INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Natural Language Processing (NLP)

IA Frameworks

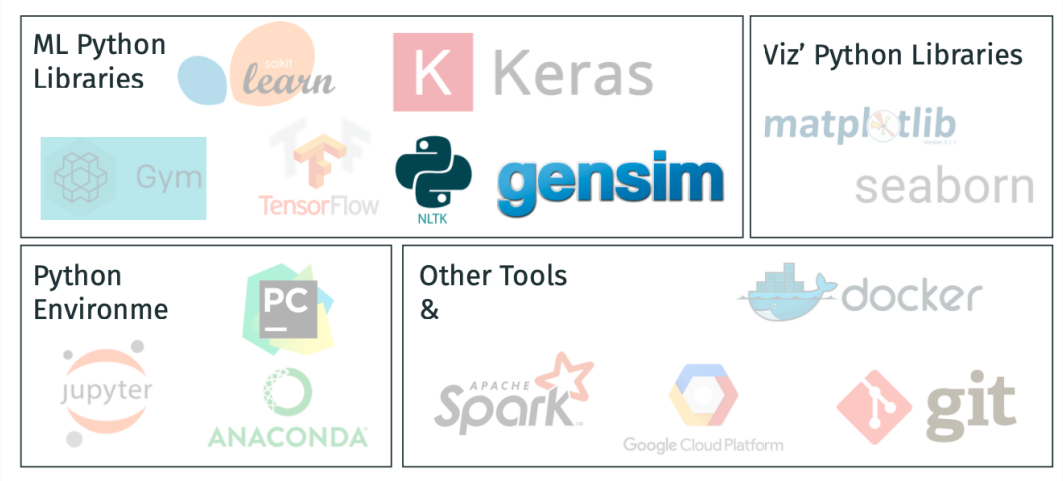
Brendan Guillouet

December 16th, 2019

Institut National des Sciences Appliquées

Plan de Présentation

Data Science tools



Introduction

Multiple **domaines de recherche** et une très grande variété d'**applications** :

- Recherche d'information.
  - *Moteur de recherche (Google, Yahoo).*
- Reconnaissance de pattern.
  - *Extraction d'information. Scrawling de page.*
- Analyse de sentiments.
  - *Marketing. Commentaires de sites..*
- Génération automatique de textes.
  - *ChatBot, Article de journaux.*
- Traduction automatique.
  - *Google Translate, DeepL.*
- Désambiguïsation.
  - *Sécurité.*

Exemple : La catégorisation de produits

Objectif :

Automatiser la catégorisation des produits dans l'arborescence du site. *(Concours Datascience.net https ://www.datascience.net/fr/challenge/20/details)*

Difficultés :

- Données : **textuelles**.
  - *Algorithmes d'apprentissage non adaptés.*
- Gérer des gros volumes de données (**Big Data**) :
  - *dizaines de millions de produits.*
- Données **réelles** :
  - *étape de nettoyage importante.*

Données issues du concours proposé par **Cdiscount** et disponible sur **Datascience**.

Fichier d'apprentissage de 15.786.885 produits.

- 47 Catégories de niveau 1.
- 536 Catégories de niveau 2.
- 5789 Catégories de niveau 3.

Champ	Type de données	Description
Identifiant produit	String	Identifiant unique du produit
Catégorie 1	String	Catégorie de niveau 1
Catégorie 2	String	Catégorie de niveau 2
Catégorie 3	String	Catégorie de niveau 3
Description	String	Description produit
Libelle	String	Description courte
Marque	String	Marque du produit

Table 1 – Données Cdiscount.

Exemples de données

Categorie1	Description	Marque
VIN - ALCOOL - LIQUIDES	Eau minérale gazeuse - Composition moyenne en ...	ROZANA
VETEMENTS - LINGERIE	AUDIGIER Sweats homme sweat à capuche panther ...	AUCUNE
TV - VIDEO - SON	Enceinte portable bluetooth fonction main libr...	MUSE
TENUE PROFESSIONNELLE	Pantalon G-Rok Carbone/Orange Taille L Molinel...	MOLINEL
TELEPHONIE - GPS	Coque souple Noire pour GOOGLE NEXUS 4 motif D...	MUZZANO
TATOUAGE - PIERCING	plug ecarteur coloris rasta - plug carteurcol...	AUCUNE
SPORT (NEW)	Brassière HEATGEAR "ENDURE D" UNDER ARMOUR - M...	AUCUNE
SONO - DJ	La classique OM PRO. Avec un équipage mobile r...	AUCUNE
PUERICULTURE	Coffret de naissance - COFFRET DE NAISSANCE p...	AMADEUS
PRODUITS FRAIS	Yaourt brassé nature - VELOUTE - Yaourt brassé...	VELOUTE
POINT DE VENTE - COMMERCE - ADMINISTRATION	Vikuiti MySafeDisplay Film de protection écran...	VIKUITI
PHOTO - OPTIQUE	Batterie pour JVC GR-D275 series - Batterie po...	ABOUT BATTERIES
PARAPHARMACIE	Originaire d'Europe et connue depuis des milli...	AUCUNE
MEUBLE	Caisson mobile 2 tiroirs mobilier Optima - Cai...	AUCUNE
MERCERIE	couleur : linde - 25m de ruban taffetas uni en...	AUCUNE

Difficultés

- **Bruits** liés aux fautes d'orthographes, aux accords, à la conjugaison.
- Nombreux termes **non significatifs**.
- Termes significatifs/Décision liés **au contexte**.
- Traitement **différent** d'une langue à l'autre.
- **Transcription** aux outils machine learning.

$\Rightarrow$ Étape de pré-traitement des données très importante.

Nettoyage des données

Racinisation (Stemming)

Problème : Des termes peuvent-être écrits de **différentes façons** (accents, conjugaison, genre, pluriels..) et néanmoins avoir le **même sens**.

Solution : Remplacer les mot par leur **racine**.

Exemple :

- épée, épee, epée = epe
- vert, verts, verte, vertes = vert
- mange, manger, mangez, mangent = mang

De nombreux algorithmes différents propre aux langues étudiées Algorithme utilisé : *Snowball (Porter)*

Problème :

Des termes très usuels, donc non différenciateurs, peuvent perturber l'apprentissage.

Solution :

Lors de l'étape de preprocessing, les mots les plus communs sont supprimés à partir d'une liste de Stopword :

(["a", "afin", "ai", "ainsi", "après", "attendu", "au", "aujourd", "auquel", "aussi","autre", "autres", "aux", "auxquelles", "auxquels", "avait", "avant", "avec", "avoir","c", "car", "ce", "ceci", "cela", "celle", "celles", "celui", "cependant", "certain","certaine", "certaines", "certains", "ces", "cet", "cette", "ceux", "chez", "ci",])

⇒ Également propre à la langue !

Autres étape de nettoyages

La plupart de ces étapes varient en fonction des données et de l'objectif à réaliser.

- Suppression de la ponctuation.
- Incrémenter les stopwords avec de la connaissance métier.
- Suppression des caractéristiques numériques (sauf pour les marques) $\Rightarrow$ Importance de la connaissance métier !
- Suppression de code HTML (*BeautifulSoup*).

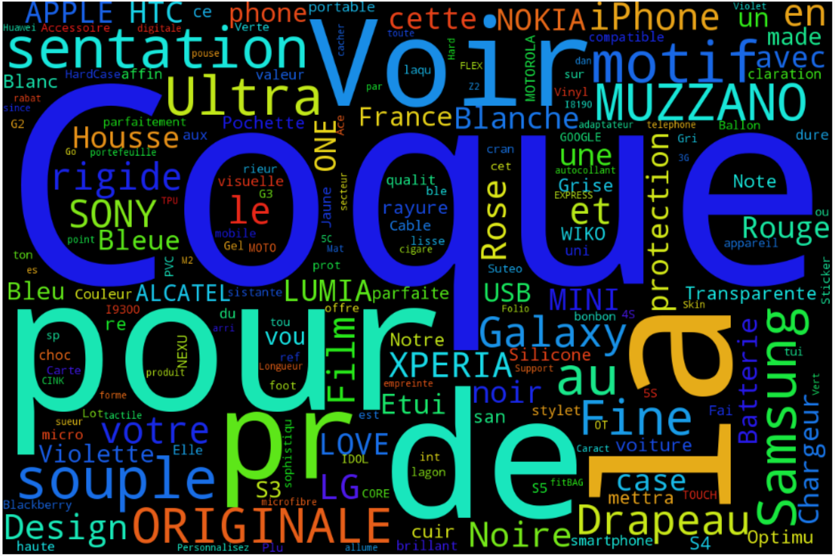
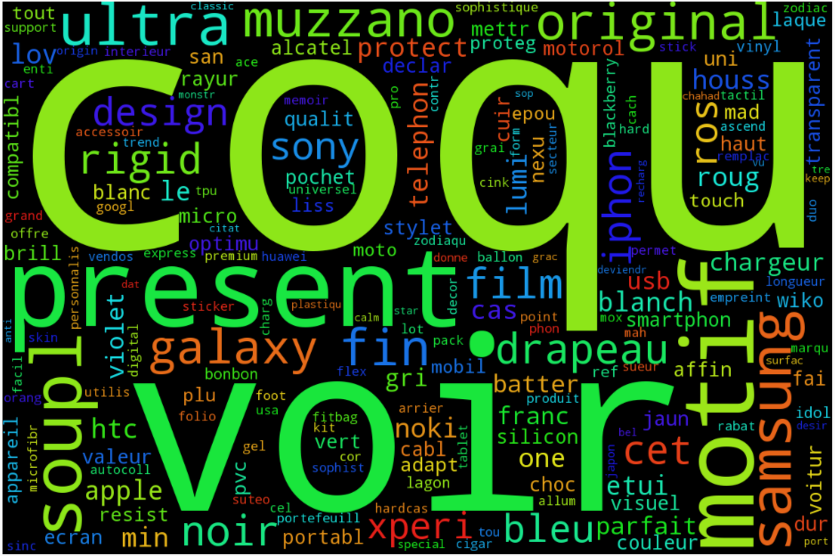
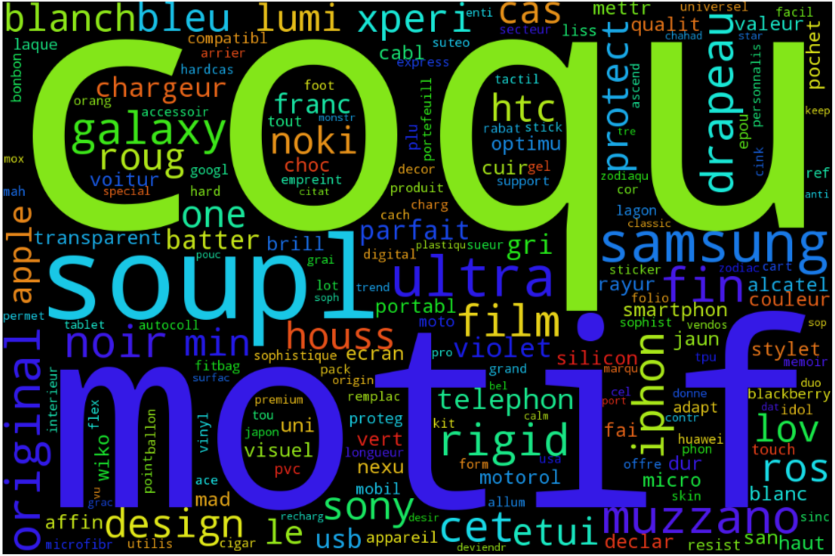

