Présentation TIPE

Théodore Chapuis-Chkaiban

10 juin 2019

Natural Language Processing (NLP)

- Applications :
  - Traduction Automatique
  - Filtres à Spam
  - Inférence de sens / sentiments : prédire le comportement utilisateur
- Prolongement Lexical :
  - Word2Vec (2013-14)
  - Glove (2015-16)
  - FastText... (2017-19)

Traduction

Application à la traduction :

```
In [138]: traduction_mpmBA(isoBA,"le chat était très malade hier")
__main__:7: RuntimeWarning: invalid value encountered in double_scalars
Out[138]: ['the', 'cat', 'was', 'very', 'sick', 'yesterday']

In [139]: traduction_mpmAB(iso,"The cat was very sick yesterday")
__main__:7: RuntimeWarning: invalid value encountered in double_scalars
Out[139]: ['le', 'chat', 'fut', 'très', 'malade', 'hier']
```

FIGURE – Traduction d'un exemple simple

```
In [173]: traduction_mpmBA(isoBA, "la biologie est une science  particulière")
__main__:7: RuntimeWarning: invalid value encountered in double_scalars
Out[173]: ['the', 'biologie', 'is', 'one', 'science', 'particular']
```

FIGURE – Défauts de traduction

Traduction

Chercher un isomorphisme entre les espaces :

Definition

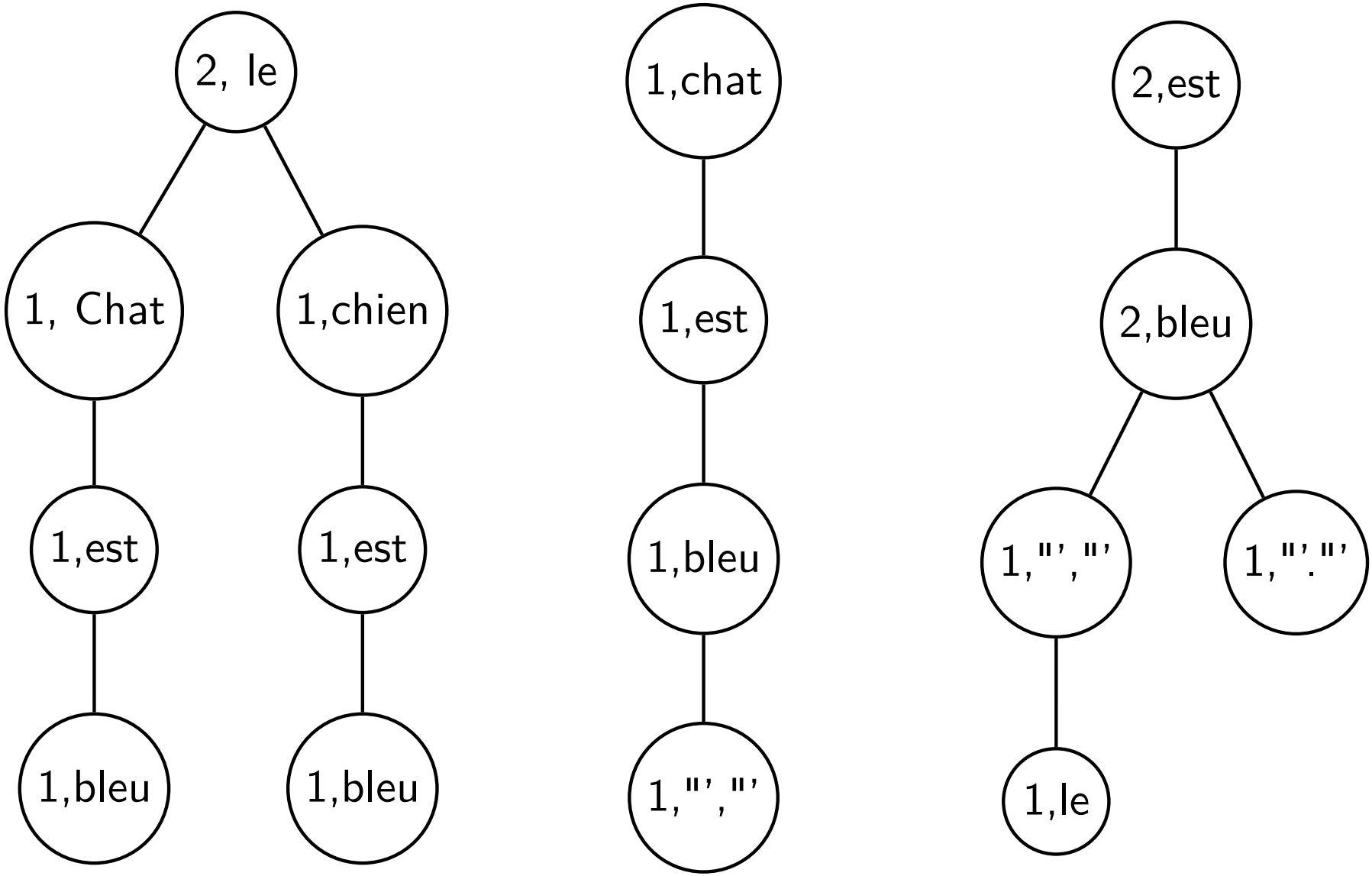
$$W^* = \underbrace{argmin}_{W \in O_d(\mathbb{R})} ||WX - Y||_F = UV^T \quad (1)$$

