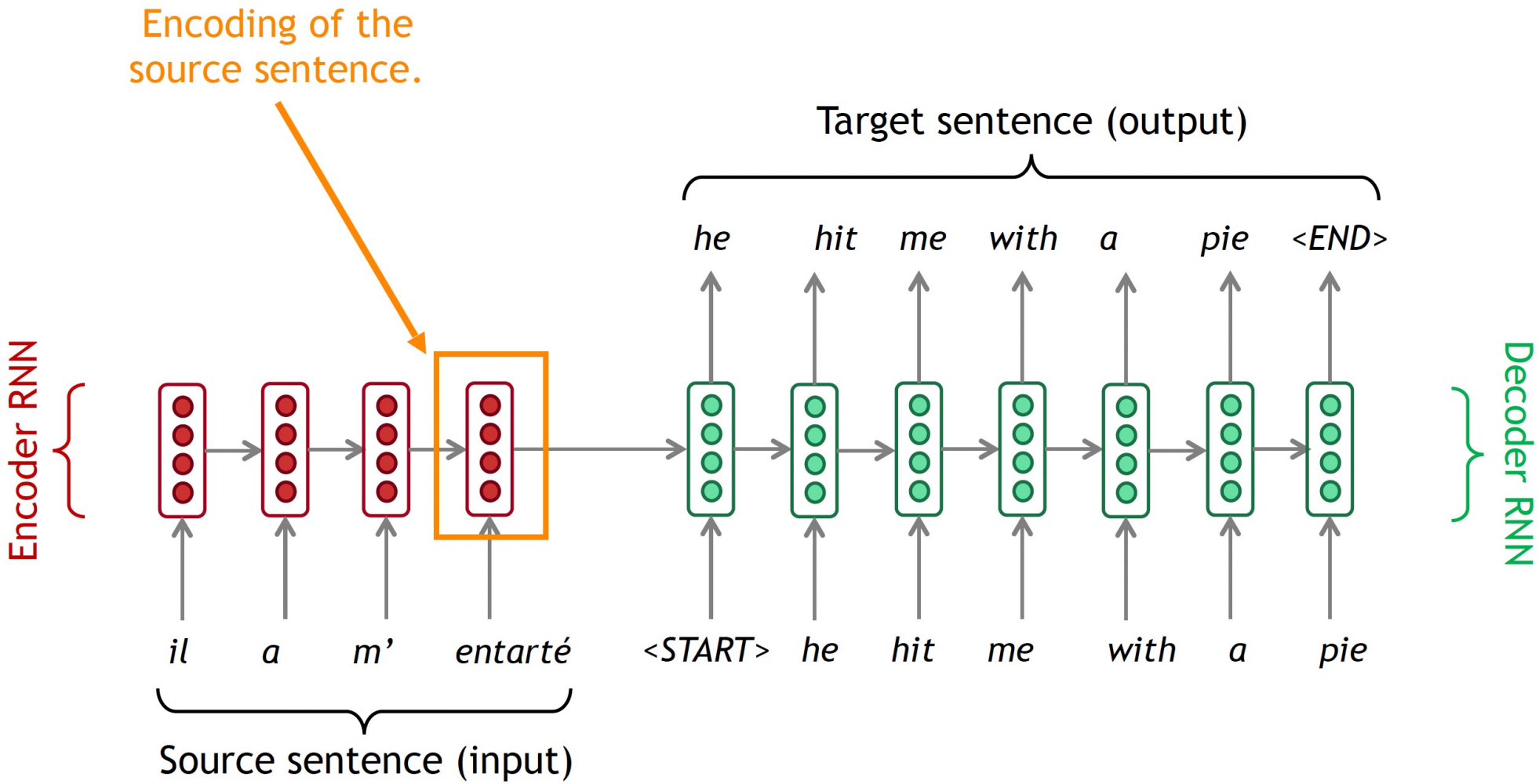






**Source:** http://colah.github.io/posts/2015-08-Understanding-LSTMs/

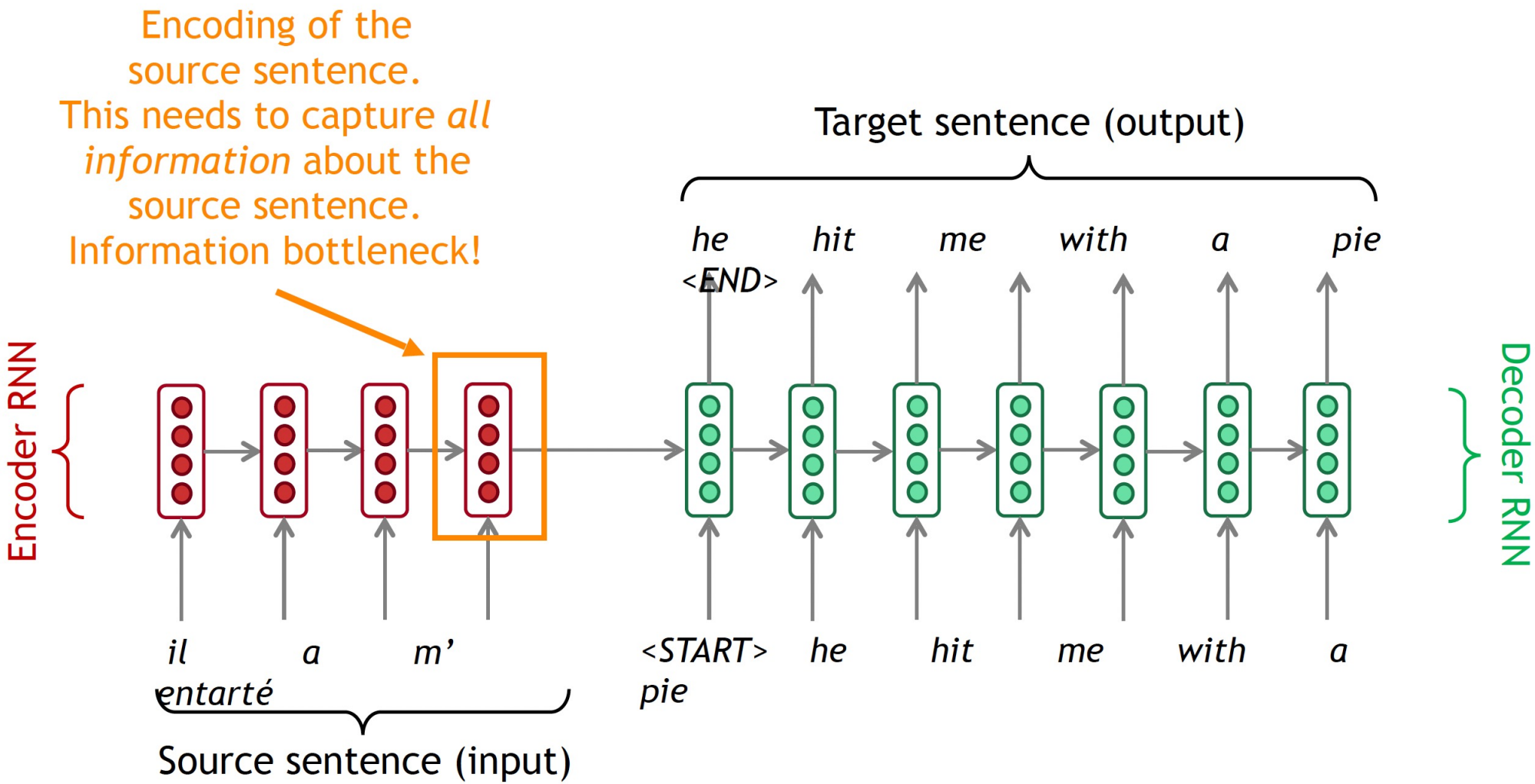
# Attention

# Sequence-to-sequence: the bottleneck problem




Problems with this architecture?