APPLE
HTC
ce
phone
portable
cette
NOKIA
iPhone
Violet
un
en
Huawei
Accessoire
digitale
Verte
toute
compatible
made
sentation
Voir
motif
avec
pouse
cacher
Hard
FLEX
MOTOROLA
dan
laqu
sur
claration
Blanc
HardCase
affin
valeur
par
Z2
Vinyl
Ultra
I8190
MUZZANO
rabat
since
France
Blanche
adaptateur
telephone
Ballon
G2
parfaitement
Pochette
Ace
GOOGLE
Gri
dure
Housse
aux
ONE
cran
Go
portefeuille
une
3G
autocollant
rieur
cet
Coque
visuelle
qualit
Note
rigide
G3
ble
Grise
EXPRESS
TPU
le
Jaune
secteur
rayure
et
SONY
Cable
ou
Rose
protection
Rouge
mobile
WIKO
point
Bleue
lisse
appareil
5C
Gel
MOTO
Mat
cigare
uni
ton
PVC
M2
prot
Suteo
es
Sticker
Bleu
Couleur
ALCATEL
sistante
LUMIA
parfaite
USB
MINI
Transparente
offre
Folio
skin
I9300
du
bonbon
4S
re
tou
arri
Galaxy
sp
pour
vou
Notre
la
NEXU
Film
tui
choc
ref
Silicone
Carte
XPERIA
Longueur
Support
CINK
produit
au
Vert
foot
empreinte
5S
forme
noir
Samsung
Etui
Batterie
Lot
tactile
est
stylet
Fai
sueur
san
Fine
OT
Chargeur
votre
micro
pr
de
Caract
voiture
Violette
Elle
sophistiqu
LOVE
int
IDOL
lagon
souple
case
S5
fitBAG
S3
LG
CORE
mettra
TOUCH
Blackberry
microfibre
Design
ORIGINALE
cuir
Noire
Drapeau
Optimu
haute
Personnalisez
Plu
allume
brillant
smartphone
S4

Illustration - Catégorie "Téléphonie - GPS"




blanch
bleu
lumi
xperi
enti
cas
mettr
universel
facil
compatibl
secteur
liss
qualit
valeur
laque
arrier
cabl
suteo
chahad
star
hardcas
express
bonbon
coqu
drapeau
protect
ascend
personnalis
pochet
chargeur
foot
tactil
orang
accessoir
franc
tout
monstr
htc
galaxy
noki
rabat
stick
optimu
tre
epou
keep
portefeuill
plu
roug
mox
choc
cuir
gel
decor
googl
voitur
empreint
support
zodiaqu
cink
ref
mah
special
hard
citat
produit
cor
anti
charg
cach
one
lagon
classic
apple
transparent
batter
brill
parfait
plastiqu
sueur
gri
samsung
pouc
digital
grai
soupl
ultra
sticker
zodiac
cart
lot
trend
rayur
fin
alcatel
soph
folio
couleur
portabl
permet
smartphon
film
tablet
autocoll
moto
sophist
vendos
sop
houss
noir
min
jaun
original
violet
stylet
sophistique
ecran
pro
silicon
tpu
memoir
fitbag
pack
grand
marqu
surfac
origin
iphon
interieur
motif
bel
duo
premium
remplac
cel
donne
blackberry
telephon
port
dat
idol
flex
adapt
tou
calm
contr
vinyl
proteg
kit
lov
japon
uni
fai
huawei
wiko
pointballon
vert
phon
visuel
rigid
touch
longueur
usa
offre
dur
nexu
pvc
form
ros
ace
micro
motorol
vu
mobil
skin
mad
blanc
grac
allum
sony
recharg
desir
muzzano
sinc
affin
design
le
usb
appareil
cetetui
declar
san
haut
microfibr
utilis
cigar
deviendr
resist

Librairies pour le traitement du langage (TP)

- NLTK *(Python, Spark)* : Traitement du langage (racinisation, stopwords, ...).
- Lucène *(Java)* : Librairie d'indexation et de recherche de texte.
- BeautifulSoup : Suppression des balises HTML.
- Regex : Langage de recherche de texte.

On définit une fonction de nettoyage :

```
def clean_txt(txt):
    ### remove html stuff
    txt = BeautifulSoup(txt,"html.parser",from_encoding='utf-8').get_text()
    ### lower case
    txt = txt.lower()
    ### special escaping character '...'
    txt = txt.replace(u'\u2026','.')
    txt = txt.replace(u'\u00a0',' ')
    ### remove accent btw
    txt = unicodedata.normalize('NFD', txt).encode('ascii', 'ignore').decode("utf-8")
    ### remove non alphanumeric char
    txt = re.sub('[^a-z_]', ' ', txt)
    ### remove french stop words
    tokens = [w for w in txt.split() if (len(w)>2) and (w not in stopwords)]
    ### french stemming
    tokens = [stemmer.stem(token) for token in tokens]
    ### tokens = stemmer.stemWords(tokens)
    return ' '.join(tokens)
```

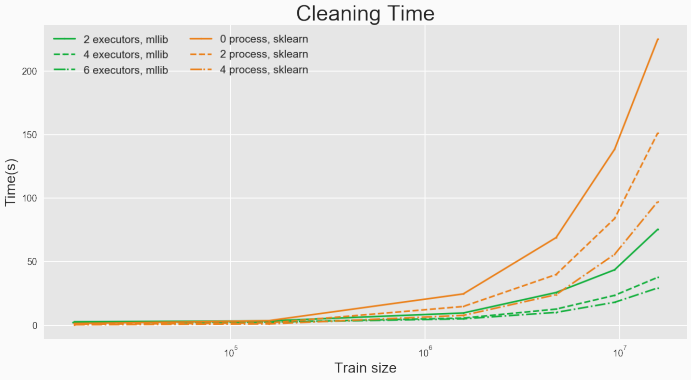
que l'on va ensuite appliquer sur chaque description de texte.

Spark possède des Transformer qui permettent d'appliquer ces transformations.

```
STOPWORDS = set(nltk.corpus.stopwords.words('french'))
# Tokenizer
regexTokenizer = RegexTokenizer(inputCol="description", outputCol="tokenizedDescr",
                pattern="[^a-z_]", minTokenLength=3, gaps=True)
dataTokenized = regexTokenizer.transform(dataEchDF)

# StopWordsRemover q
remover = StopWordsRemover(inputCol="tokenizedDescr",
                           outputCol="tokenizedRemovedDescr",
stopWords = list(STOPWORDS))
dataTokenizedRemoved = remover.transform(dataTokenized)

# Stemmer
STEMMER = nltk.stem.SnowballStemmer('french')

def clean_text(tokens):
    tokens_stem = [ STEMMER.stem(token) for token in tokens]
    return tokens_stem
udfCleanText =  udf(lambda lt : clean_text(lt), ArrayType(StringType()))
dataClean = dataTokenizedRemoved.withColumn("cleanDescr",
udfCleanText(col('tokenizedRemovedDescr')))
```

- RegexTokenizer = Regex + Tokenizer.

TP - Résultat - Temps Nettoyage



Vectorisation et Word Embedding

Objectifs et difficultés

- Transformer la liste de mots sous un format **interprétable** par les différents algorithmes d'apprentissage.
- Gérer le très **grand nombre** de features.
  - Exemple sur 21.543 lignes de la catégorie "TÉLÉPHONIE-GPS"
  - 24.486 mots uniques -> 8.384 après nettoyage.
- Choisir des poids **significatifs**.

$\Rightarrow$ 2 types de solutions :

- Basées sur la fréquence (*Vectorizer*)
- Basées sur l'apprentissage (*Word Embedding*)

Vectorisation et Word Embedding

Vectorisation

One-Hot-Encoder

”la langue française a des règles de grammaire et de conjugaison compliquées”

One-Hot-Encoder

"la	langue	française	a	des	règles	de	grammaire	et	de	conjugaison	compliquées"

a
conjugaison
compliquées
de
des
et
française
grammaire
la
langue
règles

$V = 11$, Taille du dictionnaire

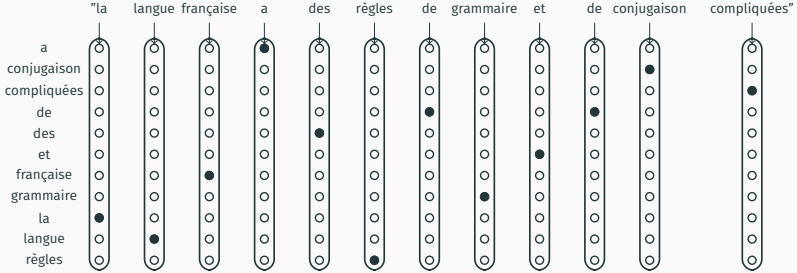
One-Hot-Encoder

$V = 11$, Taille du dictionnaire

One-Hot-Encoder

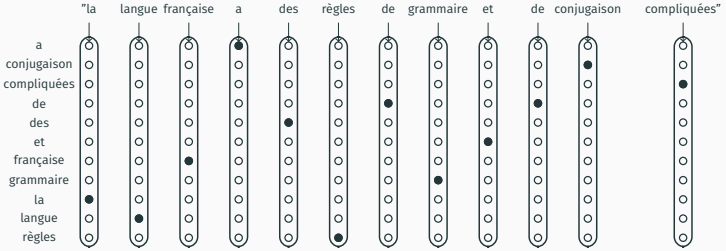
$V = 11$, Taille du dictionnaire

One-Hot-Encoder



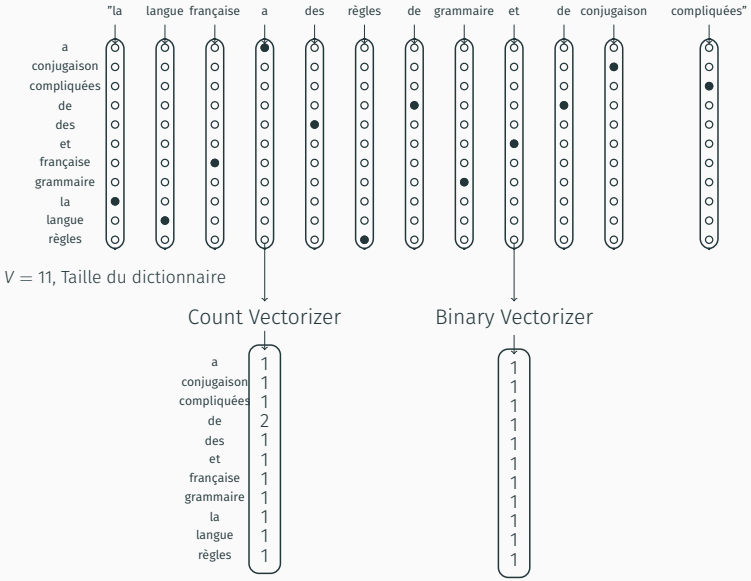
$V = 11$, Taille du dictionnaire

Count & Binary Vectorizer



$V = 11$, Taille du dictionnaire

Count & Binary Vectorizer



TF-IDF

Assigner un **score d'importance** d'un mot, ou d'une association de mots, dans un document **relativement à un ensemble de document**.

- t : mot ou association de mot.
- d : un document.
- D : un ensemble de document.

Definition (Formule Générale TF-IDF)

$$tfidf(t, d) = tf(t, d) \times idf(t, D)$$

- $tf(t, d)$: *Term-Frequency* Nombre d'occurrence du terme t dans le document d.
- $idf(t, D)$: *Inverse-Document-Frequency* Mesure l'importance du terme t dans l'ensemble des documents D.

Le terme $tf(t, d)$ est généralement défini comme le nombre d'occurrences du terme t dans le document d :

$$tf(t, d) = f_{t,d}$$

Cette définition est celle utilisée dans les librairies *scikit-learn* de *Python* et *Mllib* de *Spark*.

