Deep Text

Michel.RIVEILL@univ-cotedazur.fr

2023-2024

Sommaire

- Travailler avec un texte
- Principales taches
  - Associer un label à un ensemble de mots
  - Associer un label à chacun des mots
  - Prédire le prochain mot
    - Traduction
    - Chat bot
    - Résumé un texte
    - Générer du texte
- Principales étapes
  - Associer un vecteur à un mot (représentation d'un mot)
    - Embedding d'un mot
  - Rechercher des corrélations dans le temps
    - La couche récurrente
  - Représentation d'une phrase

Représenter un mot par un vecteur Embedding

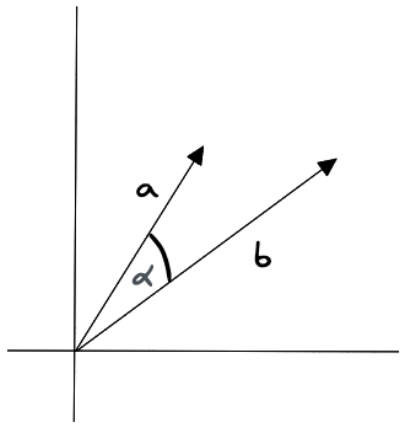
Solution naive

- Exemple
  - Phrase1 = Le chat s'assoit sur le paillasson
  - Phrase2 = Un félin se repose sur le tapis

- Représentation des mots par One Hot Encoding
  - Un chat est un félin → $sim('chat','felin') \approx 1$
    - Avec OHE → $sim(mot1, mot2) = 0, si\ mot1 \neq mot2$

- Représentation des phrases par Bag of Word
  - Avec BOW → $sim(phrase1, phrase2) = 0,\ si\ mots\ d$ $mots2\ de\ la\ phrase\ 2$

- Rappel : $sim(a,b) = \frac{a.b}{\|a\|\|b\|}$



Solution naive (con't)

- Exemple
  - Phrase1 = Le chat s'assoit sur le paillasson
  - Phrase2 = Un félin se repose sur le tapis
- Vocabulaire

Le	Chat	Se	Assoit	Sur	Le	Paillas-son	Un	Félin	Repose	Tapis

- OHE (représentation des mots)

0	I	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	I	0	0

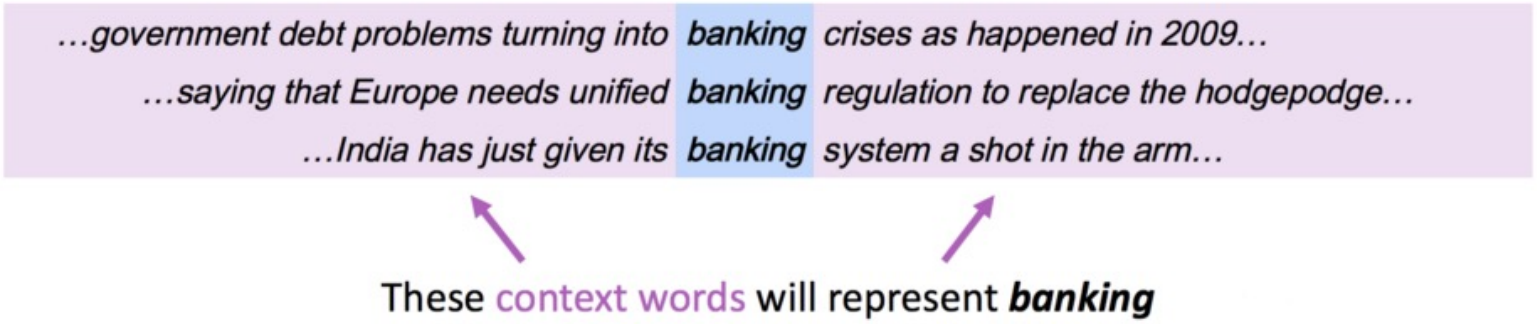
- BOW (représentation des phrases

I	I	I	I	I	I	I	0	0	0	0
0	0	I	0	I	I	0	I	I	I	I

  - $sim(phrase1, phrase2) = \frac{3}{\sqrt{7}.\sqrt{7}} = \frac{3}{7}$ mais dépend de 'se', 'sur', 'le'

Idées de base

- Associer un vecteur représentant le sens du mot à la forme syntaxique d'un mot
  - Le sens du mot dépend de son contexte d'utilisation



  - Importante évolution dans le domaine du traitement de la langue
  - Avec cette approche : la même forme syntaxique à toujours le même sens
    - Les mots ont bien souvent plusieurs senses

Comment construire ce vecteur

- C'est une longue histoire
  - Latent Semantic Analysis/Indexing (1988)
    - Basé sur la construction d'une matrice de co-occurrence
    - Cette matrice est ensuite réduite (NMF, SVD, etc.)
  - Word2Vec (2013)
    - Basé sur la prédiction du mot manquant dans une suite de mot
  - Glove (2014)
    - Un mixte des deux précédents

  - Il existe pour ces approches des modèles pré-entrainé
    - Gensim : bibliothèque pour pré-entrainé ses propres embedding
    - radimrehurek.com/gensim/models/keyedvectors.html

- On peut aussi utiliser une autre solution,
  - Une couche d'embedding de Keras Embedding
    - Disponible en Keras, Tensorflow ou Pytorch

Travail préparatoire

- Découper l'ensemble du texte en token (~ word)
  - Tokenization – dépend de la langue
- Obtenir tout le vocabulaire
  - Uniquement sur la partie 'train'
- Convertir une phrase en une liste d'entier
  - Par exemple en utilisant un dictionnaire

- De nombreux outils existent, par exemple :
  - keras.layers.TextVectorization
  - Principaux parametres
    - max_tokens=None, # S'il est fixé sont retenus les mots les plus fréquents
    - standardize="lower_and_strip_punctuation",
    - split="whitespace",
    - output_mode="int", # Les autres modes ne nous intéressent pas
    - output_sequence_length=None, # Fixe une taille maximale
    - vocabulary=None # S'il n'est pas précisé, il faut 'ajuster' le TextVectorizer

Travail préparatoire (Exemple)

- X_train = [‘J’aime bien le deep learning’,
  ’je préfère les chouquettes’,
  ‘I am groot’]

- tv = TextVectorization(output_sequence_length=4)
- tv.adapt(X_train) # ajuste le vocabulaire
- tv.get_vocabulary()
  - ['', '[UNK]', 'préfère', 'les', 'learning', 'le', 'j’aime', 'je', 'i', 'groot', 'deep', 'chouquettes', 'bien', 'am']
  - ‘’ sera utilisé pour le ‘paddind’ (remplissage)
  - ‘[UNK]’ sera utilisé pour les mots non présent dans le vocabulaire

- X_train_enc = tv(X_train)
- print(X_train_enc)
  - array([[6, 12, 5, 10], [7, 2, 3, 11], [8, 13, 9, 0]])

Approche 1 - utilisation d'un Embedding layer non pré-entrainé

- Transforme un entier en un vecteur
- keras.layers.Embedding
- Principaux paramètres
  - input_dim : taille du vocabulaire d'entrée
  - output_dim : taille du vecteur
- Exemple
  - Embedding(14, 2)