# Sequence-to-sequence: the bottleneck problem



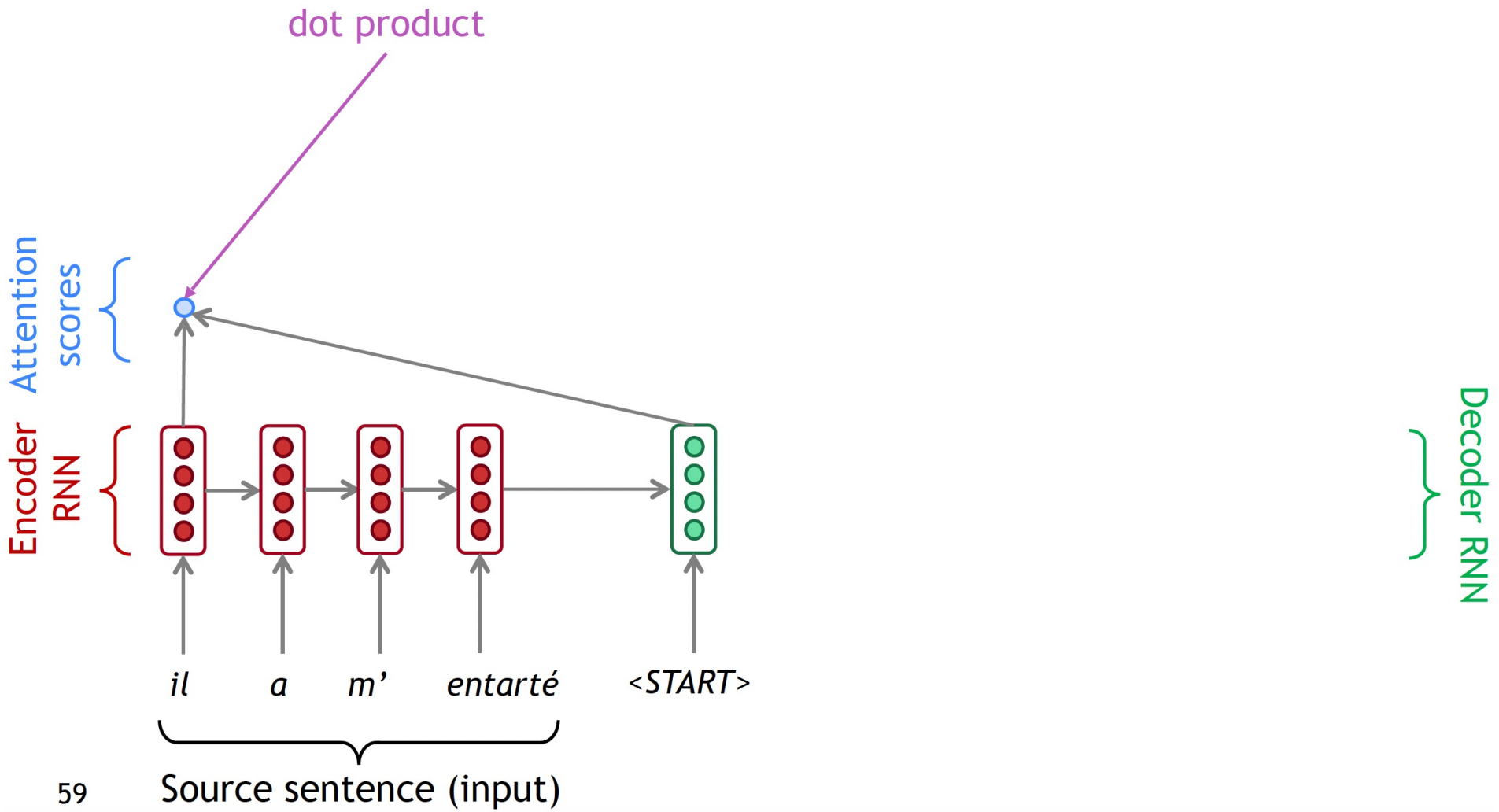
# Attention

- **Attention** provides a solution to the bottleneck problem.

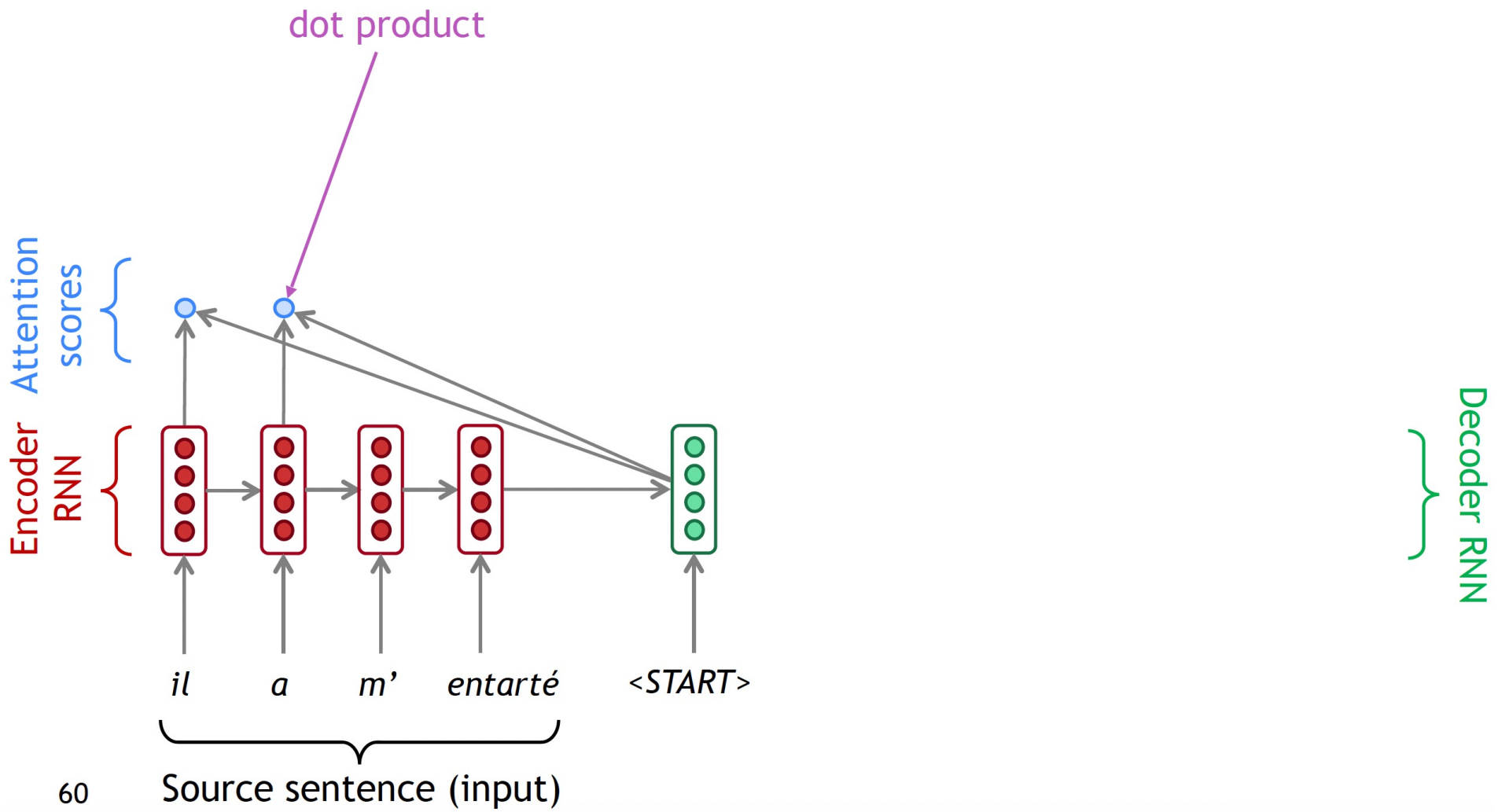
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

- First we will show via diagram (no equations), then we will show with equations