Cependant des variations peuvent exister :

binaire	$0, 1$
normalisation logarithmique	$1 + log(f_{t,d})$
normalisation « 0.5 » par le max	$0.5 + 0.5 \times \frac{f_{t,d}}{\max_{t' \in d} f_{t',d}}$
normalisation par le max	$K + (1 - K) \times \frac{f_{t,d}}{\max_{t' \in d} f_{t',d}}$

Table 2 – Variante du TF (source wikipedia)

Formule de l'IDF

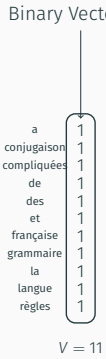
La formule du terme *IDF* varie également d'une implémentation à l'autre.

$log(\frac{N_D}{DF(t,D)})$	
$log(\frac{N_D+1}{DF(t,D)+1})$	*MlLib (Spark)*
$log(\frac{N_D+1}{DF(t,D)+1}) + 1$	*scikit-learn (Python)*

Table 3 – Variante de l'IDF

- N_D : Nombre de documents.
- $DF(t, D)$: Nombre de documents dans lequel le terme t apparaît.

Problème de Dimension



$V = 11$

$V = 10.000$

- Vecteur très creux
- Explosion rapide de la dimension
- Préparation au préalable du dictionnaires. (2 lectures du jeu de données).

Vectoriser les descriptions tout en **réduisant** l'espace de stockage.

$$X \quad \Rightarrow \quad \phi$$

Vecteur de taille V du dictionnaire et inconnu à l'avance.

Vecteurs de tailles fixes.
Taille fixé à l'avance : **n_hash**.

- Une seule passe sur les données pour construire le vecteur (Fonction deterministe).
- Produit vectoriel non biaisé : $\mathbb{E}[\langle \varphi(x), \varphi(x') \rangle] = \langle x, x' \rangle$.

Definition (Fonction de hashage - 1)

$$h : \mathbb{N} \to \{1, \dots, n\_hash\}$$
$$i \mapsto j = h(i).$$

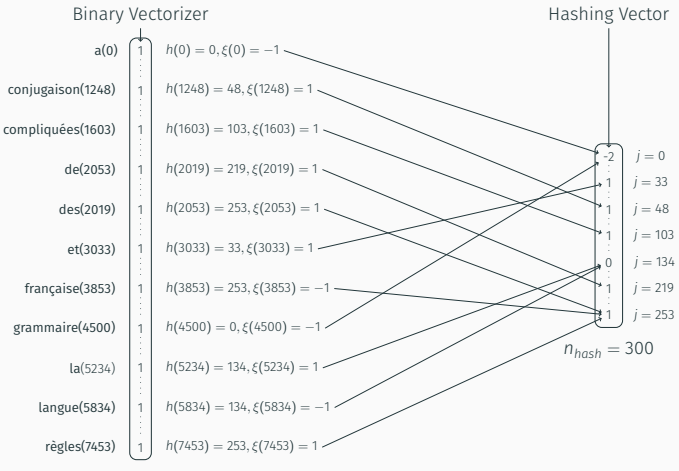
Definition (Fonction de hashage - 2)

$$\xi : \mathbb{N} \to \{1, -1\}$$
$$i \mapsto j = \xi(i).$$

Definition (Hashed Feature Map)

$$\phi_j^{\xi,h}(x) = \sum_{i \ s.t. h(i)=j} \xi(i) x_i$$

Hashage [Weinberger et al., 2009]



- Les fonctions de Hashage puis de TF-IDF sont appliquées sur le jeu de données d'apprentissage.
- La même fonction de Hashage est utilisée sur le jeu de données test.
- Les termes de TF entre un mot *t* et un document *d* sont recalculés pour le jeu de données test
- Les termes d'IDF calculés lors de l'apprentissage sont réutilisés.

Problématique :

Certains mots n'ont pas le même sens en fonction du contexte.

- *Short de bain* $\neq$ *Short* $\neq$ *bain*

Solution :

On ne considère pas que les mots uniques (*unigram*) mais les couples de mots (*bigram*), ou toutes associations de *n* mots (n-gram).

- Résout les ambiguïté du langage.
- Explosion de la taille des vecteurs
  - Exemple sur 21.543 lignes de la catégorie "TELEPHONIE-GPS"
  - 8.384 *unigram*, 50.012 *bigram*, 90.854 *trigram*...

Vectorisation et Word Embedding

Word Embedding

Problème de la représentation précédente :

Pas de relation entre les mots.

Représentation souhaitée :

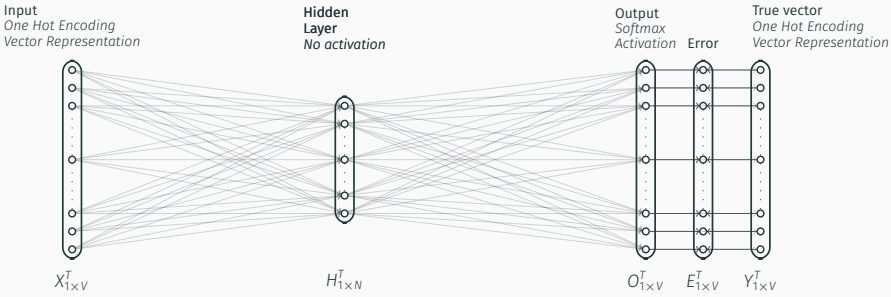
	Man	Woman	King	Queen	Apple	Orange
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

$\Rightarrow$ Comment construire cette représentation ?

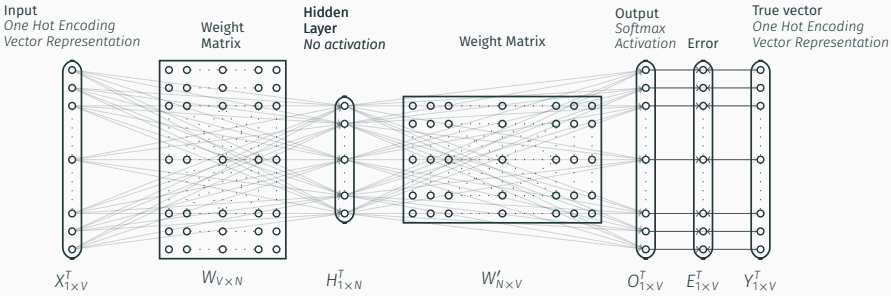
Youtube : Recurrent Neural Networks (RNNs) by Andrew NG [Full Course] Playlist

"la $\underbrace{\textit{langue}}_{\text{context}}$ $\underbrace{\textit{française}}_{\text{target}}$ $\underbrace{\textit{a}}_{\text{context}}$ *des règles de grammaire compliquées"*, $\textit{Window} = 1$

"la $\underbrace{\textit{langue}}_{\text{context}}$ $\underbrace{\textit{française}}_{\text{target}}$ $\underbrace{\textit{a}}_{\text{context}}$ *des règles de grammaire compliquées",* $\textit{Window} = 1$

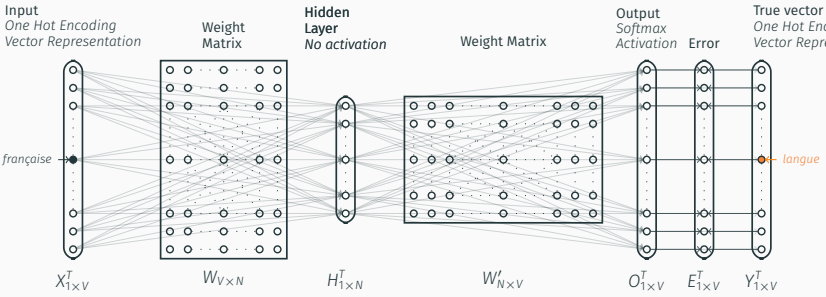


"la $\underbrace{\textit{langue}}_{\text{context}}$ $\underbrace{\textit{française}}_{\text{target}}$ $\underbrace{\textit{a}}_{\text{context}}$ *des règles de grammaire compliquées",* $\textit{Window} = 1$



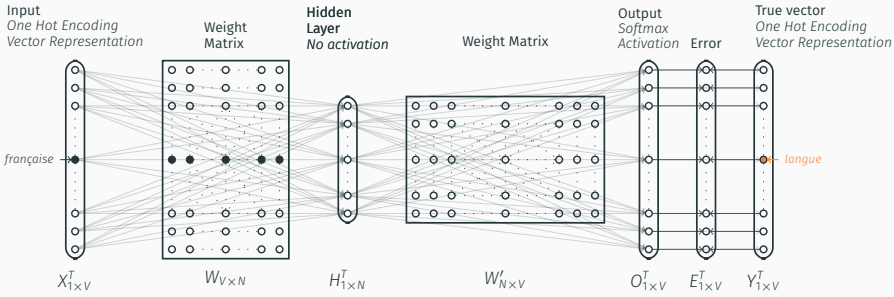
"la *langue* *française* *a* *des règles de grammaire compliquées"*, *Window* = 1

context target context



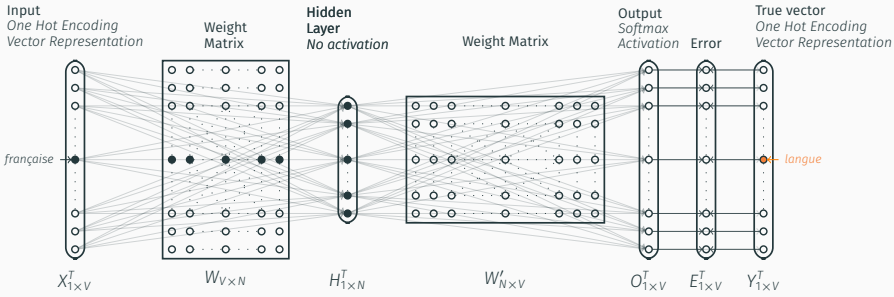
"la *langue* *française* *a* *des règles de grammaire compliquées"*, *Window* = 1

context target context



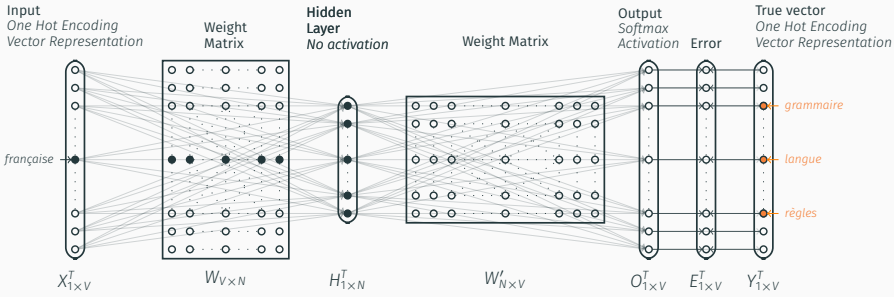
"la *langue* *française* *a* *des règles de grammaire compliquées"*, *Window* = 1