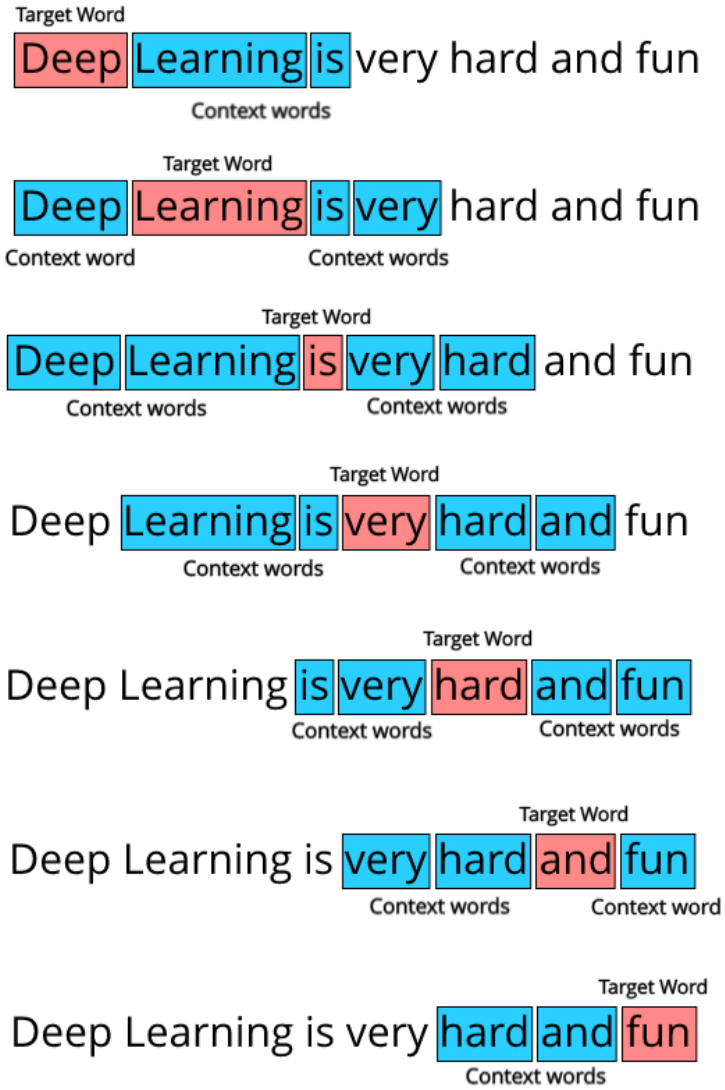
output_dim

input_dim

	E1	E2
''		
'[UNK]'		
'préfère'		
'les'		
'learning'		
'le'		
'j'aime'		
'je'		
'i'	0.5	0.4
'groot'	0.7	0.5
'deep'		
'chouquettes'		
'bien'		
'am'	0.3	0.4

Approche 2 - utilisation d'un Embedding layer pré-entrainé (Word2Vec)

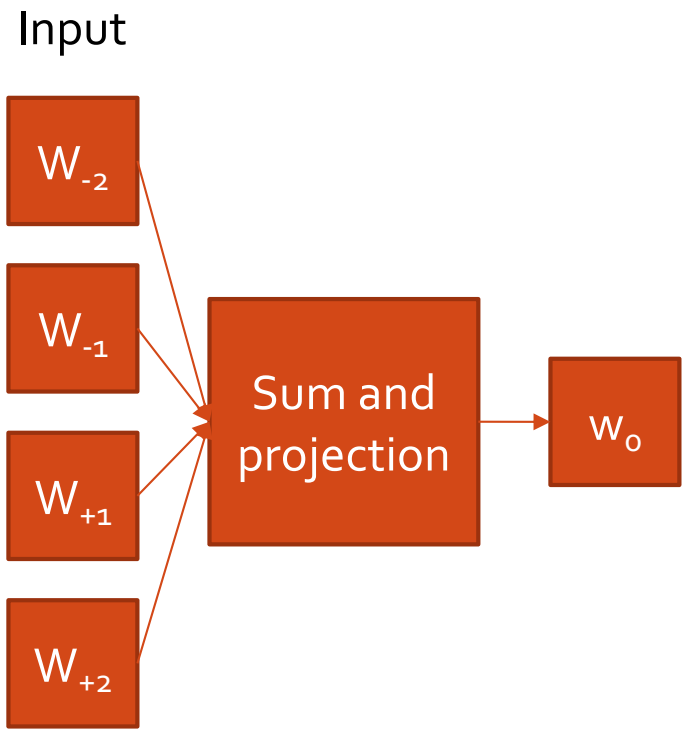
- Instead of entire documents, ***Word2Vec*** uses words ***k*** positions away from each center word.
  - These words are called **context words**.
- Example for ***k*=2**:
  - Center word: red (also called focus word).
  - Context words: blue (also called target words).
- Word2Vec words with
  - center words
  - and their context words.



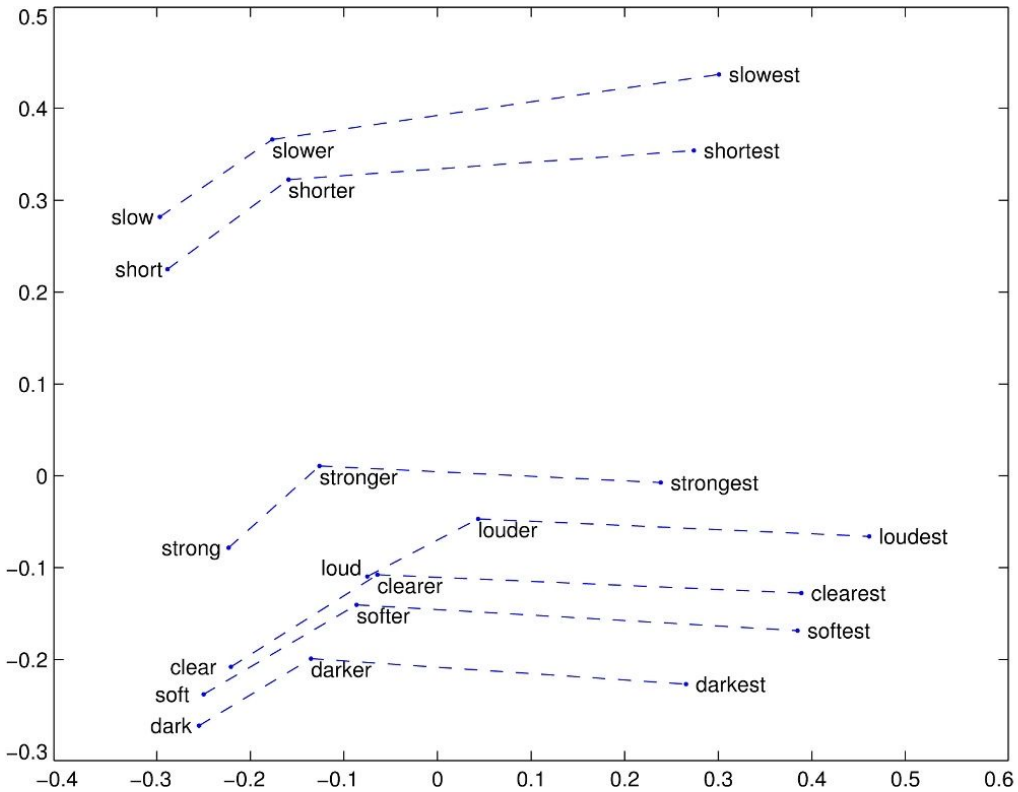
Word2vec

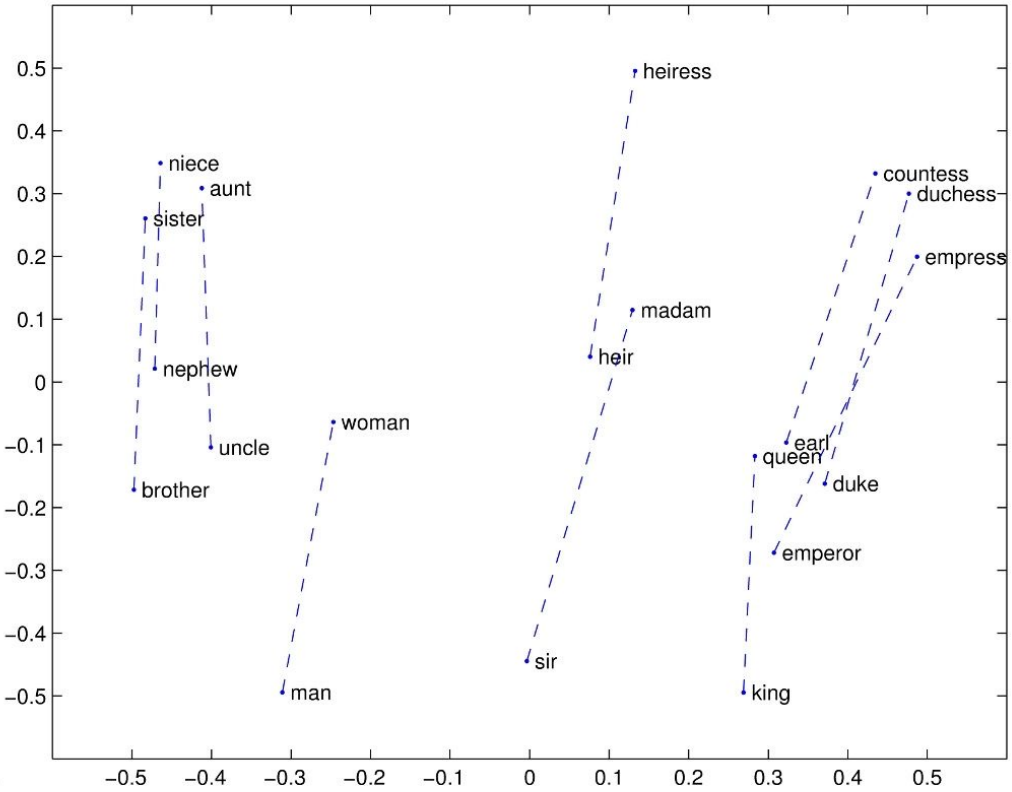
- Plusieurs variations autour du même principe
  - Continuous Bag of Words (CBOW)
    - On connaît le context
    - On cherche le mot manquant