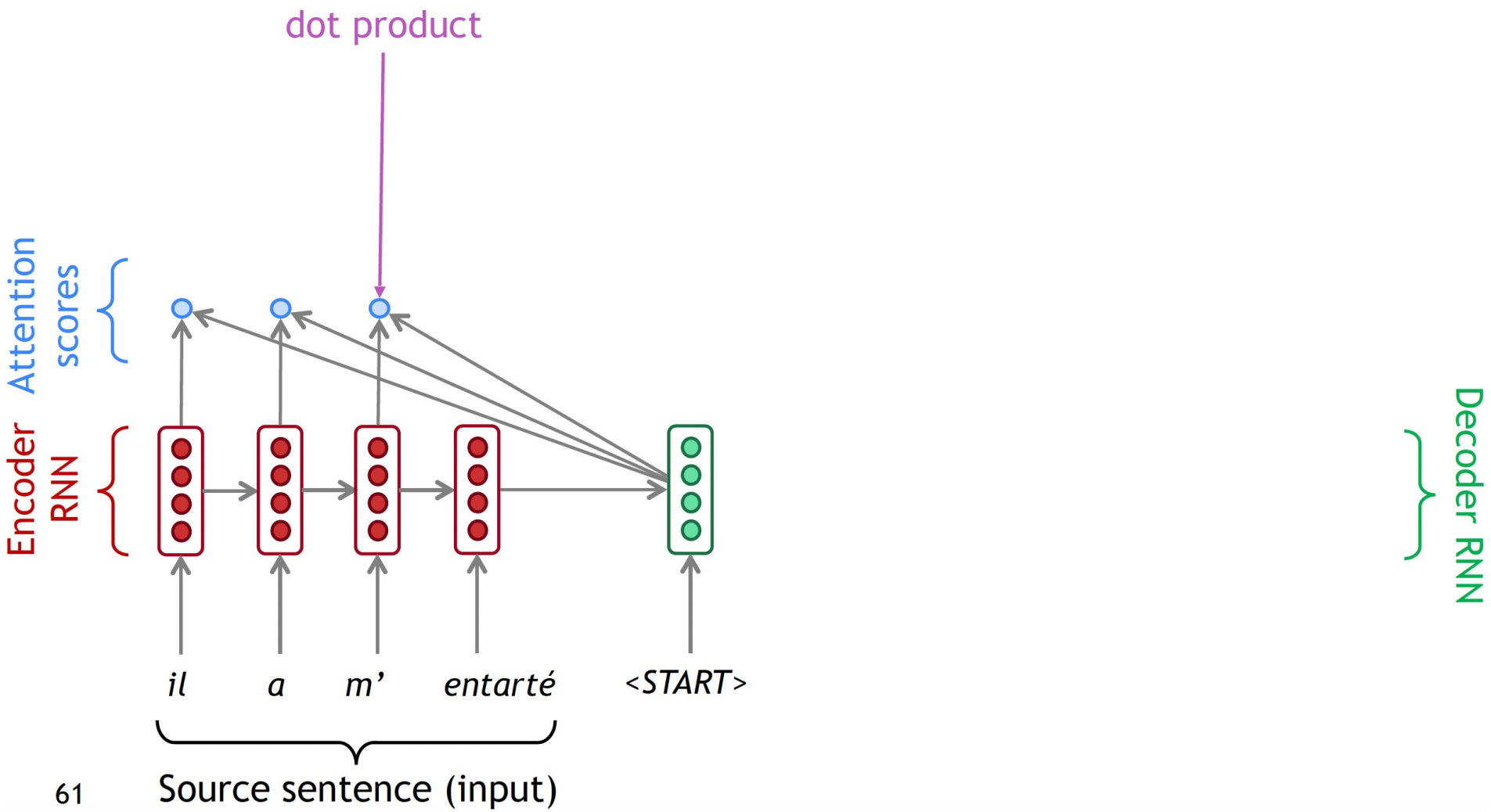
# Sequence-to-sequence with attention



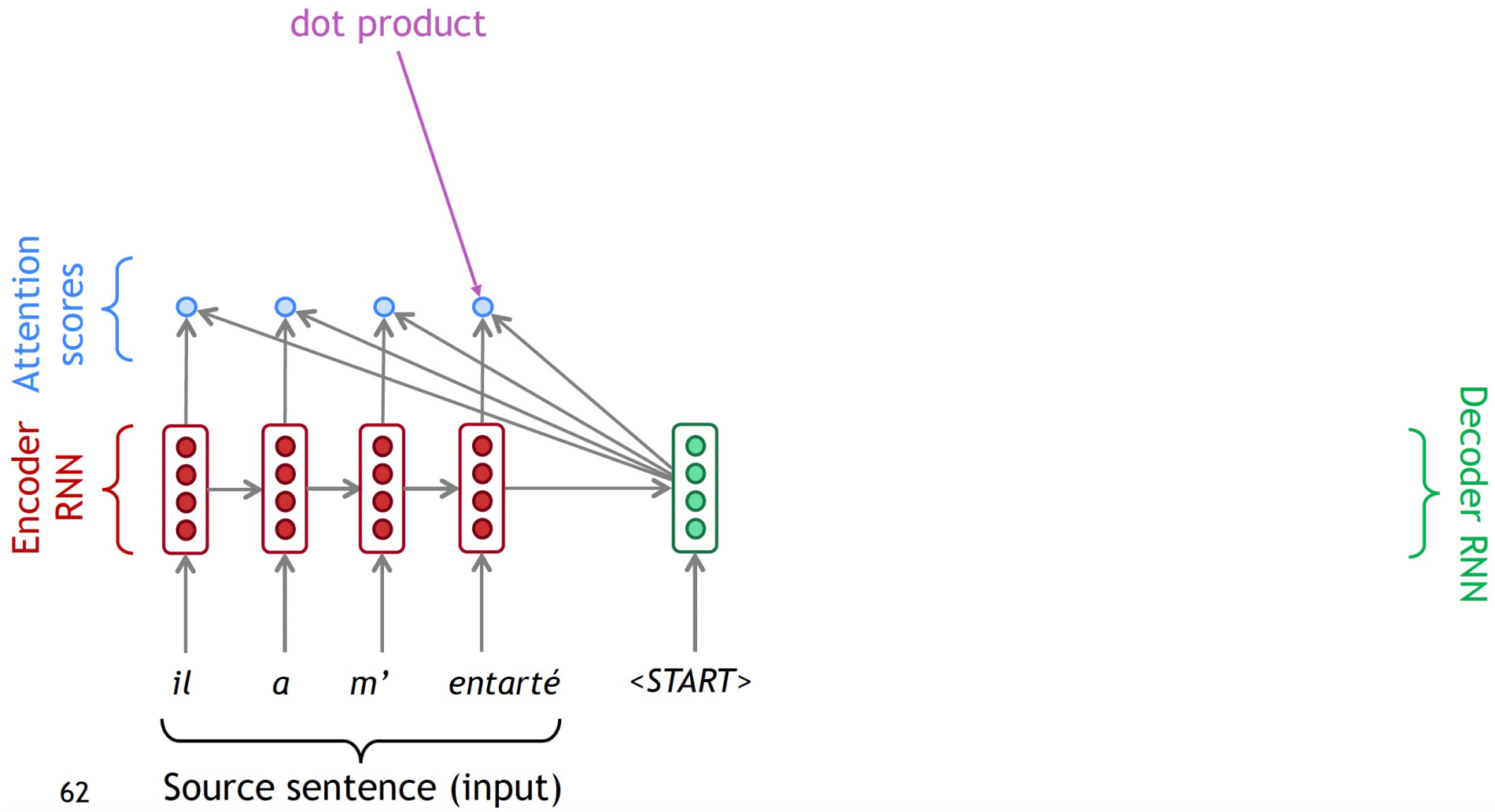
# Sequence-to-sequence with attention



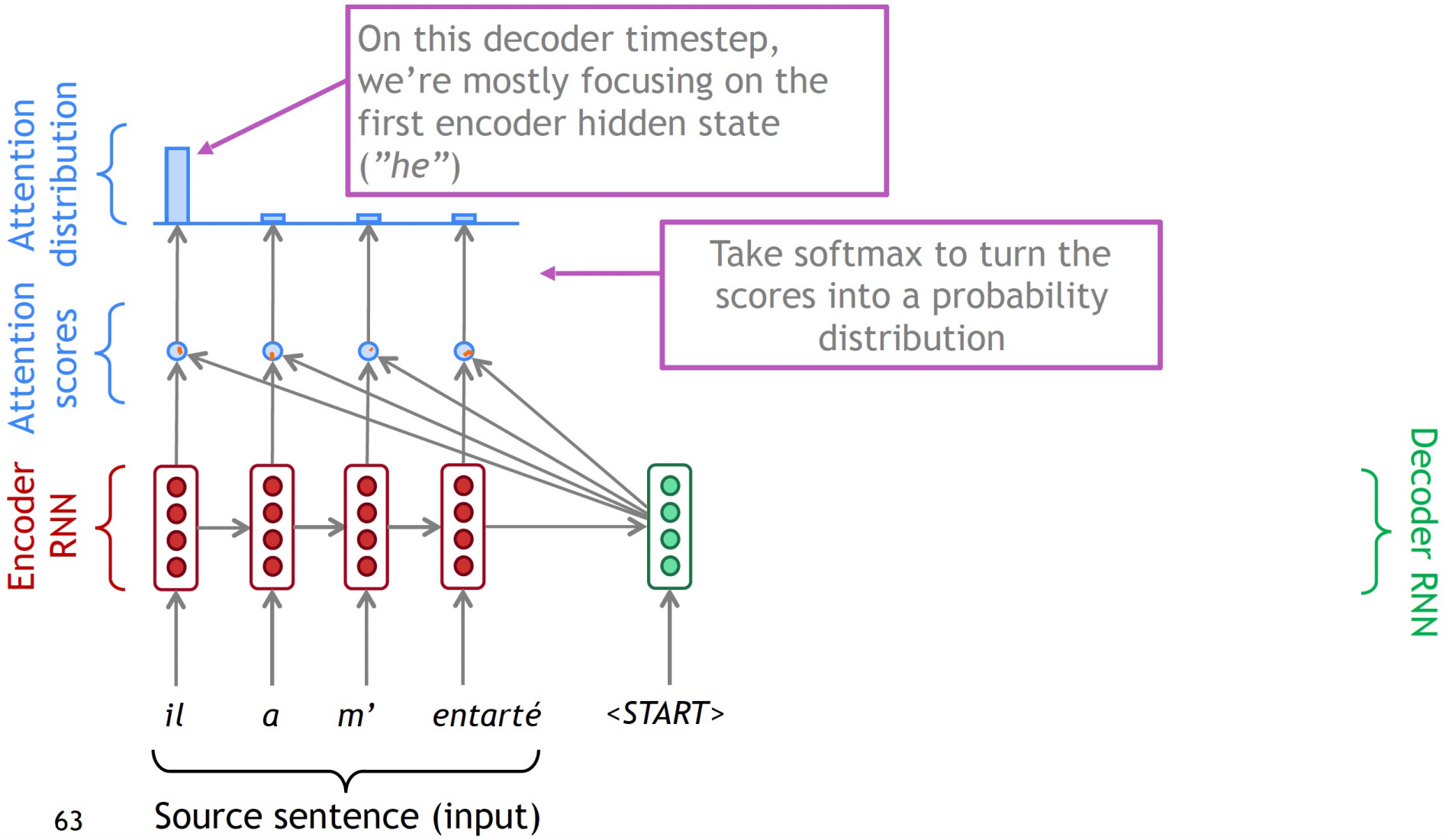
# Sequence-to-sequence with attention



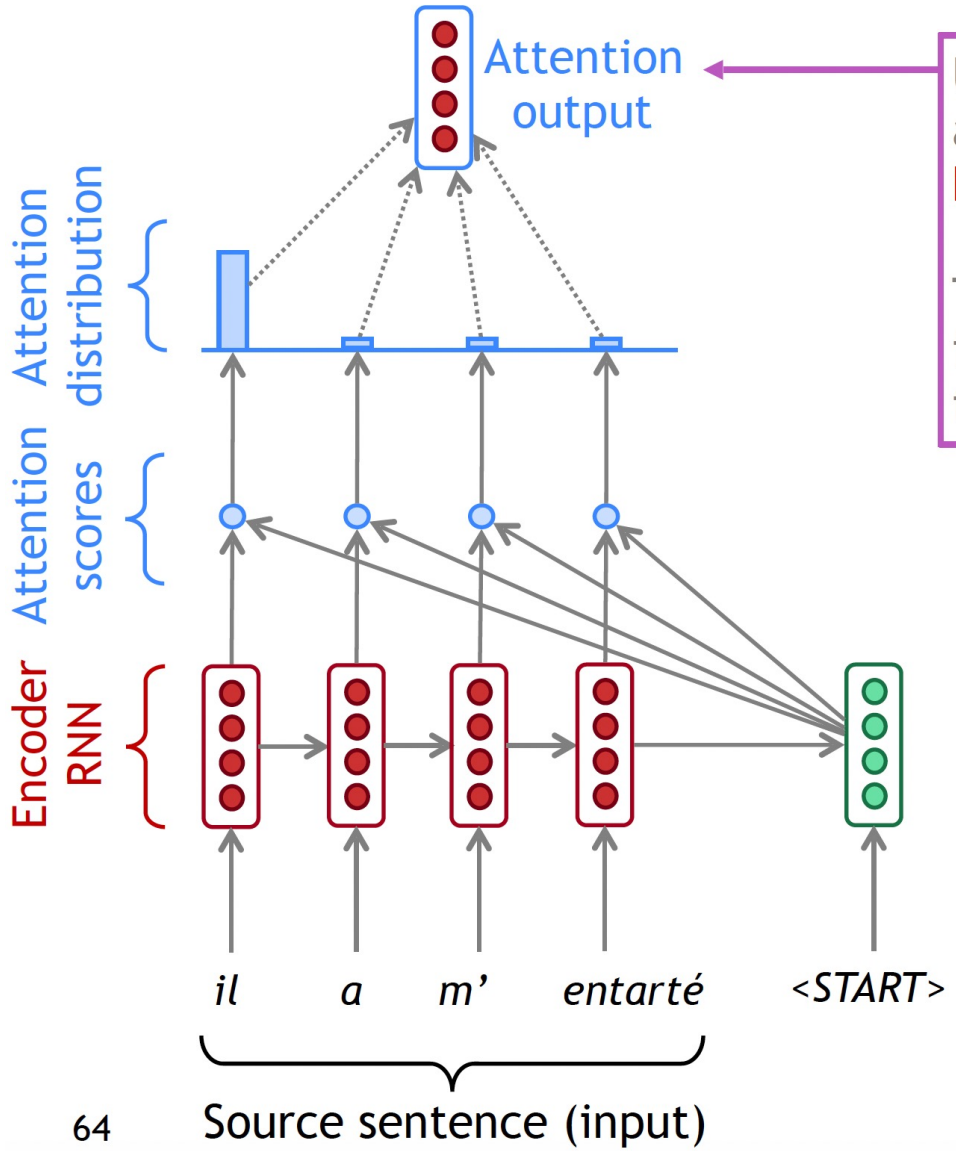