context target context



"la langue *française* *a des règles de grammaire* *compliquées"*, *Window* = 5

context target context

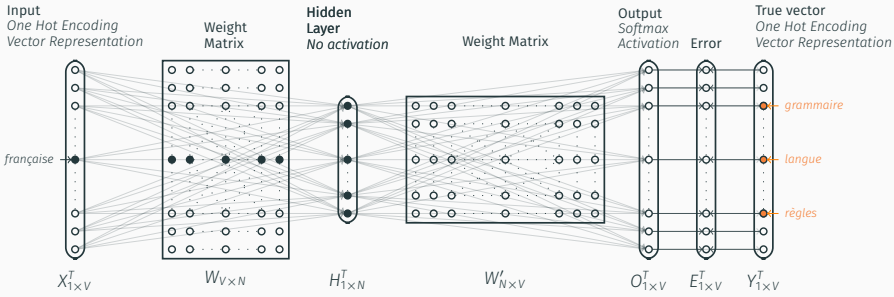


"la langue *française* *a des règles de grammaire* *compliquées"*, *Window* = 5

context target context

"la langue *allemande* *a des règles de grammaire* *compliquées"*, *Window* = 5

context target context

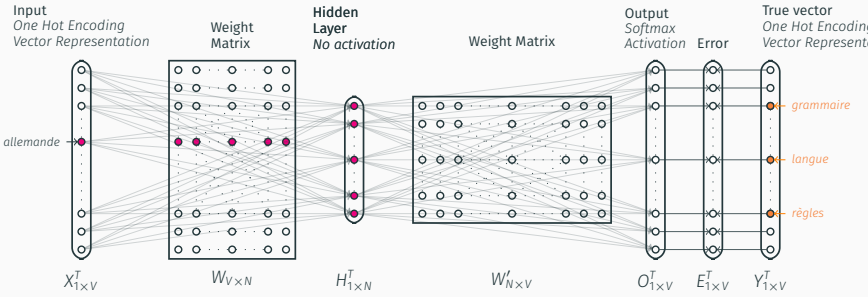


”la langue *française* *a des règles de grammaire* *compliquées”*, *Window* = 5

context target context

”la langue *allemande* *a des règles de grammaire* *compliquées”*, *Window* = 5

context target context

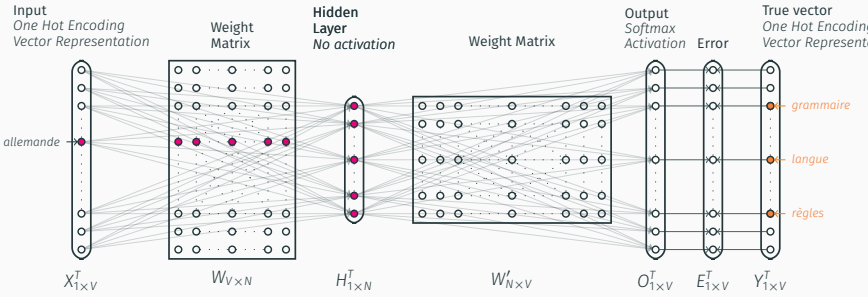


”la langue *française* *a des règles de grammaire* *compliquées”*, *Window* = 5

context target context

”la langue *allemande* *a des règles de grammaire* *compliquées”*, *Window* = 5

context target context



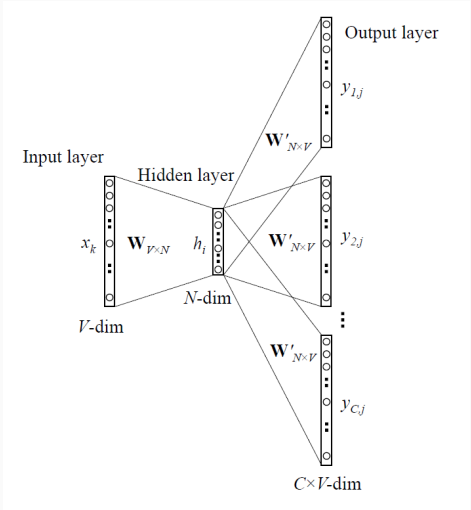
- Pas de fonction d'activation (ou activation linéaire) sur la couche cachée.
- La fonction de *loss* est la log-vraissemblance d'un mot sachant son contexte : $E = -log(p(Y_j/X_i))$ avec :

$$p(Y_j/X_i) = \frac{exp(W_{i,:} \cdot W'_{:,j})}{\sum_{k=1}^{V} exp(W_{k,:} \cdot W'_{:,j})}$$

- 2 Versions
  - Continuous Bag of Word (CBOW)
  - Skip-gram

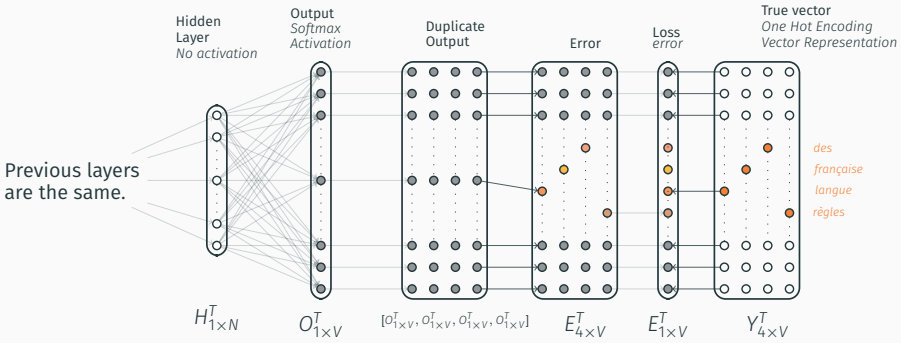
Word2Vec - Skip-Gram

input	output
la	langue, française
langue	la, française, a
français	la, langue, a, des e
a	langue, française, des, règles
des	française, a, règles, de
règles	a, des, de, grammaire
de	des, règles, grammaire, compliquées
grammaire	règles, de, compliquées
compliquées	de, grammaire



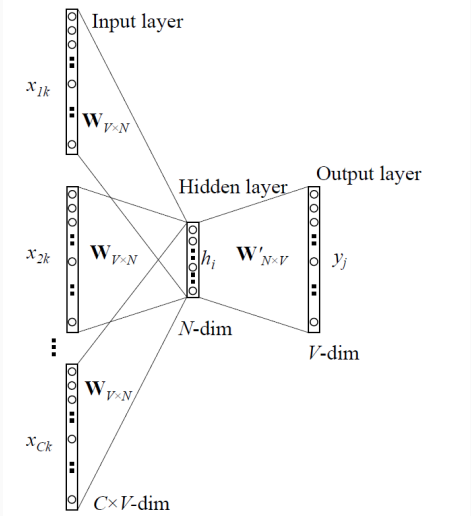
N = Dimensions dans laquelle les vecteurs vont être représentés et le nombre de neurones sur la couche caché.

Word2Vec - Skip-Gram



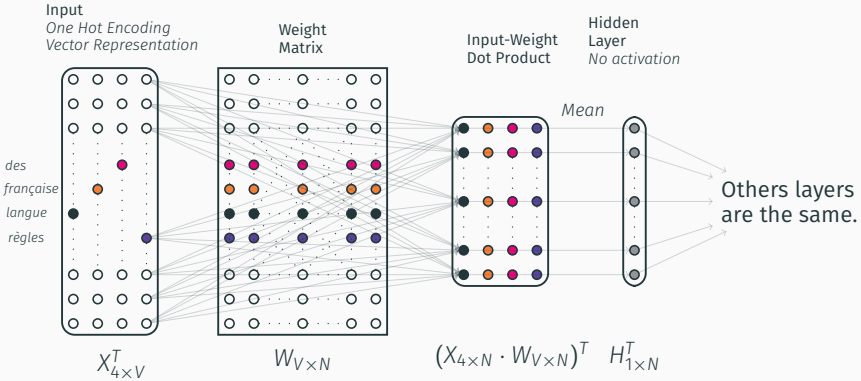
Word2Vec - Continuous Bag of Words (CBOW)

input	output
langue, française	la
la, française, a	langue
la, langue, a, des	française
langue, française, des, règles	a
française, a, règles, de	des
a, des, de, grammaire	règles
des, règles, grammaire, compliquées	de
règles, de, compliquées	grammaire
de, grammaire	compliquées



N = Dimensions dans laquelle les vecteurs vont être représentés et le nombre de neurones sur la couche caché.

Word2Vec - Continuous Bag of Words (CBOW)



- Fonction d'activation par défaut : **Softmax**

$$p(Y_j/X_i) = \frac{exp(W_{i,:} \cdot W'_{:,j})}{\sum_{k=1}^{V} exp(W_{k,:} \cdot W'_{:,j})}$$

Chaque neurones est mis à jour à chaque itération.

- Fonction d'activation avec le **negative sampling** :

$$p(T = 1, /Y_j, X_i) = \frac{1}{1 - exp(W_{i,:} \cdot W'_{:,j})}$$

Nombre limité de neurones mis à jour à chaque itération.

input	output	target
française	langue	1
française	mobylette	0
française	caramel	0
française	pudding	0
française	bateau	0

e	Man	Woman	King	Queen	Apple	Orange
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

$$e_{king} - e_{man} + e_{woman} = e_{pred} \approx e_{queen}$$

$$\begin{pmatrix} -0.95 \\ 0.93 \\ 0.7 \\ 0.02 \end{pmatrix} - \begin{pmatrix} -1 \\ 0.01 \\ 0.03 \\ 0.04 \end{pmatrix} + \begin{pmatrix} 1 \\ 0.02 \\ 0.02 \\ 0.01 \end{pmatrix} = \begin{pmatrix} 1.05 \\ 0.94 \\ 0.69 \\ -0.01 \end{pmatrix} \approx \begin{pmatrix} 0.97 \\ 0.95 \\ 0.69 \\ 0.01 \end{pmatrix}$$

Other Embeddings

- Glove
- FastText

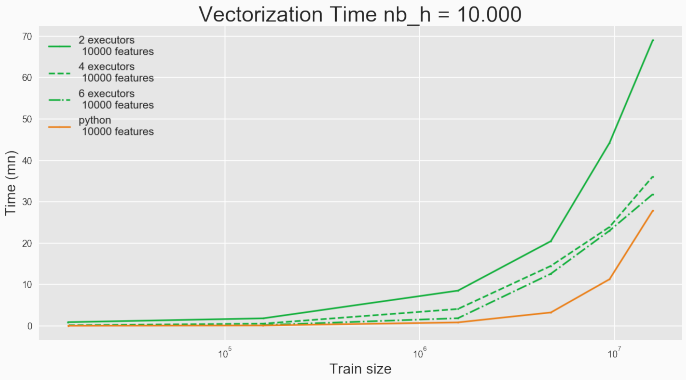
Fonctions de *hash* et de *tf-idf* séparées.

```
def vectorizer_train(df, columns=['Description', 'Libelle', 'Marque'], nb_hash=10000,
stop_words=None):

    #HASH
    # La fonction de FeatureHasher prend en compte le nombre d'apparition de chaque
    #mot.
    df_text = map(lambda x : collections.Counter(" ".join(x).split(" ")),
    df[columns].values)
    feathash = FeatureHasher(nb_hash)
    data_hash = feathash.fit_transform(map(collections.Counter,df_text))

    # TFIDF
    vec = TfidfVectorizer(min_df = 1,stop_words = stop_words,smooth_idf=True,
    norm='l2', sublinear_tf=True,use_idf=True, ngram_range=(1,2)) #bi-grams
    tfidf = vec.fit_transform(data_hash)
    return vec, feathash, tfidf



def apply_vectorizer(df, vec, columns =['Description', 'Libelle', 'Marque'],
    feathash):

    df_text = map(lambda x : collections.Counter(" ".join(x).split(" ")),
    df[columns].values)
    data_hash = feathash.transform(df_text)
```