Avec X la matrices des vecteurs de la langue A et Y celle de la langue B

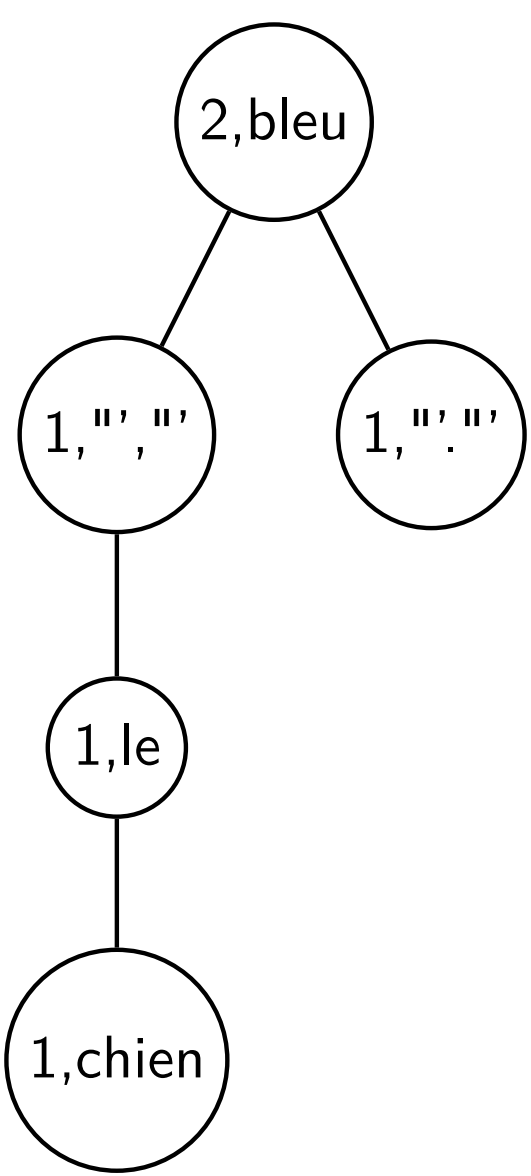
$$U\Sigma V^T = SVD(YX^T)$$

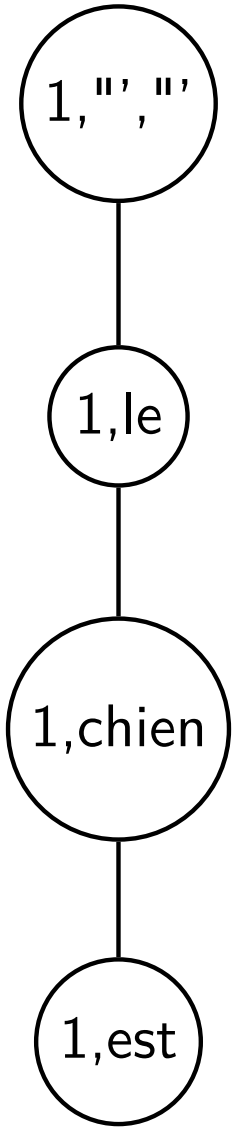
Problématique : *prendre en compte des phrases* : suites de mot de profondeur d.
Objectif $\rightarrow$ *traduction*

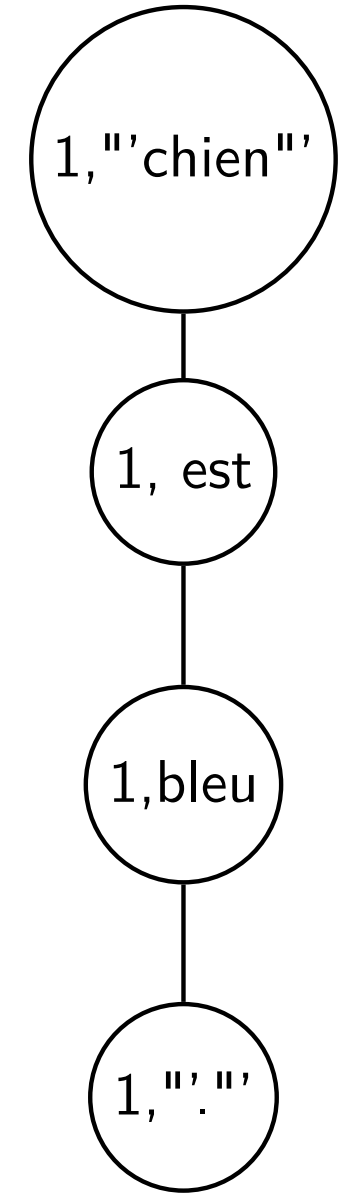
- Représenter les phrases par des arbres $\rightarrow$ optimiser l'espace
- S'arrêter aux points, prendre en compte la *ponctuation*.
- Nouvelle structure de donnée : *annuaires*



Exemple : "'Le chat est bleu, le chien est bleu."'







Exemple : "'‘Le chat est bleu, le chien est bleu."'

- Utiliser les avantages de Python : tables de hachage de type dictionnaire
- Utiliser l'aliasing pour plus de souplesse dans l'implémentation

Définir un annuaire :

Definition

Un annuaire a est défini récursivement en fonction de la profondeur d_i de ses entrées a_i :

- Si a_i est de profondeur 0, alors : $a_i = \mathrm{k}$, $k \in \mathbb{Q}$
- Si a_i est de profondeur d, alors : $a_i = (k, a')$ avec
  - $k \in \mathbb{Q}$
  - a' un annuaire dont les entrées profondeur au plus d-1

Utilisation des annuaires

En Python, les annuaire de profondeur d seront représentés par un dictionnaire D :
Les clefs D_i de ce dictionnaire sont des couples (k, D') avec k relatif positif et D' un dictionnaire
Inconvénient : Espace mémoire

Utilisation pour générer des phrases

```
In [16]: generer(4)
Out[16]: "you wouldn ' t be right . "

In [17]: generer(4)
Out[17]: '" orphan " , a person who has made a man of me indeed . '

In [18]: generer(4)
Out[18]: 'ashmead , instructed as above , dined again with the detective but out of revenge gave
him but one bottle of madeira . '
```

FIGURE – Génération de phrases aléatoires

Glove

- Apprentissage non supervisé : Ne nécessite de classer les données *à priori*
- Modèle Linéaire : Ne fait pas intervenir l'apprentissage profond
- Prendre en compte les *cooccurences d'apparition*
- Prioritiser les relations du type : a - b + c = d ("'Roi"' - "'Homme"' + "'Femme"')

Objectif : Créer un espace vectoriel dans lequel les vecteurs proches sémantiquement sont voisins
(ie : ont une distance euclidienne faible)

Exemple

"'Le chat est bleu, le chien est bleu."'