# Sequence-to-sequence with attention



# Sequence-to-sequence with attention



# Sequence-to-sequence with attention

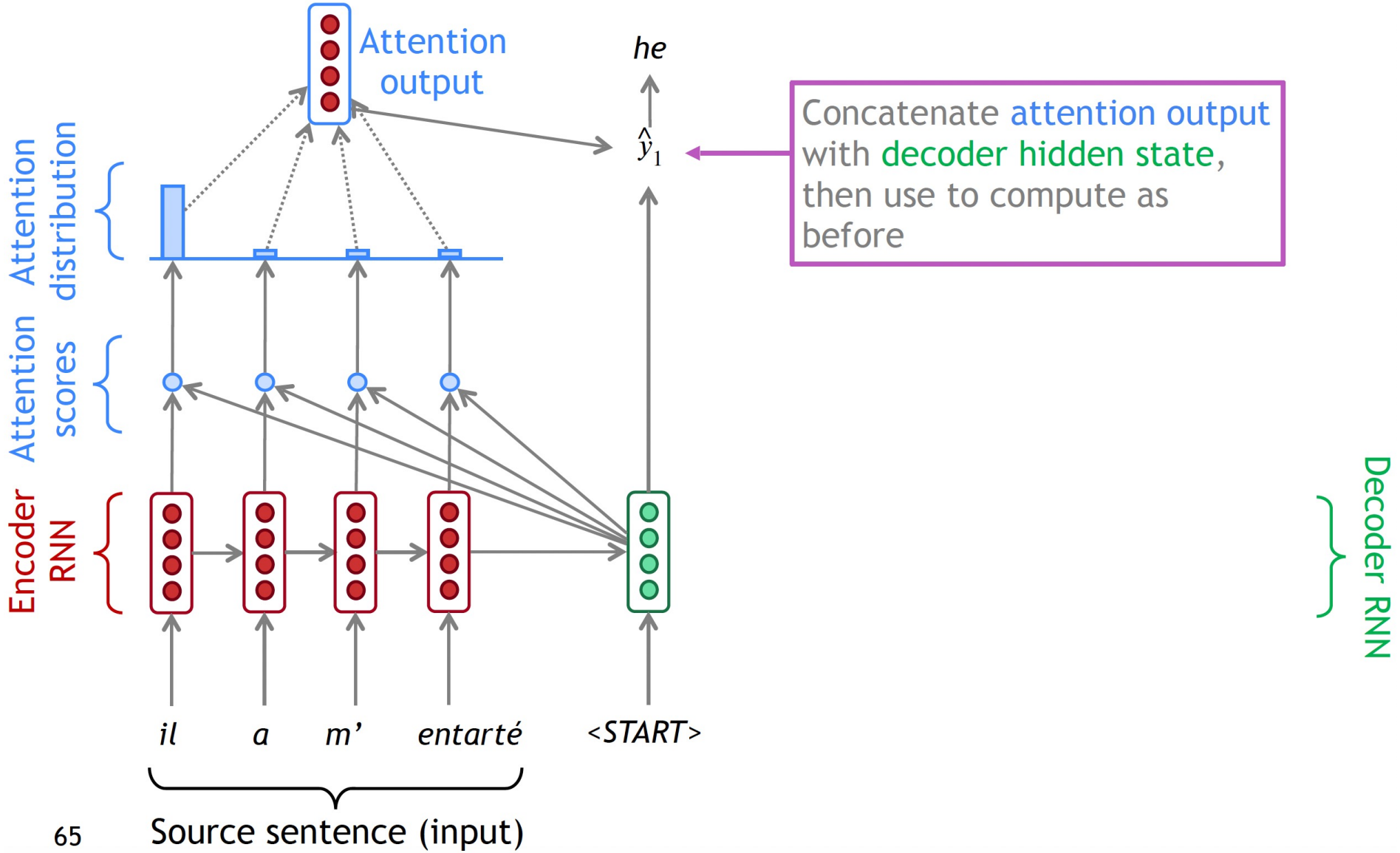



Use the attention distribution to take a **weighted sum** of the encoder hidden states.