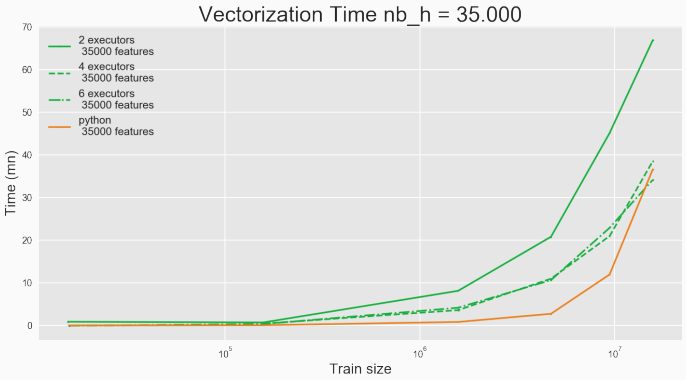
Fonctions de *hash* et de *tf* sont combinées dans un Tranformer, et la fonction *idf* dans un second.

```
# Term Frequency
hashing_tf = HashingTF(inputCol="cleanDescr", outputCol='tf', numFeatures=10000)
trainTfDF = hashing_tf.transform(trainDF)

# Inverse Document Frequency
idf = IDF(inputCol=hashing_tf.getOutputCol(), outputCol="tfidf")
idf_model = idf.fit(trainTfDF)
trainTfIdfDF = idf_model.transform(trainTfDF)

# application à l'échantillon test
testTfDF = hashing_tf.transform(testDF)
testTfIdfDF = idf_model.transform(testTfDF)
```

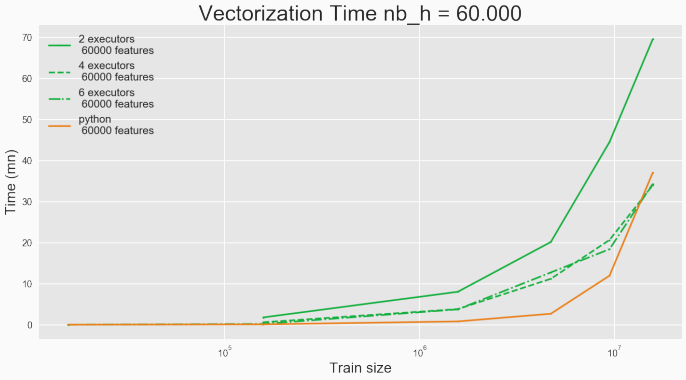
TP - Résultat - Temps Vectorization



TP - Résultat - Temps Vectorization



TP - Résultat - Temps Vectorization



Apprentissage

- Par défaut, *scikit-learn*, *MlLib* utilisent des algorithmes "*one-vs-all*"
- Dans *scikit-learn* différents solver sont disponible (*liblinear, LBFGS, sag...*).

- Les 4 meilleures solutions ont utilisées des méthodes linéaires Goutorbe et al. [2016]
- Soltutions à pluisieurs niveaux.
  - *On classifie parmi les 1ère catégories..*
  - *...puis parmi les catégories de niveau 4*
- Implémentation de la solution N°2 disponible : https ://github.com/ngaude/cdiscount

TP - Pipeline - Spark

Possibilité de combiner les étapes dans un Pipeline.

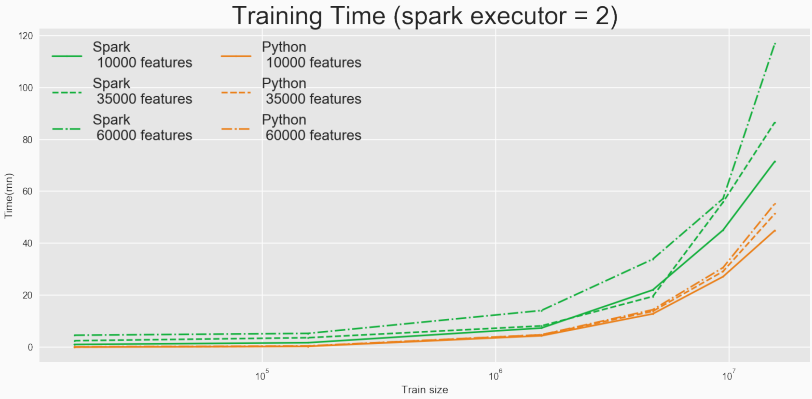
```
# Regex + Tokenizer
regexTokenizer = RegexTokenizer(inputCol="description", outputCol="tokenizedDescr",
pattern="[^a-z_]", minTokenLength=3, gaps=True)

# StopWord
remover = StopWordsRemover(inputCol="tokenizedDescr", outputCol="stopTokenizedDescr",
stopWords = list(STOPWORDS))

# Stemmer
stemmer = MyNltkStemmer(inputCol="stopTokenizedDescr", outputCol="cleanDescr")

# Indexer
indexer = StringIndexer(inputCol="categorie1", outputCol="categoryIndex")

# Hasing
hashing_tf = HashingTF(inputCol="cleanDescr", outputCol='tf', numFeatures=10000)

# Inverse Document Frequency
idf = IDF(inputCol=hashing_tf.getOutputCol(), outputCol="tfidf")

#Logistic Regression
lr = LogisticRegression(maxIter=100, regParam=0.01, fitIntercept=False,
family = "multinomial", tol=0.0001,elasticNetParam=0.0,
featuresCol="tfidf", labelCol="categoryIndex")

# Creation du pipeline
```

Définition d'un Transformer personnalisé.

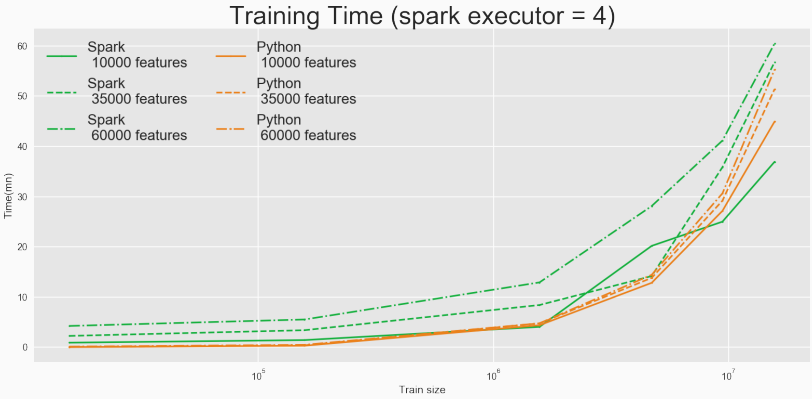
```
class MyNltkStemmer(Transformer, HasInputCol, HasOutputCol):

    @keyword_only
    def __init__(self, inputCol=None, outputCol=None):
        super(MyNltkStemmer, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

    @keyword_only
    def setParams(self, inputCol=None, outputCol=None):
        kwargs = self._input_kwargs
        return self._set(**kwargs)

    def _transform(self, dataset):
        STEMMER = nltk.stem.SnowballStemmer('french')
        def clean_text(tokens):
            tokens_stem = [ STEMMER.stem(token) for token in tokens]
            return tokens_stem
        udfCleanText =  udf(lambda lt : clean_text(lt), ArrayType(StringType()))
        out_col = self.getOutputCol()
        in_col = dataset[self.getInputCol()]
        return dataset.withColumn(out_col, udfCleanText(in_col))
```

TP - Résultat - Erreur Prédiction

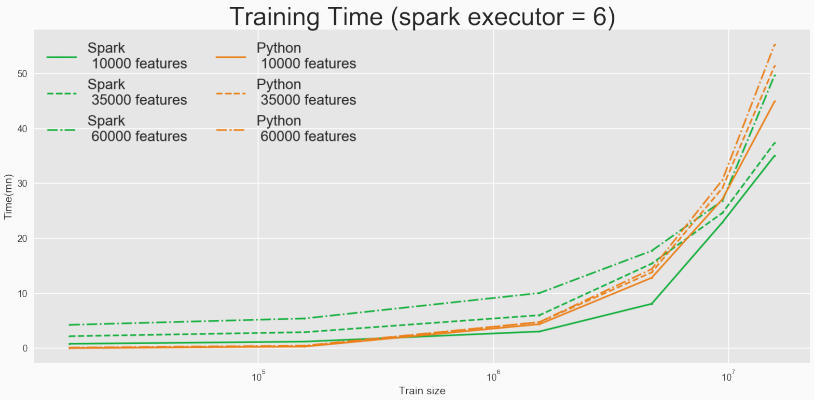
TP - Résultat - Temps Vectorization



TP - Résultat - Temps Vectorization



TP - Résultat - Temps Vectorization

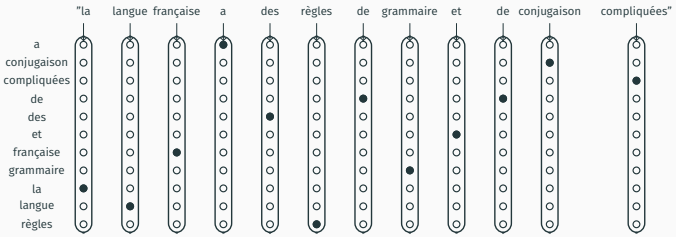


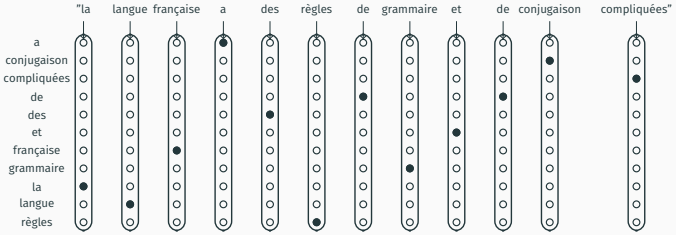
Réseau de neurones récurrents

De nombreux problèmes de NLP nécessitent de représenter les données sous forme de séquences.

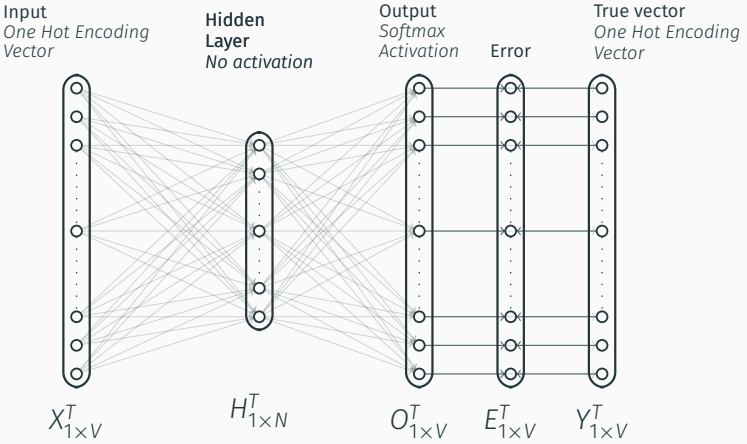
type	x	y
Generation de texte	empty,scalar $\emptyset, 1, 2$	text sequence *Ceci est généré automatiquement*
Classification de sentiment	text sequence *C'était bien mais pas top*	int 2(/5)
Traduction automatique	text sequence *Comment tu vas ?*	text sequence *How are you ?*
Reconnaissance d'entité	text sequence *Harry Potter est un sorcier*	scalar vector $[1, 1, 0, 0, 0]$

Motivations



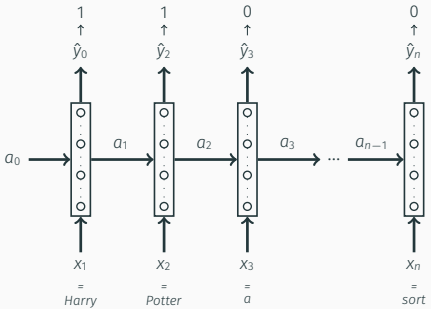


Les réseaux de neurones ”classiques” ne sont pas adaptés.



- Taille des séquences différente d'un exemple à l'autre
- Information entre variable (leur position) n'est pas partagé

RNN - Notations - Reconnaissance d'entité



- $x_t : 1 \times V$, (OHE representation)
- V : Taille du dictionnaire
- $y_t : 1 \times 1, (0/1)$
- $a_t : 1 \times H$
- H : Nombre de neurones
- N : Nombre de pas de temps (Timestep) ($\neq$ Nombre de couches !)