- Il existe des modèles pré-entrainé (comme pour les images avec ResNet, VGG16, …)
  - On peut donc réutiliser ces 'embeddings'
  - Généralement vecteur de taille
    - 50, 150 ou 300



Exemple de résultat





13

Word2vec (Exemple)

```
from gensim.models import KeyedVectors, Word2Vec
import gensim.downloader as api

glove_model =
glove_model = KeyedVectors.load(" glove-wiki-gigaword-50.model")
# → revient à charger une matrice d'embedding

# vocabulary size
vocab_size = len(glove_model.key_to_index)

# embedding size
emb_size = 50

# Initialize the vectorizer layer with the pretrained vocabulary
new_vectorizer_layer = layers.TextVectorization(
        output_mode="int",
        output_sequence_length=max_len,
        vocabulary=list(glove_model.key_to_index), # Initialize the vocabulary
)
```

Word2Vec (Exemple cont'd)

```
# Construire la matrice d'embedding
embedding_matrix = np.zeros((vocab_size+2, emb_size ))
for i, word in enumerate(glove_model.key_to_index):
    embedding_matrix[i+2] = glove_model[word]

# A quoi correspond l'indice 0 et 1 ?

# Créer la couche d'embedding
layers.Embedding(vocab_size+2, emb_size ,
                 weights=embedding_matrix,
                 trainable=True) # or False
```

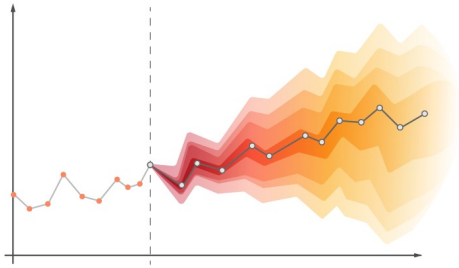
Corrélation temporelle
Neurone récurrent

Feature independence

- Until now, the order of the entries was not taken into account...

- Except that this one is important:
  - word order:
    - I like chocolate but not beer ≠ I like beer but not chocolate
  - Music ?

  - Signal ?



Motivation for recurrent network

- Humans don't start their thinking from scratch every second
  - Thoughts have persistence
- Traditional neural networks can't characterize this phenomena
  - Ex: classify what is happening at every point in a movie
  - How a neural network can inform later events about the previous ones
- Recurrent neural networks address this issue
  - Some applications
    - NLP: Same word may have a different label depending on the context.
      - NER - Apple CEO Tim Cook eat an apple
      - POS - The bank is located on the bank (La banque est située sur la berge)
    - Forcasting - Time-series Prediction
- How?
  - Add state to artificial neurons

What are RNNs?

- Main idea is to make use of sequential information
- How RNN is different from neural network?
  - Vanilla neural networks (MLP) assume all inputs and outputs are independent of each other
  - But for many tasks, that's a very bad idea
- What RNN does?
  - Perform the same task for every element of a sequence
    - That's what recurrent stands for
  - Output depends on the previous computations!
- Another way of interpretation – RNNs have a "memory"
  - To store previous computations

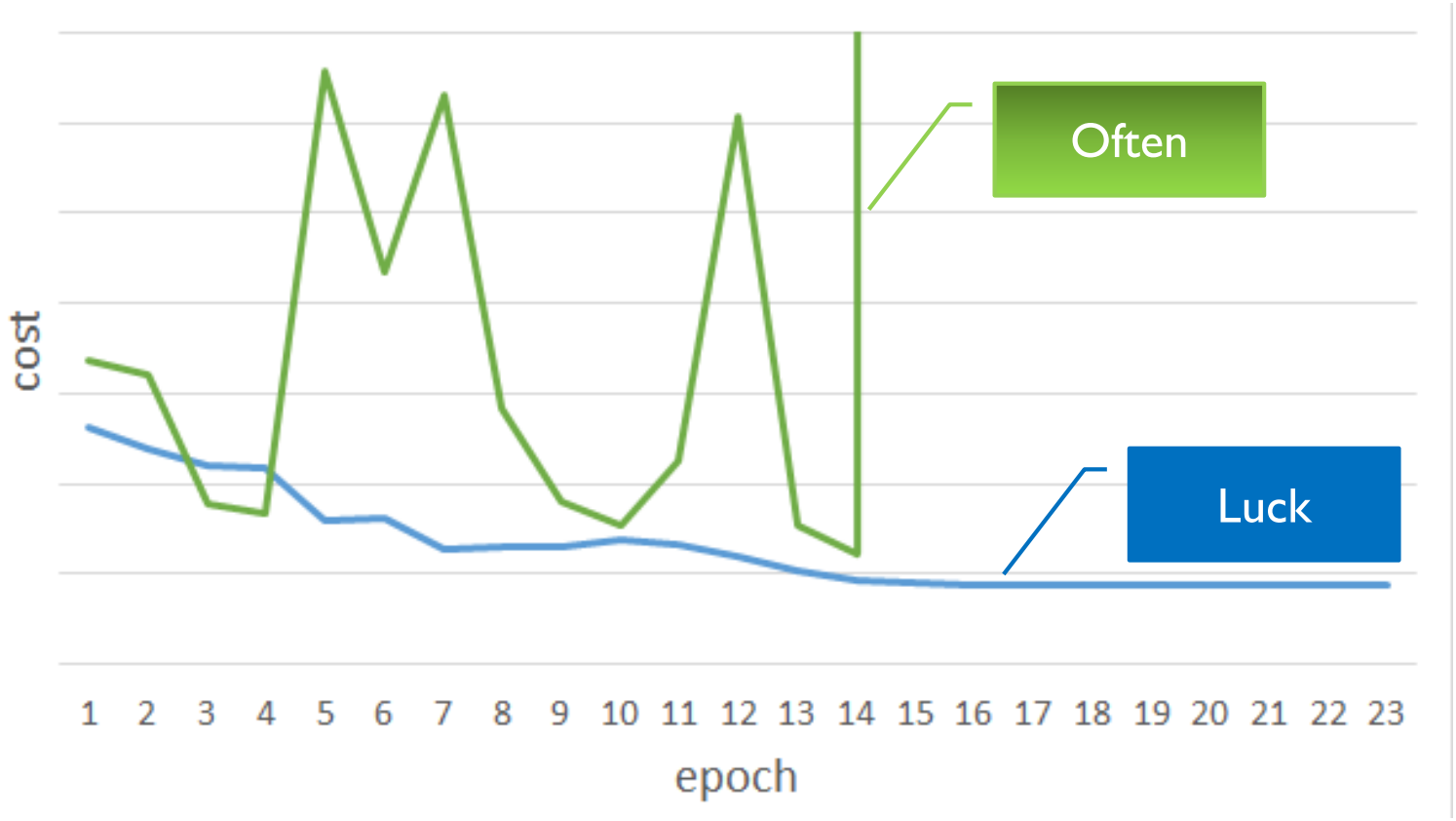
From vanilla NN to recurrent NN

- Vanilla cell
  - $y = F(U.X)$
- Recurrent cell → use for each time step the **same weights matrix**
  - $h_t = F(W_h.[h_{t\_1}, X_t])$
  - $\hat{y_t} = G(W_y.[h_t])$
- Recurrent layer, step by step
  - at each time step
    - A new entry is being supplied
    - And a new memory (h_t) is calculated using:
      - The new input X_t
      - The output of the previous step h_{t_1}
    - And a new output ($\hat{y_t}$) is calculated using
      - The new memory (h_t)