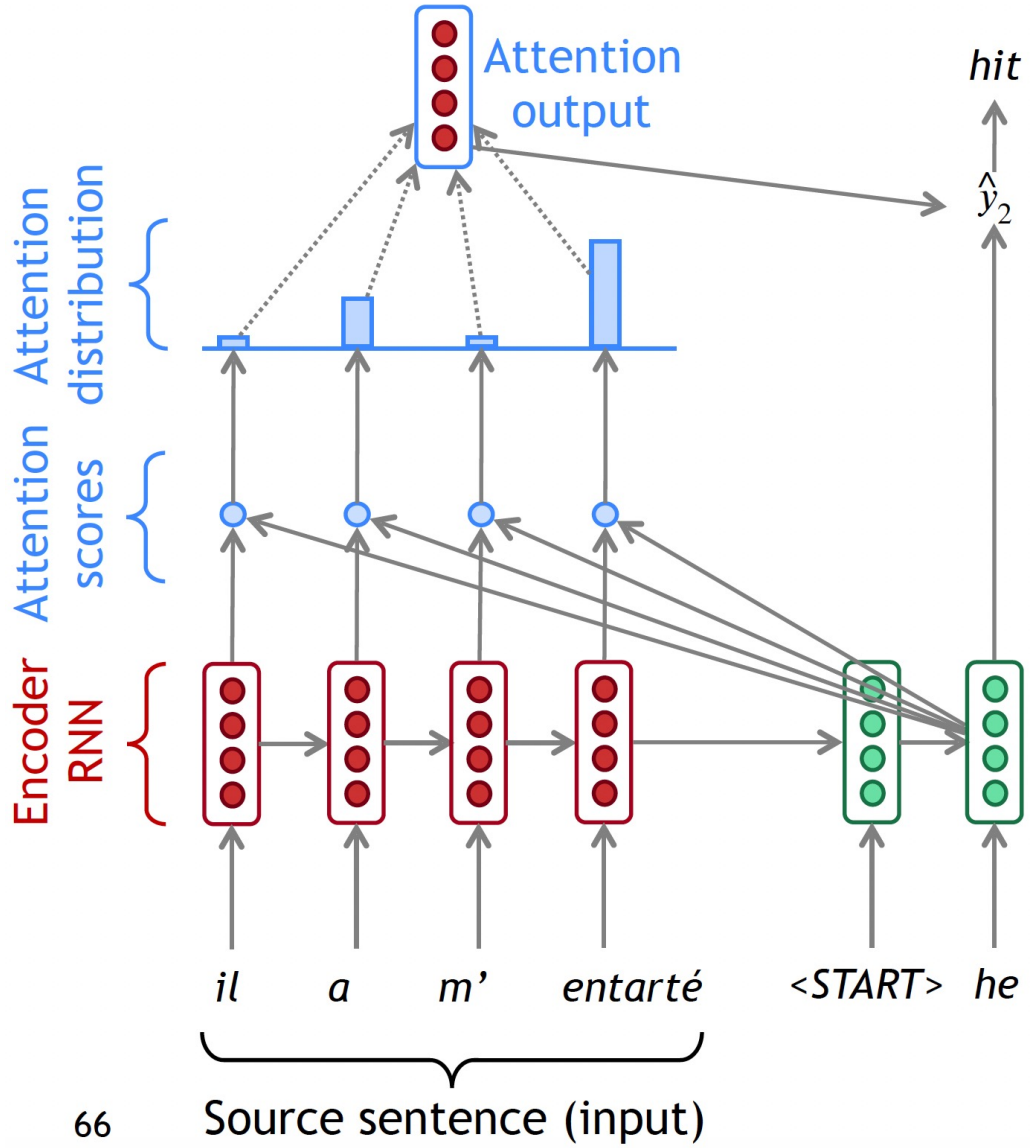
The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

# Sequence-to-sequence with attention

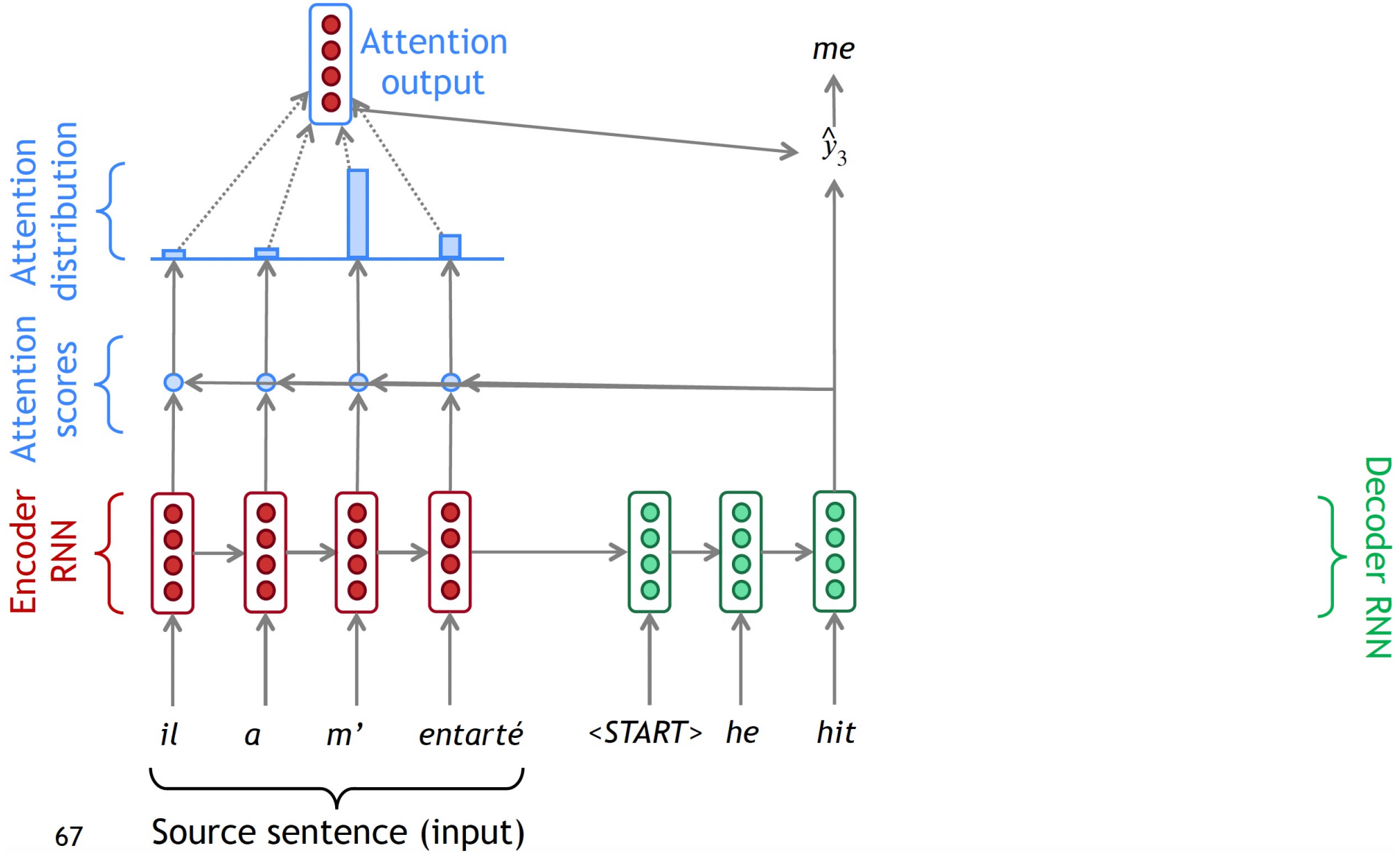


# Sequence-to-sequence with attention




Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input). We do this in Assignment 4.