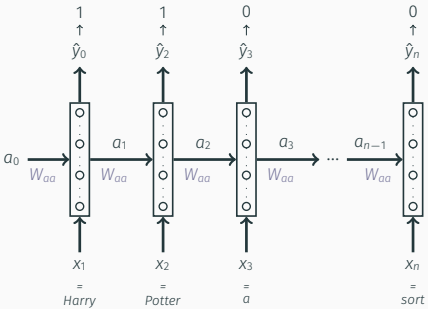
Fonctions d'activation

$$a_t = g_a(a_{t-1} \quad + x_t \quad + b_a)$$

$$\hat{y}_t = g_y(a_t \quad + b_y)$$

- g_a : Fonction d'activation : tanh/ReLu
- g_y : Fonction d'activation : Sigmoïd

RNN - Notations - Reconnaissance d'entité



- $x_t : 1 \times V$, (OHE representation)
- V : Taille du dictionnaire
- $y_t : 1 \times 1, (0/1)$
- $a_t : 1 \times H$
- H : Nombre de neurones
- N : Nombre de pas de temps (Timestep) ($\neq$ Nombre de couches !)

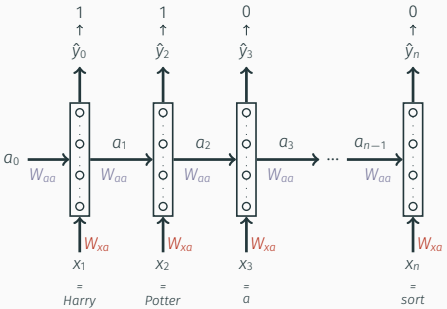
Fonctions d'activation

$$a_t = g_a(a_{t-1}W_{aa} + x_t \quad + b_a)$$

$$\hat{y}_t = g_y(a_t \quad + b_y)$$

- g_a : Fonction d'activation : tanh/ReLu
- g_y : Fonction d'activation : Sigmoïd
- $W_{aa} : H \times H, \forall t \in [1, \cdots, n]$

RNN - Notations - Reconnaissance d'entité



- $x_t : 1 \times V$, (OHE representation)
- V : Taille du dictionnaire
- $y_t : 1 \times 1, (0/1)$
- $a_t : 1 \times H$
- H : Nombre de neurones
- N : Nombre de pas de temps (Timestep) ($\neq$ Nombre de couches!)

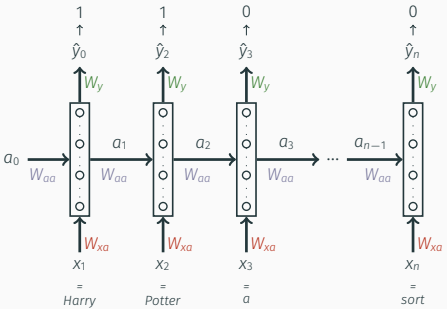
Fonctions d'activation

$$a_t = g_a(a_{t-1}W_{aa} + x_tW_{xa} + b_a)$$

$$\hat{y}_t = g_y(a_t \quad + b_y)$$

- g_a : Fonction d'activation : tanh/ReLu
- g_y : Fonction d'activation : Sigmoïd
- $W_{aa} : H \times H, \forall t \in [1, \cdots, n]$
- $W_{xa} : V \times H, \forall t \in [1, \cdots, n]$

RNN - Notations - Reconnaissance d'entité



- $x_t : 1 \times V$, (OHE representation)
- V : Taille du dictionnaire
- $y_t : 1 \times 1, (0/1)$
- $a_t : 1 \times H$
- H : Nombre de neurones
- N : Nombre de pas de temps (Timestep) ($\neq$ Nombre de couches !)

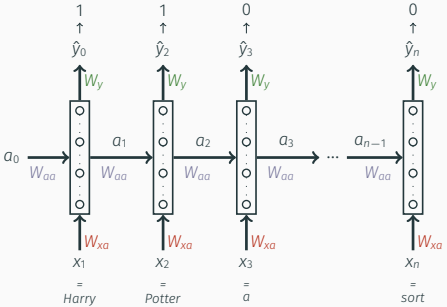
Fonctions d'activation

$$a_t = g_a(a_{t-1}W_{aa} + x_tW_{xa} + b_a)$$

$$\hat{y}_t = g_y(a_tW_y + b_y)$$

- g_a : Fonction d'activation : **tanh/ReLu**
- g_y : Fonction d'activation : **Sigmoïd**
- $W_{aa} : H \times H, \forall t \in [1, \cdots, n]$
- $W_{xa} : V \times H, \forall t \in [1, \cdots, n]$
- $W_y : H \times 1, \forall t \in [1, \cdots, n]$

RNN - Notations - Reconnaissance d'entité



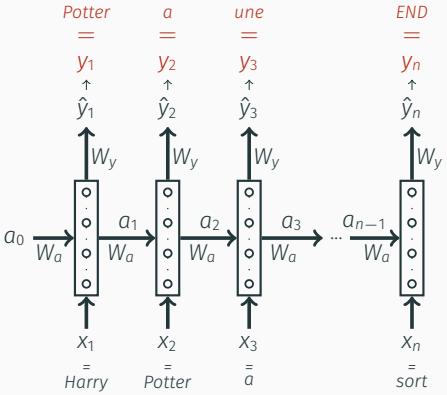
- $x_t : 1 \times V$, (OHE representation)
- V : Taille du dictionnaire
- $y_t : 1 \times 1, (0/1)$
- $a_t : 1 \times H$
- H : Nombre de neurones
- N : Nombre de pas de temps (Timestep) ($\neq$ Nombre de couches !)

Fonctions d'activation

$$
\begin{aligned}
a_t &= g_a(a_{t-1}W_{aa} + x_tW_{xa} + b_a) \\
&= g_a\left([a_{t-1}, x_t]\right)\begin{bmatrix} W_{aa} \\ W_{wa} \end{bmatrix} + b_a\Big) \\
&= g_a([a_{t-1}, x_t]W_a + b_a) \\
\hat{y}_t &= g_y(a_tW_y + b_y)
\end{aligned}
$$

- g_a : Fonction d'activation : tanh/ReLu
- g_y : Fonction d'activation : Sigmoïd
- $W_{aa} : H \times H, \forall t \in [1, \cdots, n]$
- $W_{xa} : V \times H, \forall t \in [1, \cdots, n]$
- $W_y : H \times 1, \forall t \in [1, \cdots, n]$
- $W_a : (H + V) \times H, \forall t \in [1, \cdots, n]$

RNN - Notations - Génération de texte (*Apprentissage*)

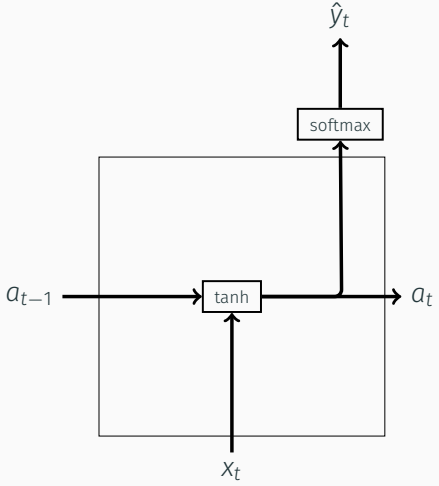


Fonctions d'activation

$$a_t = g([a_{t-1}, x_t]W_a + b_a)$$
$$y_t = g_y(a_t W_y + b_y)$$

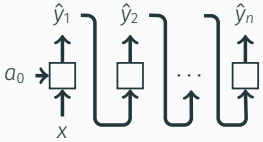
- $x_t : 1 \times V$, (OHE representation)
- V : Taille du dictionnaire
- $\hat{y}_t, y_t : 1 \times V$, (OHE representation)
- $a_t : 1 \times H$
- H : Nombre de neurones
- N : Nombre de pas de temps (Timestep) ($\neq$ Nombre de couches !)
- g_a : Fonction d'activation : tanh/ReLu
- g_a : Fonction d'activation : Softmax
- $W_y : H \times V, \forall t \in [1, \cdots, n]$
- $W_a : (H + V) \times H, \forall t \in [1, \cdots, n]$

RNN Unit



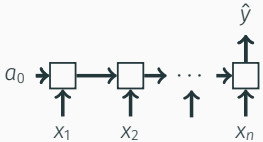
$$a_t = tanh([a_{t-1}, x_t]W_a + b_a)$$
$$\hat{y}_t = softmax(a_t W_y + b_y)$$

Different type of RNN



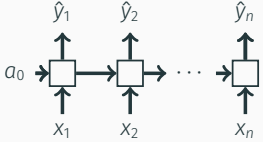
One to many

Génération de texte



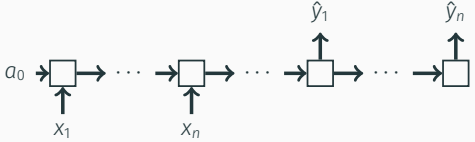
Many to one

Sentiment classification



Many to many

Reconnaissance d'entité

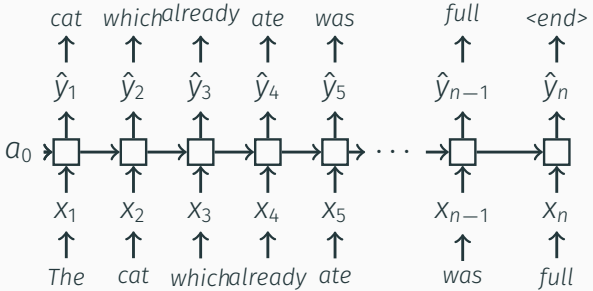


Many to Many

Traduction automatique

Problème : Dépendence à long terme :

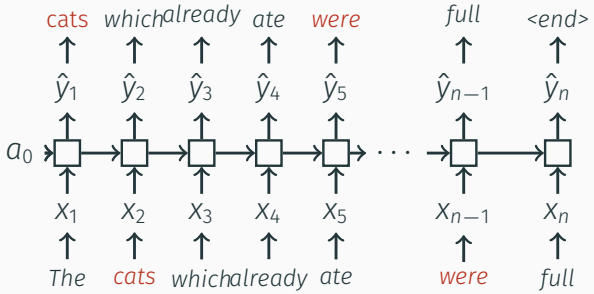
The cat, which already ate ..., was full.



Problème : Dépendence à long terme :

The cat, which already ate ..., was full.

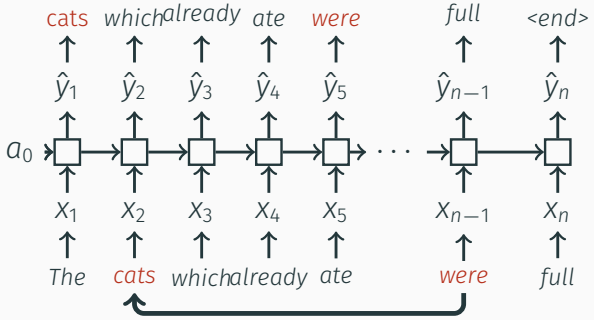
The cats, which already ate ..., were full.



Problème : Dépendence à long terme :

The cat, which already ate ..., was full.

The cats, which already ate ..., were full.