20

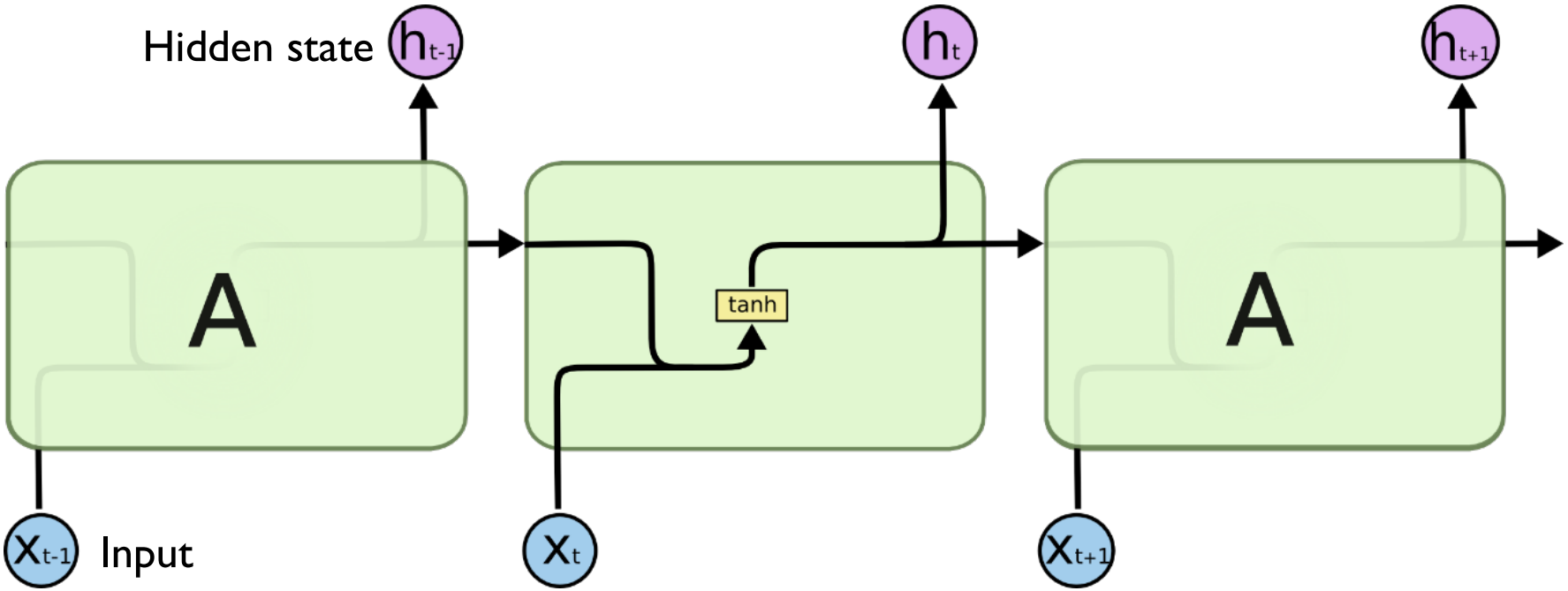
Problems with naive RNN

- RNNs do not learn easily
- Unfolding the network for learning leads to vanishing gradient problems or gradient exploding problem!



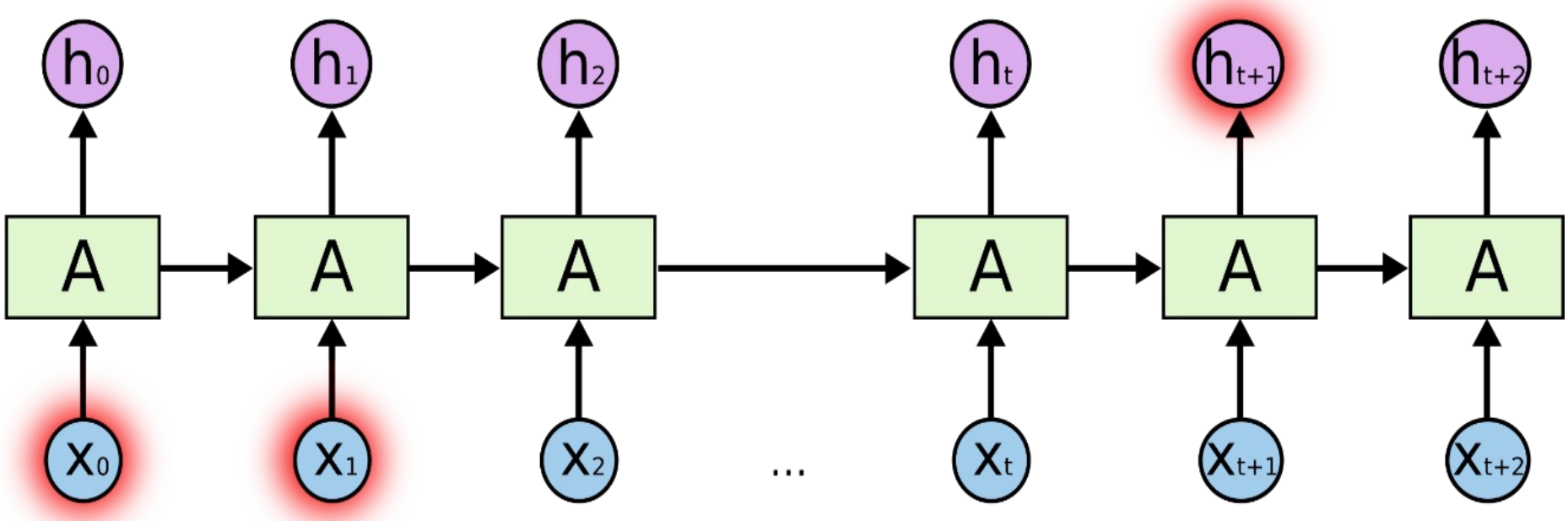
From vanilla Short Term Memory…

- Traditionnal implementation of RNN cells
  - $h_t = tanh(W \cdot [h_{t-1}, X_t])$
  - Involves a single level of processing
  - No control of hidden state
  - Creating the risk of the evanescent gradient.



… to LSTM (Long Short Term Memory)

- But if the context is far from the word to predict
  - Many iterations separate them!
  - **Pb1:** Possible gradient problem
  - **Pb2:** We want to be able to control the information to be stored in memory: to preserve the past or, on the contrary, to renew the state