# Sequence-to-sequence with attention



# Attention: in equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores $e^t$ for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution $\alpha^t$ for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use $\alpha^t$ to take a weighted sum of the encoder hidden states to get the attention output

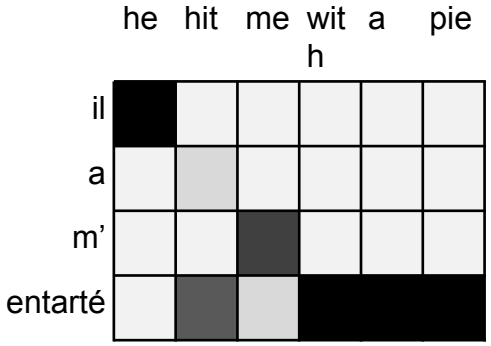
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state $s_t$ and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

# Attention is great

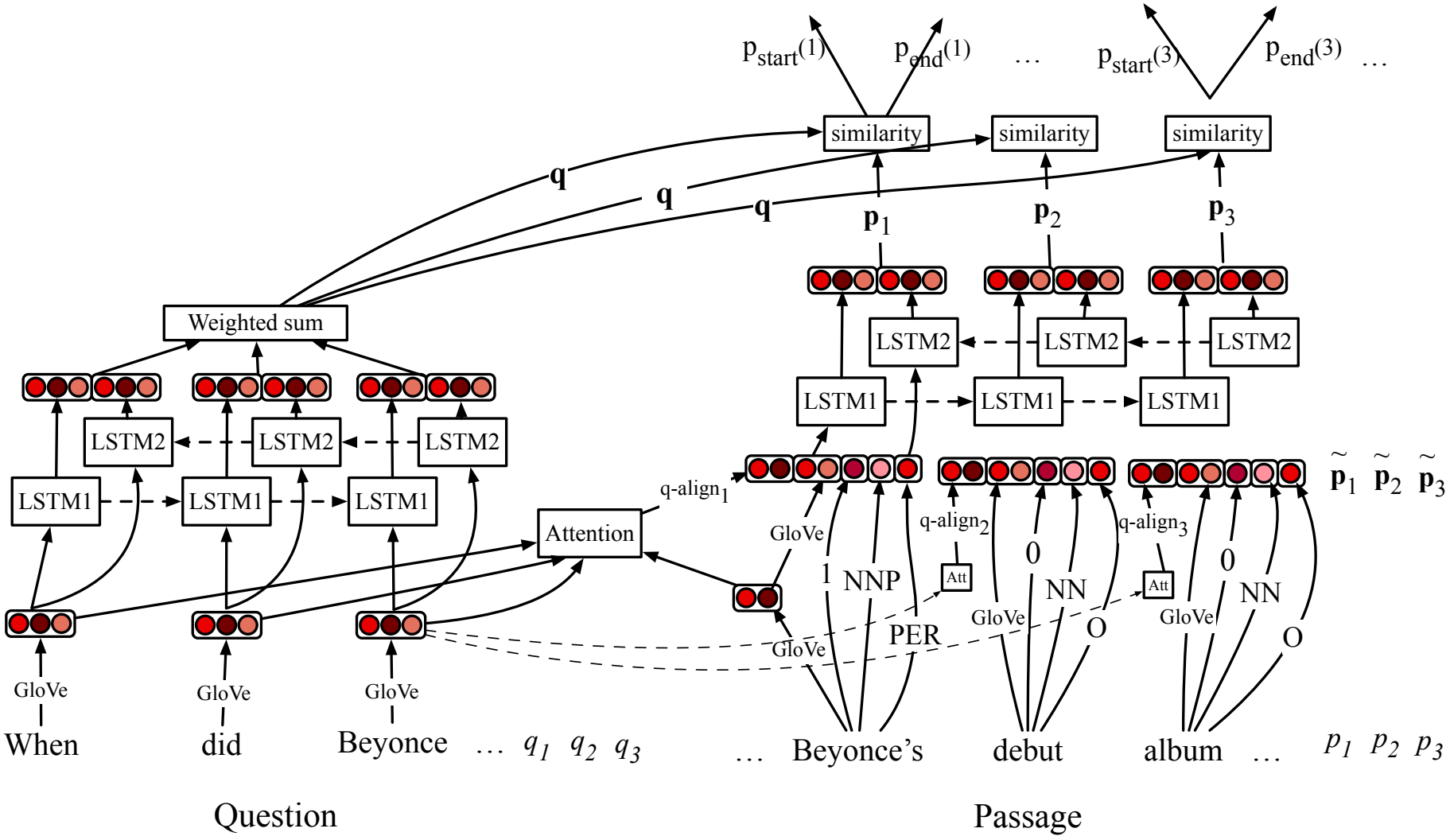
- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself






**Learnable attention then became a general architectural elemnt in Deep Learning**

# Stanford Attentive Reader++




Training objective: $\mathcal{L} = -\sum \log P^{(\text{start})}(a_{\text{start}}) - \sum \log P^{(\text{end})}(a_{\text{end}})$

# Transformers in NLP

# 7. ELMo and BERT preview

**Contextual word representations**
Using language model-like objectives

**Elmo**

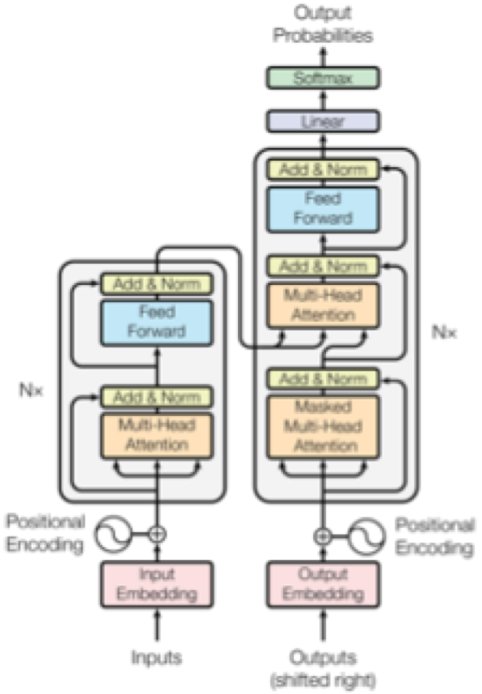
(Peters et al, 2018)

**Bert**

(Devlin et al, 2018)

Look at SDNet as an example of how to use BERT as submodule: https://arxiv.org/abs/1812.03593

The transformer architecture used in BERT is sort of attention on steroids. More later!




(Vaswani et al, 2017)

# Métriques d’évaluation d’un modèle de langage

# Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left( \frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left( \frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left( \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

**Lower** perplexity is better!

# RNNs have greatly improved perplexity

*n*-gram model →

Increasingly complex RNNs ↓

| Model | Perplexity |
|---|---|
| Interpolated Kneser-Ney 5-gram (Chelba et al., 2013) | 67.6 |
| RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013) | 51.3 |
| RNN-2048 + BlackOut sampling (Ji et al., 2015) | 68.3 |
| Sparse Non-negative Matrix factorization (Shazeer et al., 2015) | 52.9 |
| LSTM-2048 (Jozefowicz et al., 2016) | 43.7 |
| 2-layer LSTM-8192 (Jozefowicz et al., 2016) | 30 |
| **Ours small** (LSTM-2048) | 43.9 |
| **Ours large** (2-layer LSTM-2048) | 39.8 |

↓ Perplexity improves (lower is better)

**Source:** https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

# Métriques d’évaluation applicatives

# How do we evaluate Machine Translation?

**BLEU** (**B**ilingual **E**valuation **U**nderstudy)

You'll see BLEU in detail in Assignment 4!

- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - *n*-gram precision (usually for 1, 2, 3 and 4-grams)
  - Plus a penalty for too-short system translations