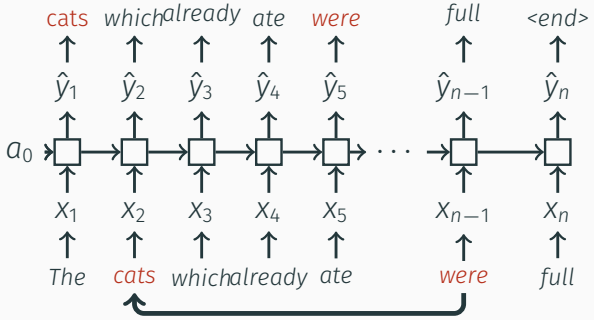


Difficile de retropoprager le gradient

Problème : Dépendence à long terme :

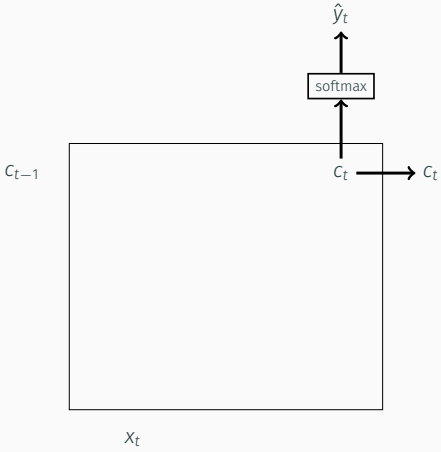
The cat, which already ate ..., was full.

The cats, which already ate ..., were full.



Solution : Inclure des cellules *mémoire*.

GRU Unit - Cho et al. [2014], Chung et al. [2014]

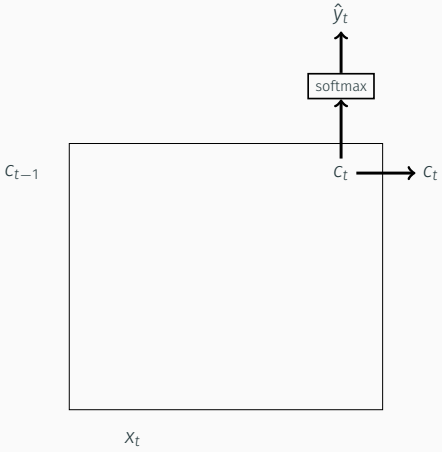


c_t = memory cell

$c_t = a_t$

$\hat{y}_t = softmax(c_t W_y + b_y)$

GRU Unit - Cho et al. [2014], Chung et al. [2014]



c_t = memory cell

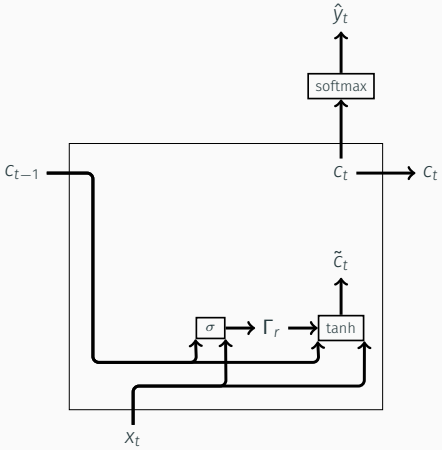
$c_t = a_t$

$\Gamma_r = \sigma([c_{t-1}, x_t]W_r + b_r)$

$\tilde{c_t} = tanh([\Gamma_r * c_{t-1}, x_t]W_c + b_c)$

$\hat{y}_t = softmax(c_t W_y + b_y)$

GRU Unit - Cho et al. [2014], Chung et al. [2014]



c_t = memory cell

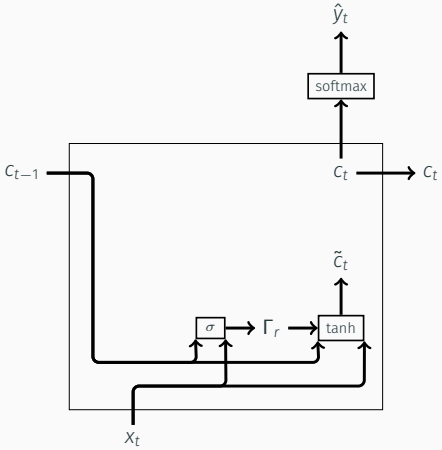
$c_t = a_t$

$\Gamma_r = \sigma([c_{t-1}, x_t]W_r + b_r)$

$\tilde{c_t} = tanh([\Gamma_r * c_{t-1}, x_t]W_c + b_c)$

$\hat{y}_t = softmax(c_t W_y + b_y)$

GRU Unit - Cho et al. [2014], Chung et al. [2014]



$$c_t = \text{memory cell}$$

$$c_t = a_t$$

$$\Gamma_r = \sigma([c_{t-1}, x_t]W_r + b_r)$$

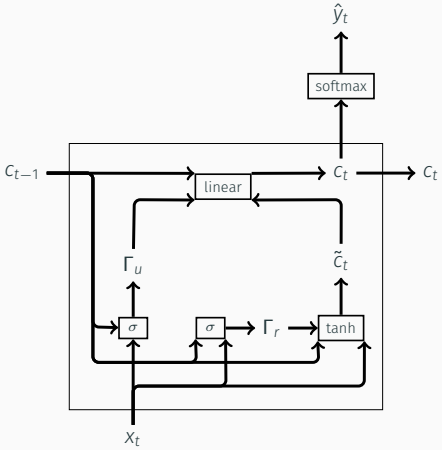
$$\tilde{c_t} = tanh([\Gamma_r * c_{t-1}, x_t]W_c + b_c)$$

$$\Gamma_u = \sigma([c_{t-1}, x_t]W_u + b_u)$$

$$c_t = \Gamma_u * \tilde{c_t} + (1 - \Gamma_u) * c_{t-1}$$

$$\hat{y}_t = softmax(c_t W_y + b_y)$$

GRU Unit - Cho et al. [2014], Chung et al. [2014]



$$c_t = \text{memory cell}$$

$$c_t = a_t$$

$$\Gamma_r = \sigma([c_{t-1}, x_t]W_r + b_r)$$

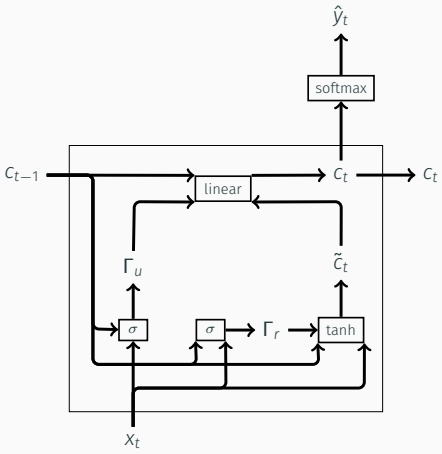
$$\tilde{c_t} = tanh([\Gamma_r * c_{t-1}, x_t]W_c + b_c)$$

$$\Gamma_u = \sigma([c_{t-1}, x_t]W_u + b_u)$$

$$c_t = \Gamma_u * \tilde{c_t} + (1 - \Gamma_u) * c_{t-1}$$

$$\hat{y}_t = softmax(c_t W_y + b_y)$$

GRU Unit - Cho et al. [2014], Chung et al. [2014]



$$c_t = \text{memory cell}$$

$$c_t = a_t$$

$$\Gamma_r = \sigma([c_{t-1}, x_t]W_r + b_r)$$

$$\tilde{c}_t = tanh([\Gamma_r * c_{t-1}, x_t]W_c + b_c)$$

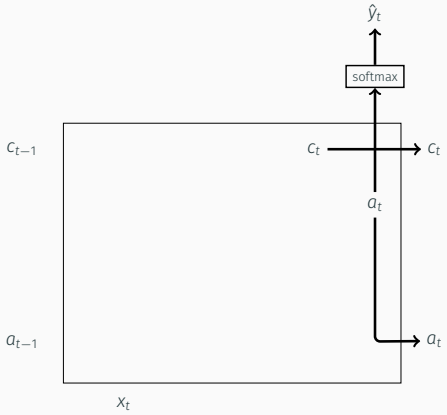
$$\Gamma_u = \sigma([c_{t-1}, x_t]W_u + b_u)$$

$$c_t = \Gamma_u * \tilde{c}_t + (1 - \Gamma_u) * c_{t-1}$$

$$\hat{y}_t = softmax(c_t W_y + b_y)$$

the	cats	which	already	ate	$\cdots$	were	full
$c_1^i = 0$	$c_2^i = 1$	$c_3^i = 1$	$c_4^i = 1$	$c_5^i = 1$	$\cdots$	$c_{n-1}^i = 1$	$c_n^i = 0$
$\Gamma_u^i = 0$	$\Gamma_u^i = 1$	$\Gamma_u^i = 0$	$\Gamma_u^i = 0$	$\Gamma_u^i = 0$	$\cdots$	$\Gamma_u^i = 1$	$\Gamma_u^i = 0$

LSTM Unit - Hochreiter and Schmidhuber [1997]



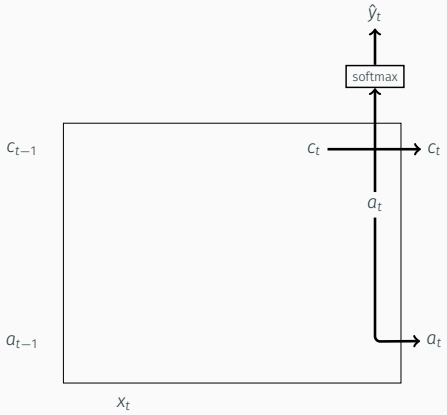
Cola's Blog

c_t = memory cell

$c_t \neq a_t$

$\hat{y}_t = softmax(a_t W_y + b_y)$

LSTM Unit - Hochreiter and Schmidhuber [1997]



Cola's Blog

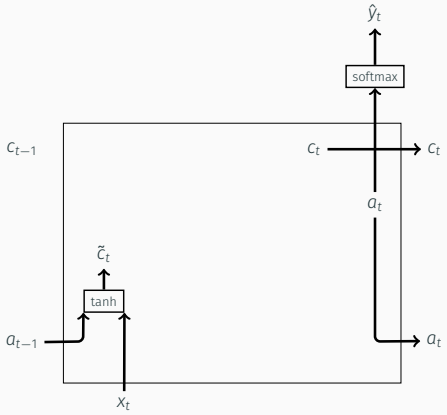
c_t = memory cell

$c_t \neq a_t$

$\tilde{c}_t = tanh([a_{t-1}, x_t]W_a + b_a)$

$\hat{y}_t = softmax(a_t W_y + b_y)$

LSTM Unit - Hochreiter and Schmidhuber [1997]



Cola's Blog

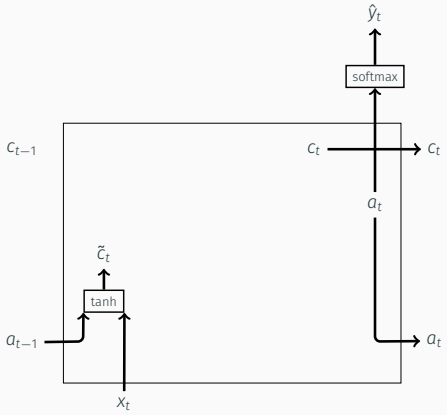
c_t = memory cell

$c_t \neq a_t$

$\tilde{c}_t = tanh([a_{t-1}, x_t]W_a + b_a)$

$\hat{y}_t = softmax(a_t W_y + b_y)$

LSTM Unit - Hochreiter and Schmidhuber [1997]