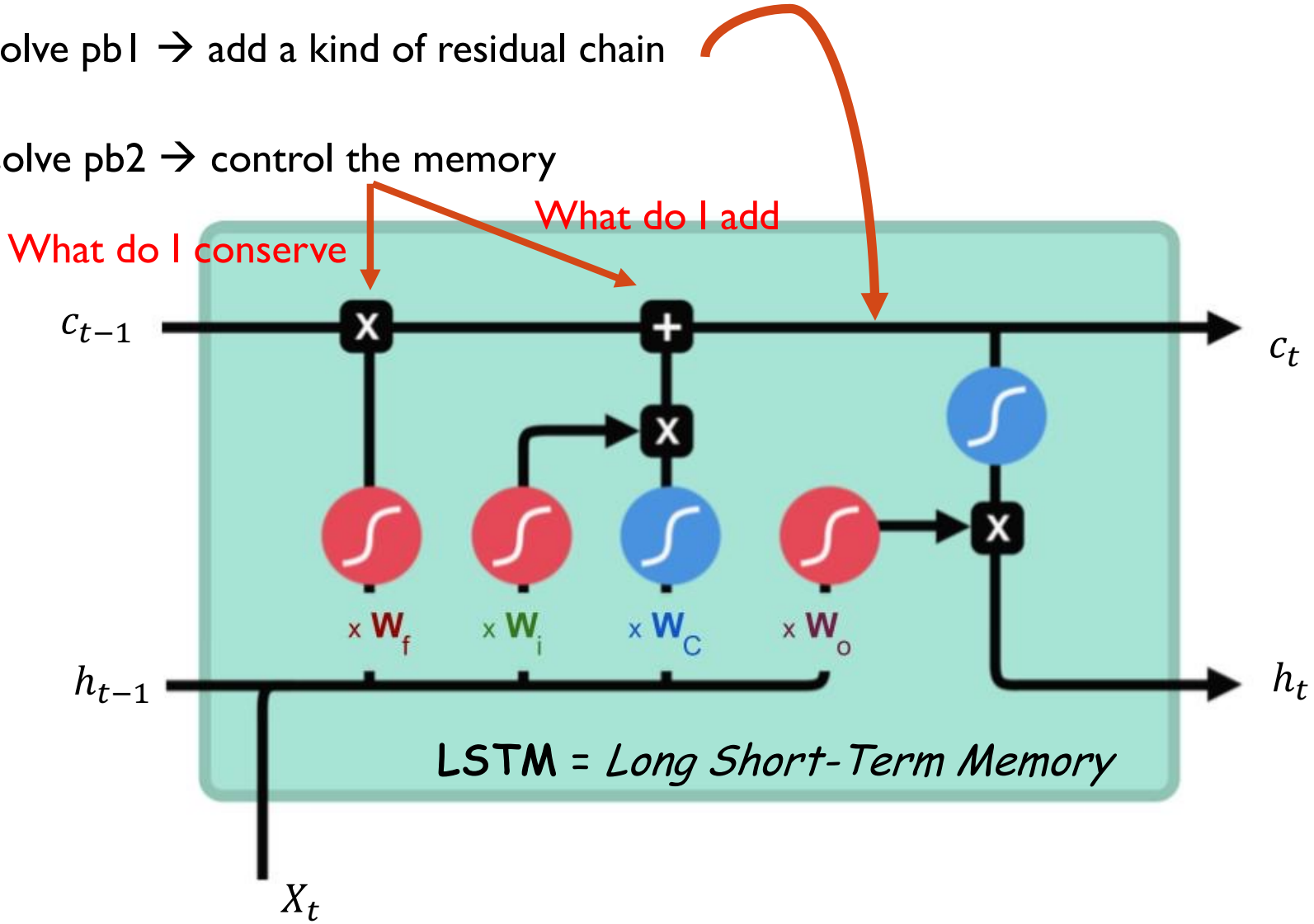


I grew up in France… I speak French…

Dealing with the vanishing gradient problem → LSTM cell

Resolve pb1 → add a kind of residual chain

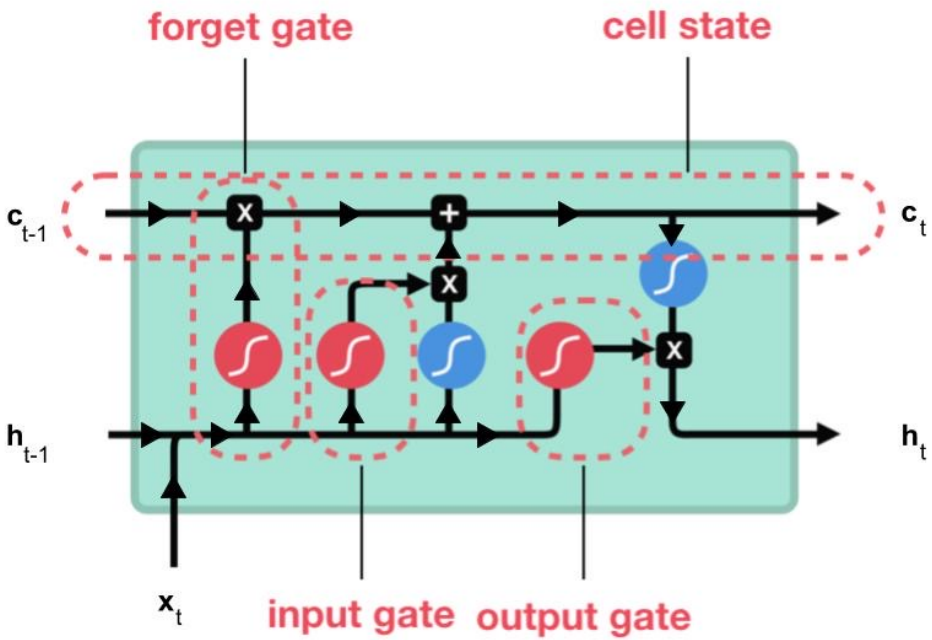
Resolve pb2 → control the memory



(crédit : image modifiée de Michaël Nguyen)

LSTM cell

- Contain three "gates":
  - Calculation zones that regulate the flow of information (by performing specific actions).
  - Forget gate (porte d'oubli)
  - Input gate (porte d'entrée)
  - Output gate (porte de sortie)
- h_t: Hidden state (état caché)
  - The eventual output
- c_t: Cell state (état de la cellule)
  - Like residual



(crédit : image modifiée de Michaël Nguyen)

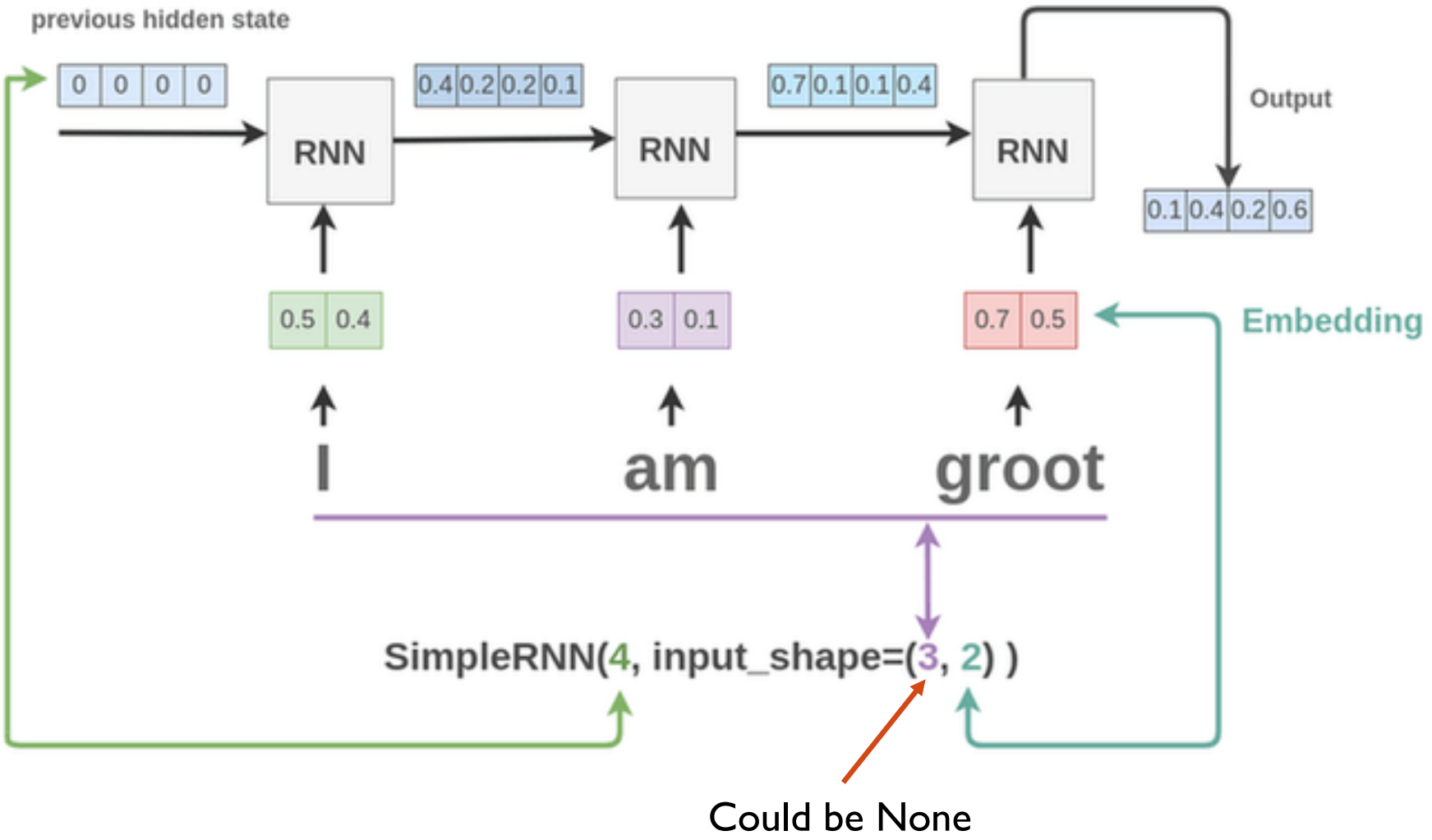
Keras Long Short-Term Memory Cell

- **from** tf.keras.layers **import** LSTM

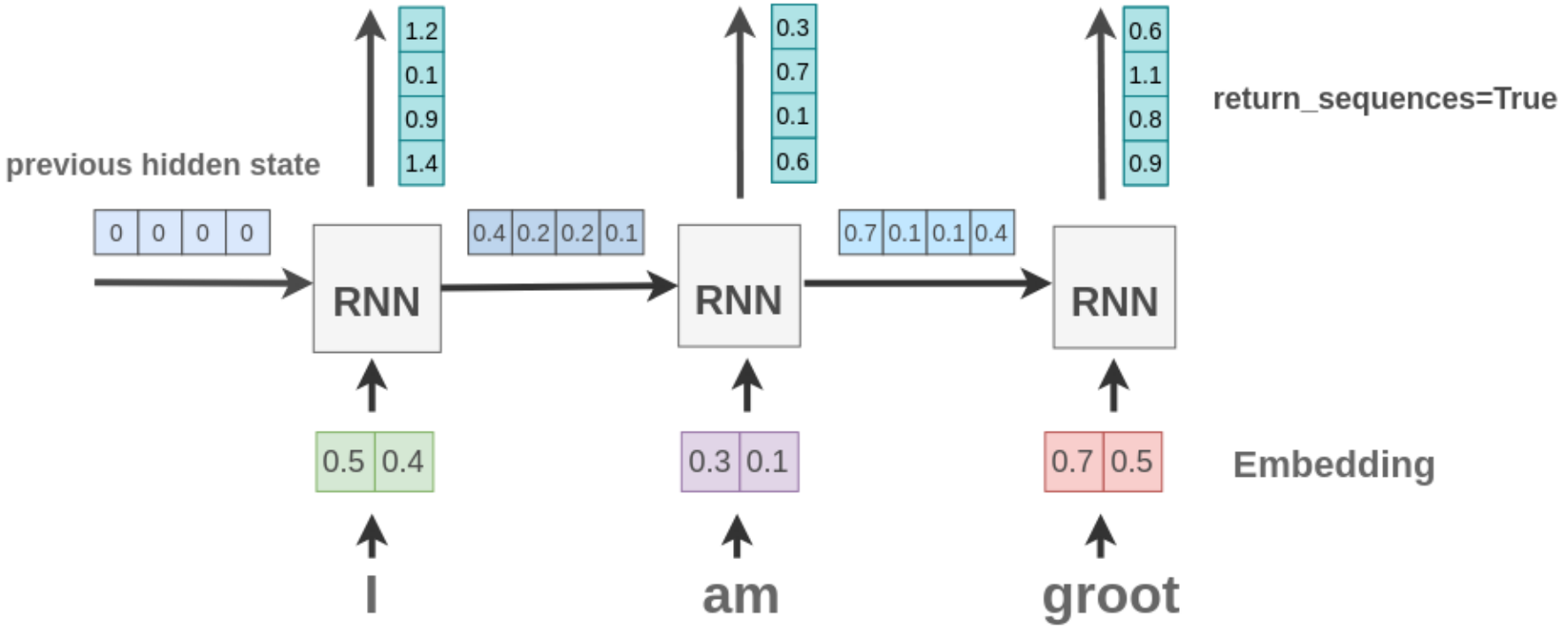
- Main params
  - **Units:** dimension of output space