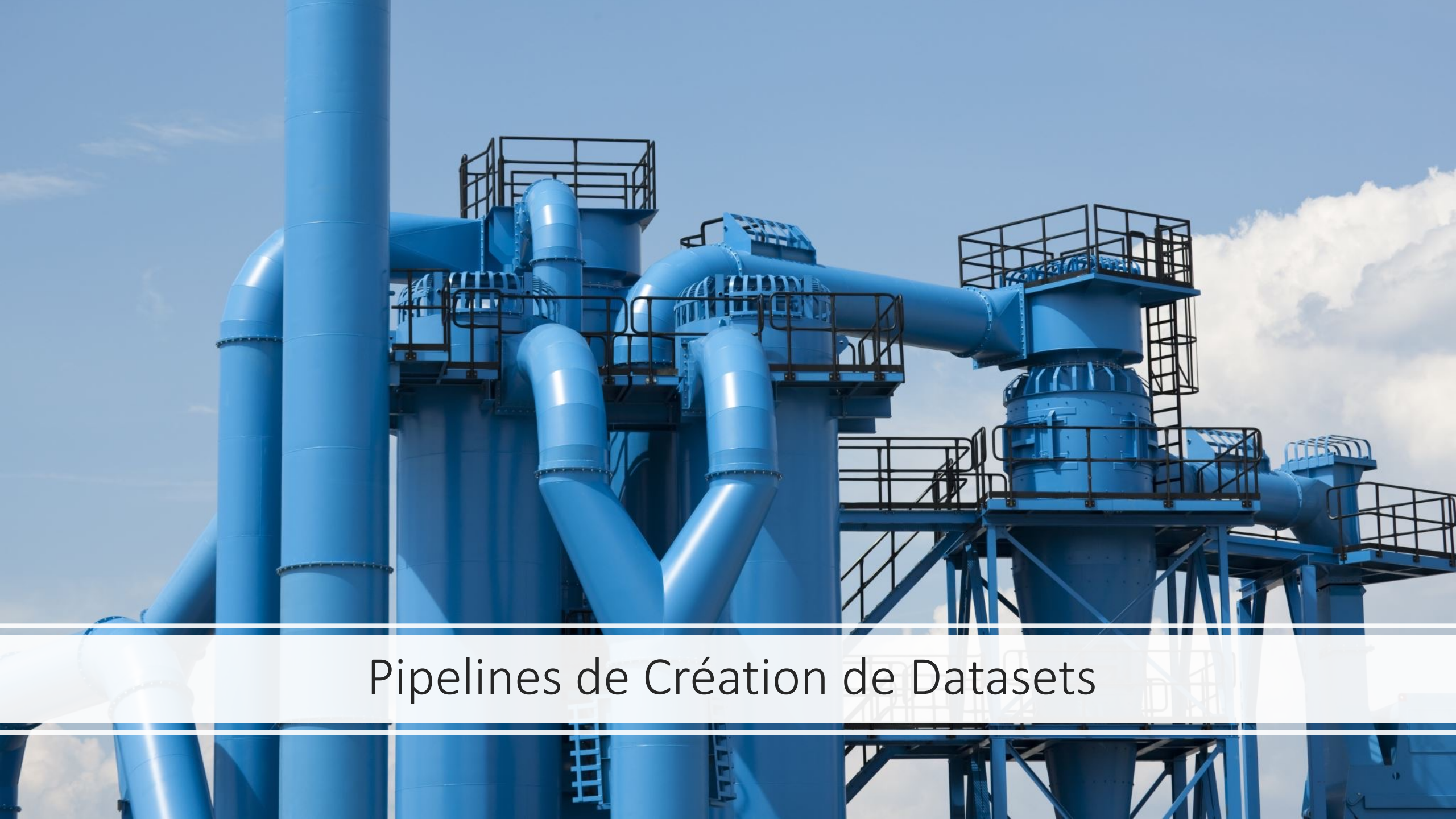
Métriques:
ROUGE
METEOR

Souvent nécessaire:
Evaluation par des gens

- BLEU is useful but imperfect
  - There are many valid ways to translate a sentence
  - So a good translation can get a poor BLEU score because it has low *n*-gram overlap with the human translation ☹



Source: "BLEU: a Method for Automatic Evaluation of Machine Translation", Papineni et al, 2002. http://aclweb.org/anthology/P02-1040



# Pipelines de Création de Datasets

# Création de datasets

- La philosophie est un peu la même que pour la vision par ordinateur:
  - Utiliser datasets existants
  - Construire des datasets, notamment par crawling/scrapping web
    - Epochs, iteration, batch size, …
  - Annoter des données
  - Augmenter les datasets

# Datasets (existants)

- Via sklearn

  https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

```
>>> from sklearn.datasets import fetch_20newsgroups >>>
twenty_train = fetch_20newsgroups(subset='train', ...
categories=categories, shuffle=True, random_state=42)
```

- Via huggingface datasets

  https://huggingface.co/docs/datasets/loading_datasets.html

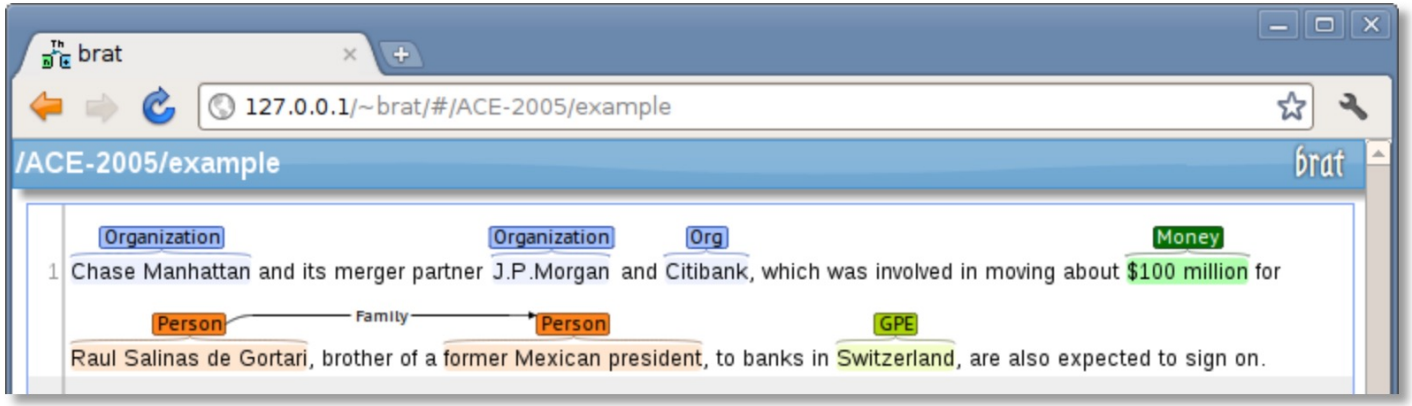
```
>>> from datasets import load_dataset >>> dataset =
load_dataset('squad', split='train')
```

# Data scrapping/crawling

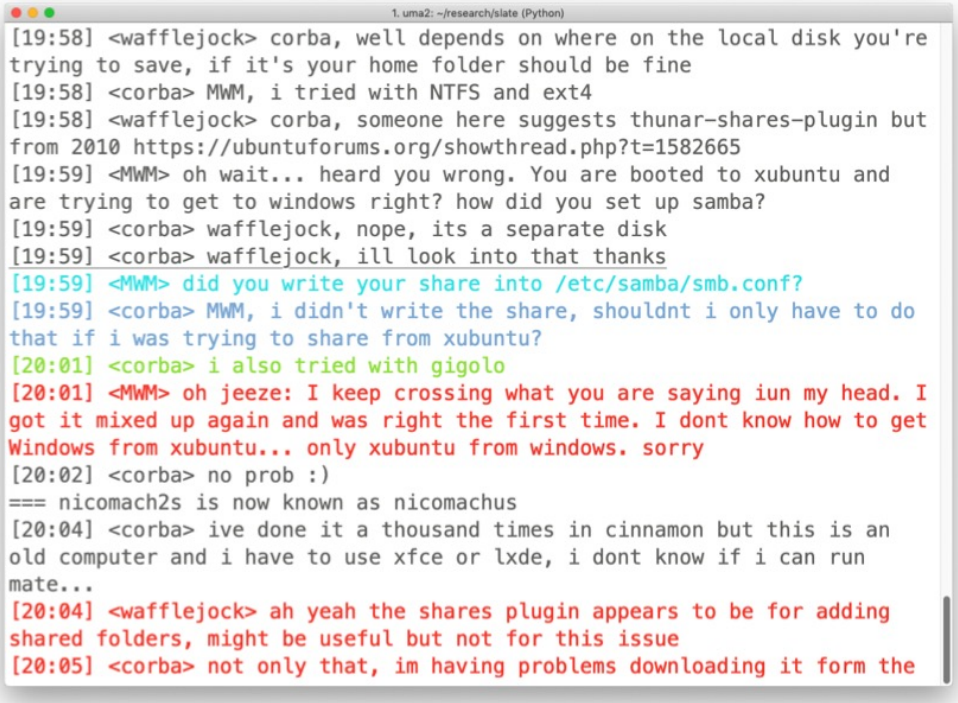
- Outils à utiliser
  - https://scrapy.org
  - https://github.com/gbolmier/newspaper-crawler
  - https://github.com/jeugregg/FakeNewsDetectionFr

# Data annotation

- Outil à utiliser
  - http://jkk.name/slate/
  - https://aclanthology.org/P19-3002.pdf
  - brat



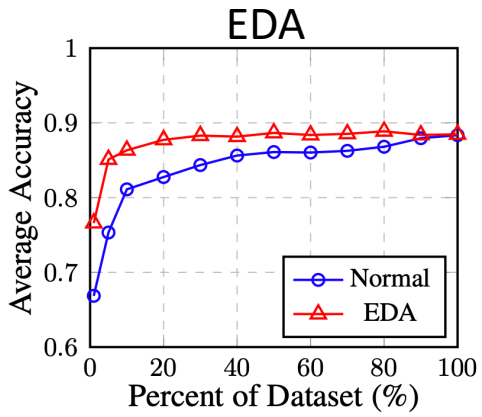

example annotations (following in part the ACE 2005 entity and relation annotation guidelines)

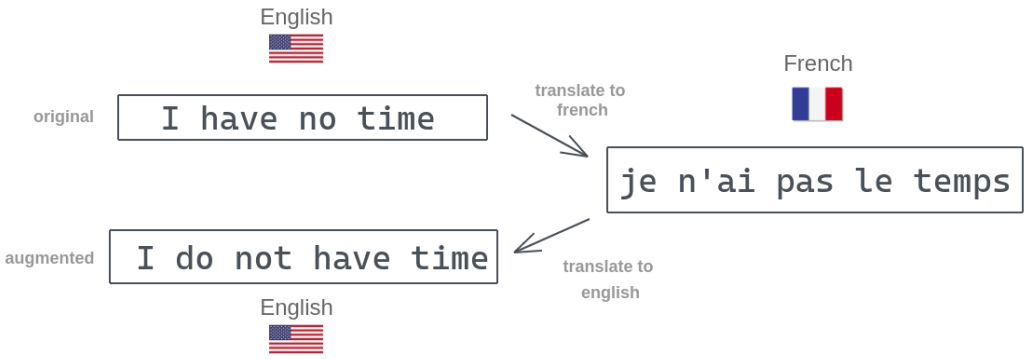



(b) Adjudication of a linking task annotating reply-to relations on IRC. The current message in green, an antecedent that is agreed on is dark blue, one that is not agreed on is light blue, other messages with disagreements are red, and the underline indicates a message that is being considered for linking to the green message (incorrectly in this case).