Cola's Blog

$c_t = \text{memory cell}$

$c_t \neq a_t$

$\tilde{c}_t = tanh([a_{t-1}, x_t]W_a + b_a)$

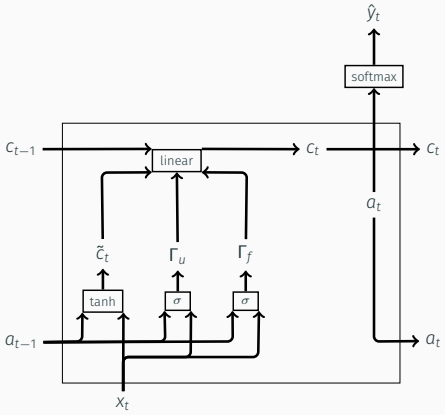
$\Gamma_u = \sigma([a_{t-1}, x_t]W_u + b_u)$

$\Gamma_f = \sigma([a_{t-1}, x_t]W_u + b_f)$

$c_t = \Gamma_u * \tilde{c}_t + \Gamma_f * c_{t-1}$

$\hat{y}_t = softmax(a_t W_y + b_y)$

LSTM Unit - Hochreiter and Schmidhuber [1997]



Cola's Blog

$c_t = \text{memory cell}$

$c_t \neq a_t$

$\tilde{c_t} = tanh([a_{t-1}, x_t]W_a + b_a)$

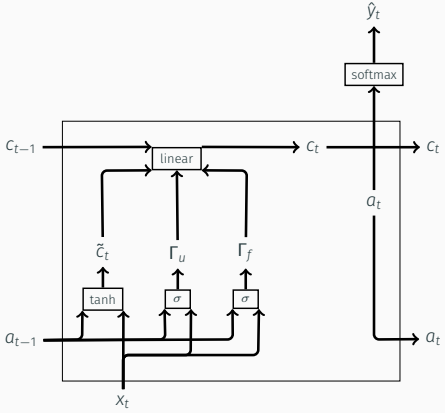
$\Gamma_u = \sigma([a_{t-1}, x_t]W_u + b_u)$

$\Gamma_f = \sigma([a_{t-1}, x_t]W_u + b_f)$

$c_t = \Gamma_u * \tilde{c_t} + \Gamma_f * c_{t-1}$

$\hat{y}_t = softmax(a_t W_y + b_y)$

LSTM Unit - Hochreiter and Schmidhuber [1997]



Cola's Blog

$c_t = \text{memory cell}$

$c_t \neq a_t$

$\tilde{c}_t = tanh([a_{t-1}, x_t]W_a + b_a)$

$\Gamma_u = \sigma([a_{t-1}, x_t]W_u + b_u)$

$\Gamma_f = \sigma([a_{t-1}, x_t]W_u + b_f)$

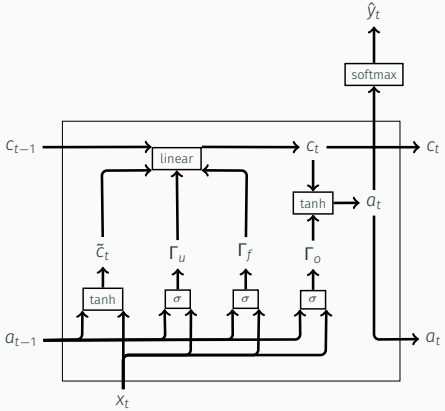
$c_t = \Gamma_u * \tilde{c}_t + \Gamma_f * c_{t-1}$

$\Gamma_o = \sigma([a_{t-1}, x_t]W_r + b_o)$

$a_t = \Gamma_o * tanh(c_t)$

$\hat{y}_t = softmax(a_t W_y + b_y)$

LSTM Unit - Hochreiter and Schmidhuber [1997]



Cola's Blog

$c_t = \text{memory cell}$

$c_t \neq a_t$

$\tilde{c_t} = tanh([a_{t-1}, x_t]W_a + b_a)$

$\Gamma_u = \sigma([a_{t-1}, x_t]W_u + b_u)$

$\Gamma_f = \sigma([a_{t-1}, x_t]W_u + b_f)$

$c_t = \Gamma_u * \tilde{c_t} + \Gamma_f * c_{t-1}$

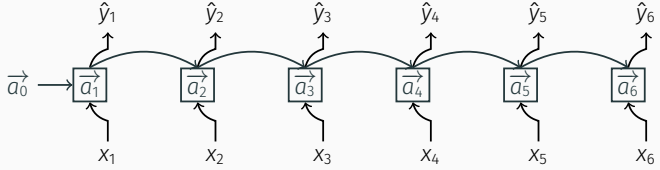
$\Gamma_o = \sigma([a_{t-1}, x_t]W_r + b_o)$

$a_t = \Gamma_o * tanh(c_t)$

$\hat{y}_t = softmax(a_t W_y + b_y)$

Bidirectionnal RNN (BRNN)

Problème : Fin de la phrase explique le début.
Exemple : Reconnaissance d'entité (Nom) :
Hollande était le président de la France.
Hollande est un pays d'Europe.



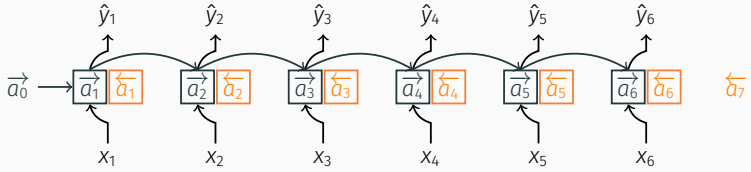
Bidirectionnal RNN (BRNN)

Problème : Fin de la phrase explique le début.

Exemple : Reconnaissance d'entité (Nom) :

Hollande était le président de la France.

Hollande est un pays d'Europe.



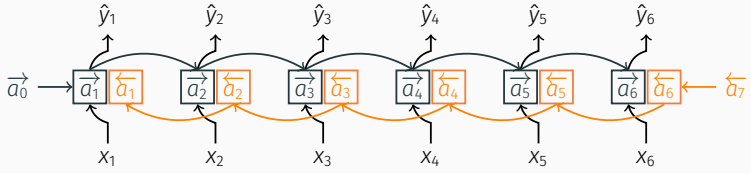
Bidirectionnal RNN (BRNN)

Problème : Fin de la phrase explique le début.

Exemple : Reconnaissance d'entité (Nom) :

Hollande était le président de la France.

Hollande est un pays d'Europe.



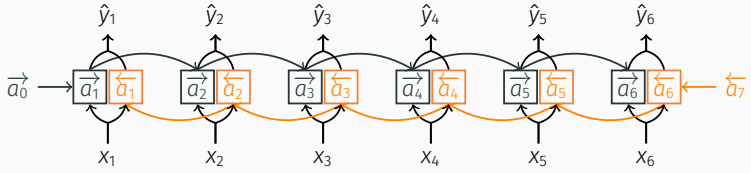
Bidirectionnal RNN (BRNN)

Problème : Fin de la phrase explique le début.

Exemple : Reconnaissance d'entité (Nom) :

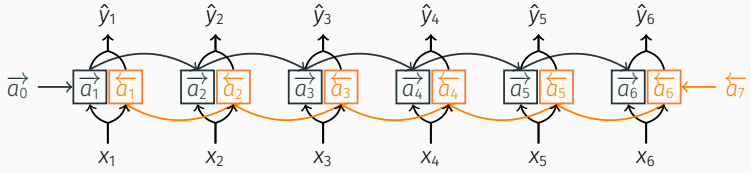
Hollande était le président de la France.

Hollande est un pays d'Europe.



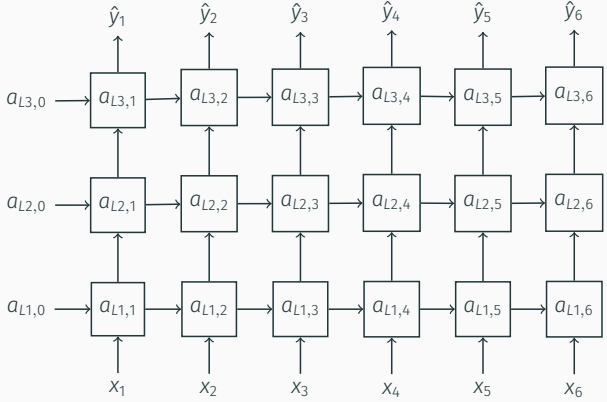
Bidirectionnal RNN (BRNN)

Problème : Fin de la phrase explique le début.
Exemple : Reconnaissance d'entité (Nom) :
Hollande était le président de la France.
Hollande est un pays d'Europe.



$$\overleftarrow{a_t} = tanh([\overleftarrow{a}_{t-1}, x_t]W_{\overleftarrow{a}} + b_{\overleftarrow{a}})$$
$$\overrightarrow{a_t} = tanh([\overrightarrow{a}_{t-1}, x_t]W_{\overrightarrow{a}} + b_{\overrightarrow{a}})$$
$$\hat{y}_t = softmax([\overrightarrow{a}_t, \overleftarrow{a}_t]W_y + b_y)$$

Deep RNN



$$a_{L1,t} = tanh([a_{L1,t-1}, x_t]W_{L1,a} + b_{L1,a})$$
$$a_{Li,t} = tanh([a_{Li,t-1}, a_{Li-1,t}]W_{Li,a} + b_{Li,a}), \forall i > 1$$
$$\hat{y}_t = softmax(a_{L3,t}W_y + b_y)$$

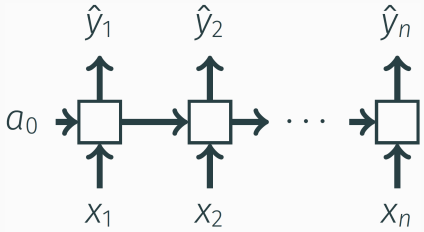
Exemple : Génération de texte

x = *Pour apple iphone 4 : coque bumper silicone blanc - Cet étui en silicone rigide...*

$x = [\ x_1, \quad x_2, \quad x_3, \quad x_4 \quad ..., \quad x_n\ ]$

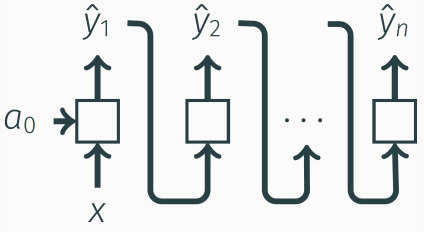
$x = [\ \mathit{start}, \quad \mathit{OHE}(P), \quad \mathit{OHE}(o), \quad \mathit{OHE}(u), \quad ..., \quad \mathit{END}\ ]$

$\mathit{Vocabulary} = [a, \cdots, z, A, \cdots, Z, 0, \cdots, 0, ., ?, ;, :, !, \cdots]$



Many to many

Apprentissage



One to many

Génération

TP

3 parties :

- Catégorisation : Nettoyage, Vectorisation et Apprentissage.
  - *Part1-1-AIF-PythonNltk-Explore&CleanText-Cdiscount.ipynb*
  - *Part1-2-AIF-PythonNltkGensim-FeatureExtraction-Cdiscount.ipynb*
  - *Part1-3-AIF-PythonScikitLearn-Prediction-Cdiscount.ipynb*
- Categorisation : Workflow complet Python/Pyspark.
  - *Part2-1-AIF-PythonWorkflow-Cdiscount.ipynb*
  - *Part2-2-AIF-PysparkWorkflow-Cdiscount.ipynb*
  - *Part2-2bis-AIF-PysparkWorkflowPipeline-Cdiscount.ipynb*
- RNN : Génération de texte.
  - *Part3-AIF-PythonKeras-TextGeneration-Cdiscou.ipynb*

Références

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation : Encoder-decoder approaches. *arXiv preprint arXiv :1409.1259*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv :1412.3555*, 2014.

References II

Bruno Goutorbe, Yang Jiao, Matthieu Cornec, Christelle Grauer, and Jérémie Jakubowicz. A large e-commerce data set released to benchmark categorization methods. 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv :1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009.