  - **return_sequences:** True or False
    - If False return only the last output
    - If True return the full sequence of the output sequence
      - Output sequence = hidden state (the vocabulary change regardind documentation)
  - **return_state**: True or False
    - If True return 3 values
      - The full output sequence or only the last one (depend on return_sequences)
      - The last output sequence
      - The cell state
    - If False return nothing

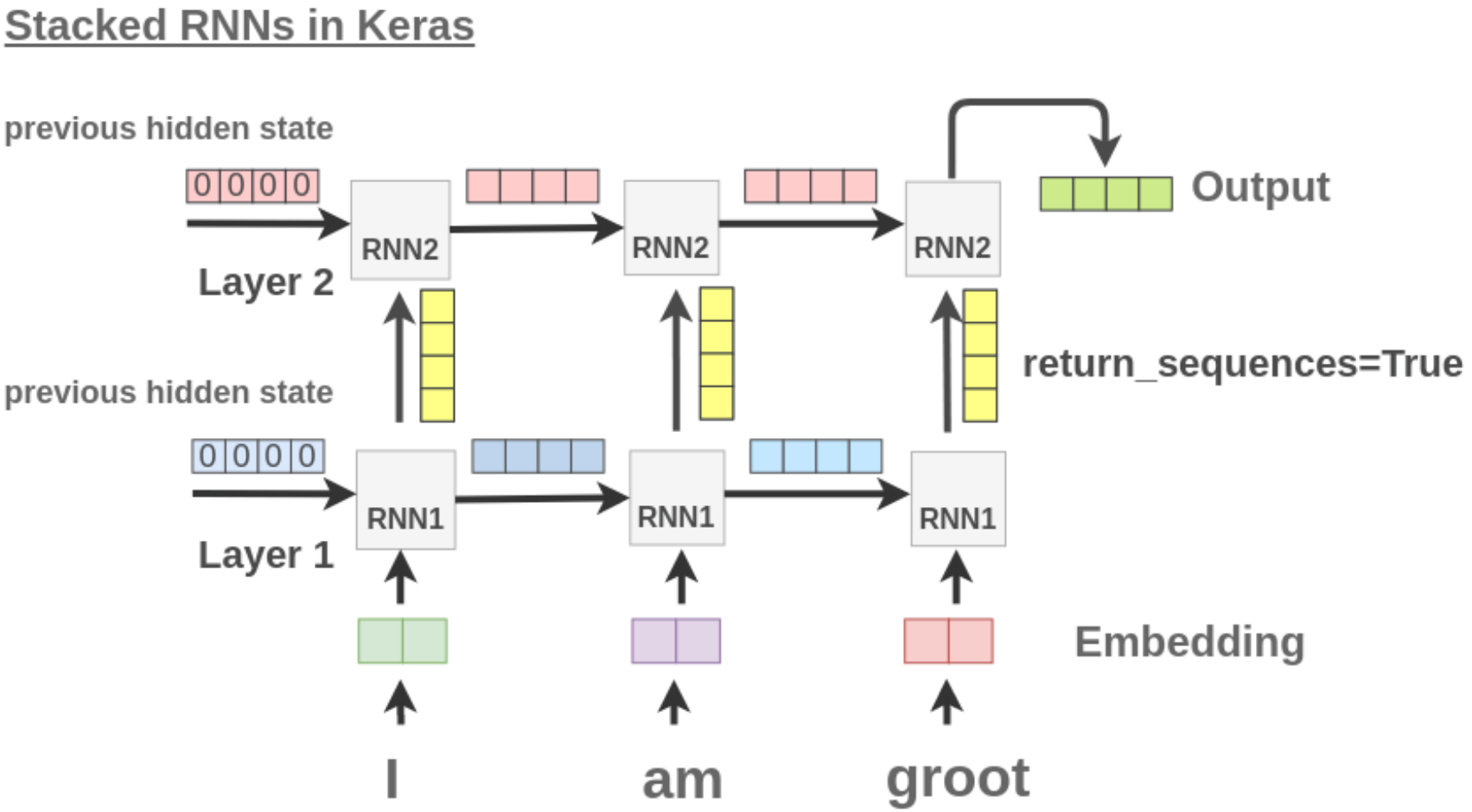
RNN cells (return_sequence=False)



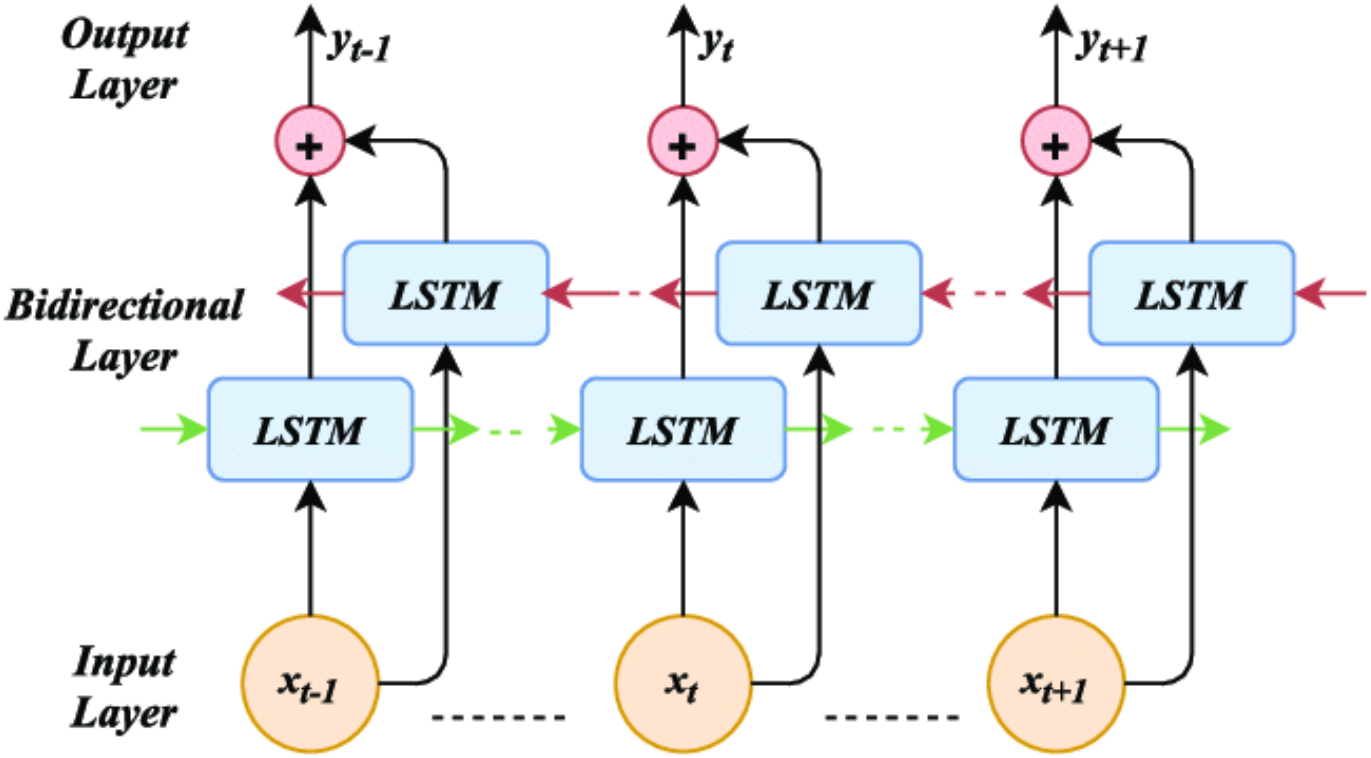
RNN cells (return_sequence=True)