# Data augmentation

- Tools
  - EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks - https://github.com/jasonwei20/eda_nlp
  - Albumentations: Fast and Flexible Image Augmentations - https://github.com/albumentations-team/albumentation
  - NLPAug: NLP Augmentations - https://github.com/makcedward/nlpaug
  - Translation systems => back translation



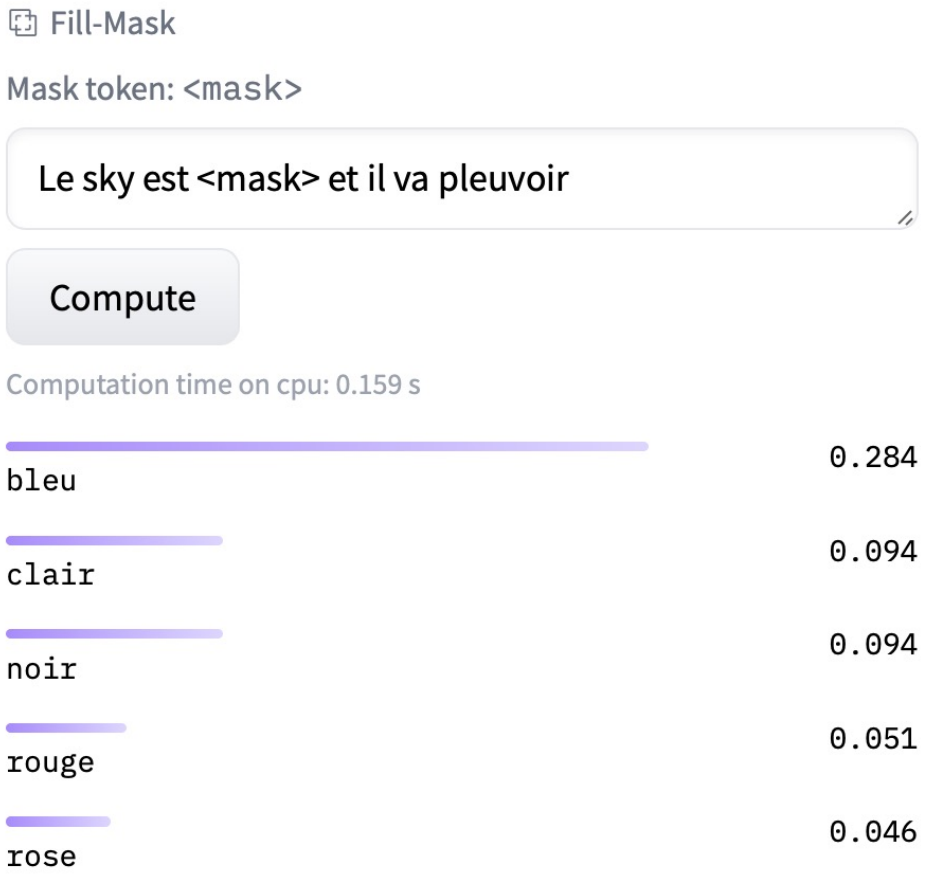

- Tuto:
  - https://neptune.ai/blog/data-augmentation-nlp

# Data augmentation

- Techniques dans le notebook
  - Thésaurus & dictionnaires de synonymes
  - Similarité de mots dans un espace Word2Vec
  - Modèle de langage pour remplacer un mot

# Multilinguisme

- https://huggingface.co/xlm-roberta-base?text=Le+sky+est+%3Cmask%3E+et+il+va+pleuvoir



# 2020’ – Retour à la déraison

- 2021 - DALL-E (OpenAI):
  - Logiciel génrant des images sur base d’une description texte - domaine représenté très large.

TEXT PROMPT an armchair in the shape of an avocado. . . .

AI-GENERATED IMAGES

Edit prompt or view more images↓

# Ressources pour aller plus loin

https://web.stanford.edu/~jurafsky/

https://nlp.stanford.edu

https://nlp.stanford.edu/manning/

# Let’s get practical

# Classification tasks – sentence or document level

- The latest Pixar movie deserves some Oscars

$\Rightarrow$ classify sentence as [“positive”, “negative”, “neutral”]

**Classification** of complete sentences

(Undertanding - NLU)

- Tâche de classification de texte (phrase ou document)
  - Identifiation de la langue
  - Classification thématique
  - Détection de harcèlement et incitations à la haine
  - Analyse de sentiment (et de polarité d’opinion)
  - Détection de “fake news”
  - Détection de fraude
  - Triage d’appels ou de déclaration de sinistres
  - …

# scikit-learn

*Machine Learning in Python*

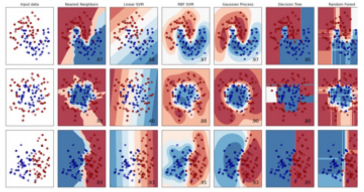
Getting Started | Release Highlights for 1.0 | GitHub

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

## Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.
**Algorithms:** SVM, nearest neighbors, random forest, and more...

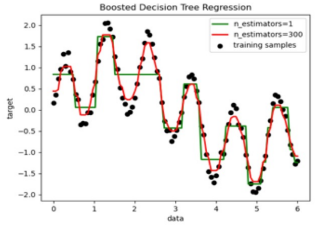


Examples

## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.
**Algorithms:** SVR, nearest neighbors, random forest, and more...

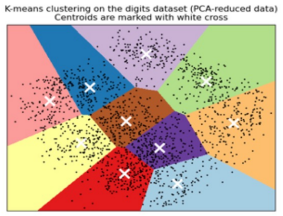



Examples

## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes
**Algorithms:** k-Means, spectral clustering, mean-shift, and more...




Examples

v4.12.5

transformers

Star 54,617

Search docs

GET STARTED
- Quick tour
- Installation
- Philosophy
- Glossary

USING 🤗 TRANSFORMERS
- Summary of the tasks
- Summary of the models

SIGN IN | MODELS | FORUM

Docs » Transformers | View page source

# Transformers

Séparer low-level et high)level

State-of-the-art Natural Language Processing for Jax, Pytorch and TensorFlow

🤗 Transformers (formerly known as *pytorch-transformers* and *pytorch-pretrained-bert*) provides general-purpose architectures (BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet...) for Natural Language Understanding (NLU) and Natural Language Generation (NLG) with over 32+ pretrained models in 100+ languages and deep interoperability between Jax, PyTorch and TensorFlow.

This is the documentation of our repository transformers. You can also follow our online course that teaches how to use this library, as well as the other libraries developed by Hugging Face and the Hub.

**If you are looking for custom support from the Hugging Face team**

GENSIM
topic modelling for humans

Home | Documentation | Support | API | About | Donate

Gensim is a FREE Python library

# Topic modelling for humans

- ✔ Train large-scale semantic NLP models
- ✔ Represent text as semantic vectors
- ✔ Find semantically related documents

**Out now:** spaCy v3.2

USAGE | MODELS | API | UNIVERSE | 21,812

# Industrial-Strengt Natural Languag Processing

IN PYTHON

# Rappel- Taille des modèles

- Petits modèles (sm) et gros modèles (lg)
  - Utilisez les petits pour commencer, afin de réduire les temps de download et de calacul,
  - mais ne vous étonnez donc pas si cela ne casse rien en termes de précision des résultats d’analyse du langage.

# Objectifs des exercices du jour

- Continuer à s'approprier les outils:
  - Chargement de bases de données existantes
  - Entainer des classificateurs de texte
    - Du plus simple et le moins contextuel: Tf-idf + MLP (via sklearn)
    - … au plus puissant: Transformer (via transformer)
  - Exploiter un modèle de langage.
  - Tenter l'augmentation de données pour améliorer vos classificateurs.

Au travail