Stacked RNN cells



Bi-RNN cells (keras.layers.Bidirectional)

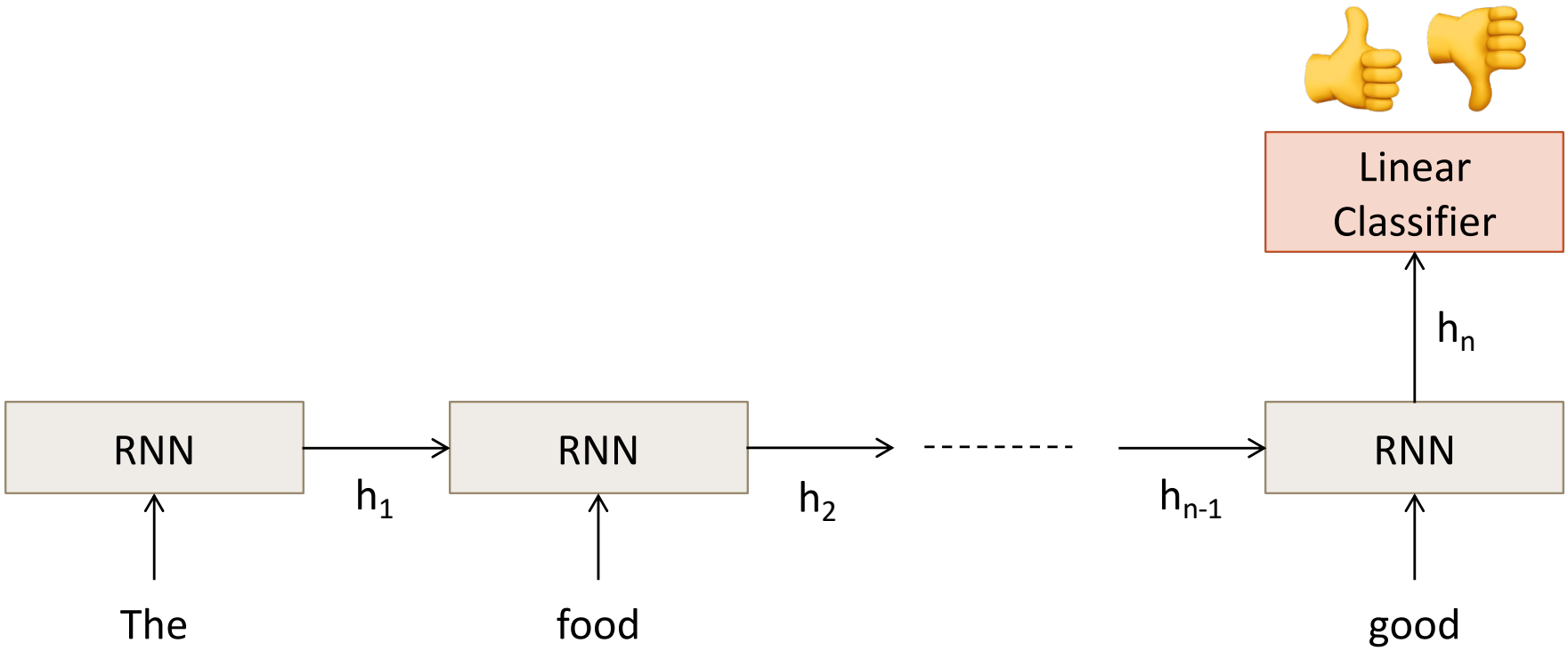


- keras.layers.Bidirectional
- Paramètres principaux
  - layer
  - merge_mode : '*sum*', '*mul*', '*concat*' (default), '*ave*'

- Exemple : Bidirectional(LSTM, merge_mode='sum')

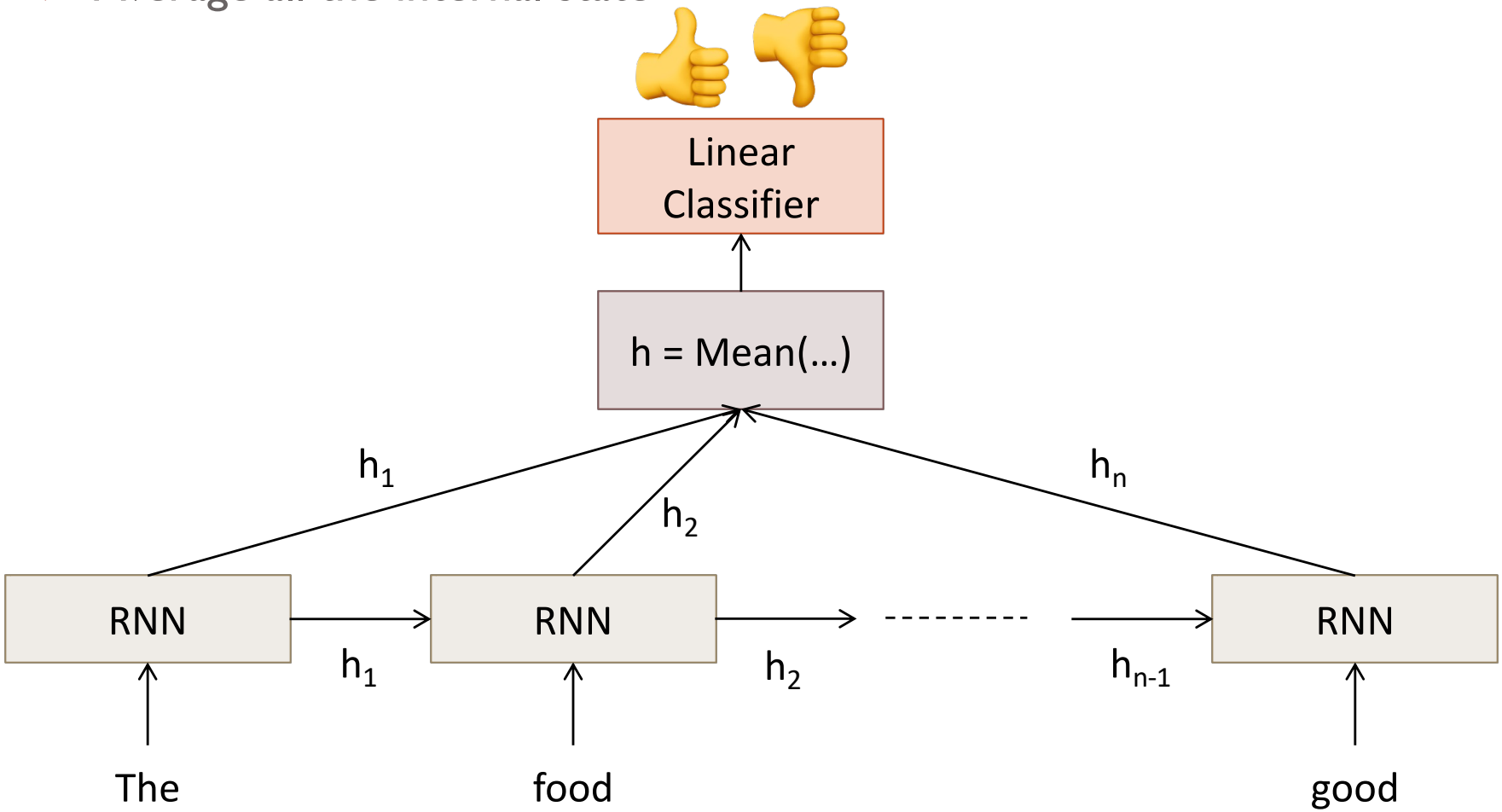
Sentiment analysis/Document classification solution2

- retrieve only the last state



Sentiment analysis/Document classification solution2

- Other possible architecture
  - Average all the internal state



http://deeplearning.net/tutorial/lstm.html

Sample

```
_EMB_SIZE = 50, 100, 150 or 300
_HIDDEN_SIZE = 64, 128 or 256
_SEQ = True or False
new_vectorizer_layer = layers.TextVectorization(output_mode="int",
        output_sequence_length=max_len, vocabulary=list(glove_model.key_to_index))

inputs = layers.Input(shape=(None, 1), dtype='string')
emb = layers.Embedding(len(glove_model.key_to_index)+2, _EMB_SIZE)(inputs)
hidden1 = layers.LSTM(_HIDDEN_SIZE, return_sequences=_SEQ)(hidden)
if _SEQ:
    hidden2 = layers.GlobalAveragePoolind1D()(hidden1)          # solution 2
    assert np.array_equal(hidden1.shape==hidden2.shape)
else:
    hidden2 = hidden1               # Solution 1
ouputs = layers.Dense(_NB_CLASSES, activation='softmax')(hidden2)

model = models.Model(inputs, outputs)
```

NER: Named Entity Recognition

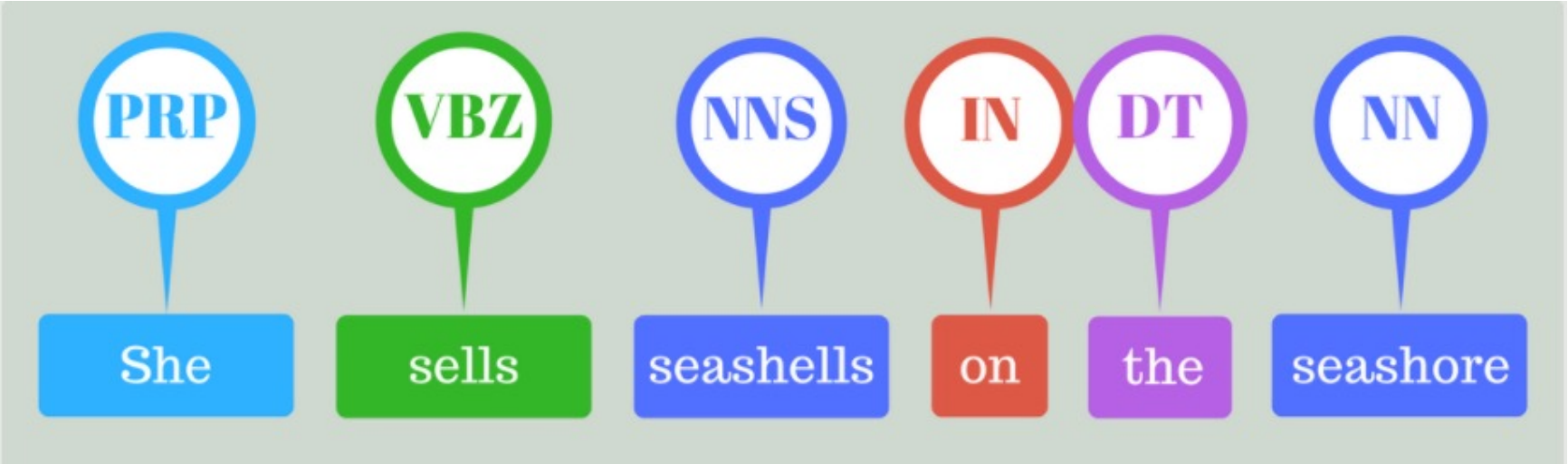
- Affect a label to each word
  - **find** and **classify** names in text
    - Could bean entity : number, country, person, … (NER)

In fact, the Chinese NORP market has the three CARDINAL most influential names of the retail and tech space – Alibaba GPE , Baidu ORG , and Tencent PERSON (collectively touted as BAT ORG), and is betting big in the global AI GPE in retail industry space . The three CARDINAL giants which are claimed to have a cut-throat competition with the U.S. GPE (in terms of resources and capital) are positioning themselves to become the 'future AI PERSON platforms'. The trio is also expanding in other Asian NORP countries and investing heavily in the U.S. GPE based AI GPE startups to leverage the power of AI GPE . Backed by such powerful initiatives and presence of these conglomerates, the market in APAC AI is forecast to be the fastest-growing one CARDINAL , with an anticipated CAGR PERSON of 45% PERCENT over 2018 - 2024 DATE .

To further elaborate on the geographical trends, North America LOC has procured more than 50% PERCENT of the global share in 2017 DATE and has been leading the regional landscape of AI GPE in the retail market. The U.S. GPE has a significant credit in the regional trends with over 65% PERCENT of investments (including M&As, private equity, and venture capital) in artificial intelligence technology. Additionally, the region is a huge hub for startups in tandem with the presence of tech titans, such as Google ORG , IBM ORG , and Microsoft ORG .

POS: Part of Speech Tagging

- Affect a label to each word
  - **find** and **classify** names in text
    - Could be a function : noun, verb, adverbs, … (POS)



Sample

- Généralement un dataset pour NER se présente avec des mots dejà découpé en token et avec leur propre label
  - On n'utilise donc pas TextVectorization

```
inputs = layers.Input(shape=(_MAX_LEN or None), dtype='int')
emb = layers.Embedding(len(glove_model.key_to_index)+2, _EMB_SIZE)(inputs)
hidden = layers.LSTM(_HIDDEN_SIZE, return_sequences=True)(hidden)
ouputs = layers.Dense(_NB_TAGS, activation='softmax')(hidden)

model = models.Model(inputs, outputs)
```

Label encoding for NER

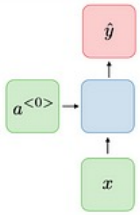
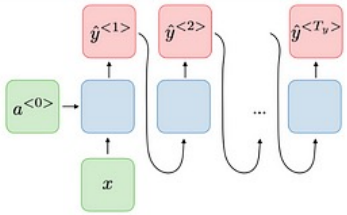
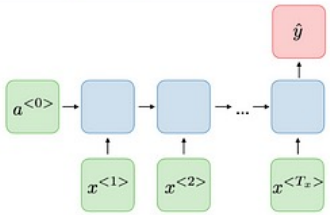
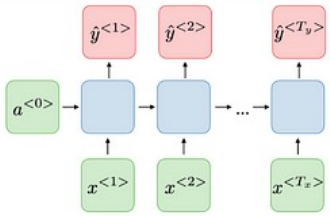
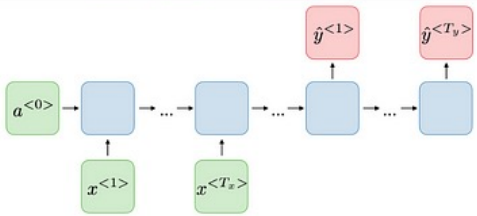
Il existe de très ombreuse manière de définir les labels pour une tache de NER. Dans le cadre de ce lab, on utilisera le 'IOB encoding'

	IO encoding	IOB encoding
Fred	PER	B-PER
showed	O	O
Sue	PER	B-PER
Mengqiu	PER	B-PER
Huang	PER	I-PER
's	O	O
new	O	O
painting	O	O

Types of Errors in NER systems

- Evaluer un modèle NER est assez complexe car il faut se mettre d'accord sur les erreurs possibles.
- Dans le cadre de ce lab, nous allons faire simple et utiliser la matrice de confusion
  - i.e. la prédiction est correcte si elle correspond au label à prédire
- A titre d'exemple, voici les différents types d'erreur
  1. **Spurious (SPU):** : complètement faux positif : Une entité est prédite par le modèle NER, mais n'est pas annotée dans le texte étiqueté
  2. **Missing (MIS)** : complètement faux négatif : Une entité étiquetée à la main n'est pas prédite par le modèle
  3. **Incorrect (INC)** : mauvaise étiquette, bonne portée : une entité étiquetée à la main et une entité prédite ont les mêmes portées mais des étiquettes différentes.
  4. **Partial (PAR)** : mauvaise étiquette, chevauchement des portées : une entité étiquetée manuellement et une entité prédite ont des portées qui se chevauchent mais des étiquettes différentes.
  5. **Partial (PAR)** : étiquette correcte, chevauchement des portées : une entité étiquetée manuellement et une entité prédite ont des portées qui se chevauchent avec les mêmes étiquettes.

Main type of RNN architecture

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment analysis Document classification
Many-to-many $T_x = T_y$		NER POS
Many-to-many $T_x \neq T_y$		Translation Chatbot

This week

Next week

Lab of the day

- Tache : NER (Naming Entity Recognition)
  - le jeu de données est sur lms.

- Partie : embedding pré-entrainé
- Partie 2 : On construit un réseau récurrent de type 'Many to Many' avec len(X)=len(y)
  - Solution 1 : Input → Embedding → LSTM → Output
  - Solution 2 : Input → Embedding → BiLSTM → Output
  - Solution 3 : Input → Embedding → BiLSTM('ave') → GRU → Output

  - On utilise EarlyStopping et on babysit l'apprentissage
  - Les éléments pour construire le réseau de neurones sont dans les transparents
- Partie 3 : idem solution 1 ou 2 avec embedding pré-entrainé