Mots	*Le*	*Chat*	*Est*	*Bleu*	*Chien*
Le	0	1	0	1	1
Chat	1	0	1	0	0
Est	0	1	0	2	1
Bleu	1	0	2	0	0
Chien	1	0	1	0	0

- Matrice symétrique avec de nombreux zéros : utiliser des tables de hachage / dictionnaires
- Rétropropager le gradient sur la matrice de Cooccurences $\rightarrow$ maximiser la probabilité des cooccurences

Annuaires dans Glove

- Portabilité du modèle Glove
- A chaque entrée d'un annuaire on associe un vecteur
- On créée une matrice de Cooccurences en considérant les fils d'une séquence donnée
  On ne construit la matrice de Cooccurences en utilisant que des annuaires de profondeur au plus d-1
- On construit les vecteurs en utilisant le modèle de Glove

Obstacles techniques

Espace mémoire : Problème récurrent pour Glove,

- Recours aux bases de données SQL
- Parcours du corpus par intervalles
- Recours fréquent aux tables de hachage.

Explosion des valeurs : Pour Glove, les coordonnées des vecteurs atteignent des valeurs trop élevées
$\rightarrow$ AdaGrad règle ce problème ; mais déplacement faible des vecteurs par rapport à position initiale

Bruit sur les données : Filtrer les mots rares / les mots trop fréquents $\rightarrow$ présentent peu d'intérêt

Temps de calcul : Entraînement en parallèle

AdaGrad / Adam

AdaGrad → choix optimal du facteur d'apprentissage ;
Convergence presque sûre de l'algorithme vers un minimum local.

Definition

Si θ est le paramètre à optimiser, en notant G_t le gradient à l'étape t :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\sum_{s=0}^{t} G_t^2 + \epsilon}} \tag{2}$$

→ mais convergence très lente car écrasement rapide du gradient.

AdaGrad / Adam

Adam : Corriger les défauts de AdaGrad
Voir le gradient comme une balle qui doit tomber dans un fossé :

- AdaGrad : balle ralentit en permanence
- Adam : balle avec friction ; prendre en compte le gradient de l'itération précédente en convergeant vers la solution.

Loi de rejet / conservation des vecteurs

Nous souhaitons conserver seulement les mots qui n'apparaissent pas trop de fois.
On utilise la loi suivante :

Definition

Si $p(w_i)$ désigne la probabilité qu'un mot $\in V$ soit gardé :

$$p(w_i) = 1 - \sqrt{\frac{t}{occ(w_i)}} \tag{3}$$

Où t est un paramètre à régler en fonction du corpus considéré
En règle générale : $t = 10^{-5}$

Statistiques d'apparition

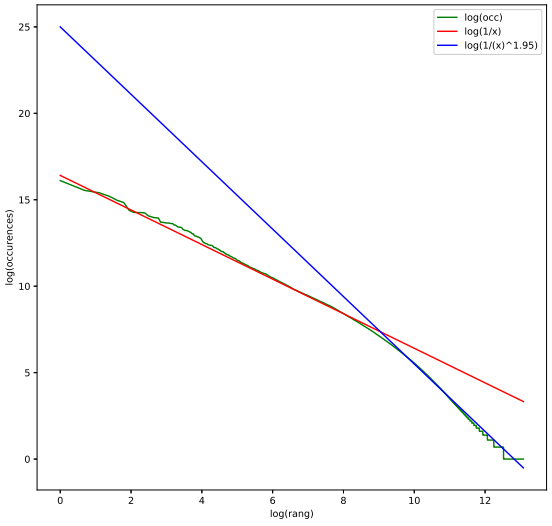


FIGURE – Répartition statistique des mots sur 2500 textes en Anglais

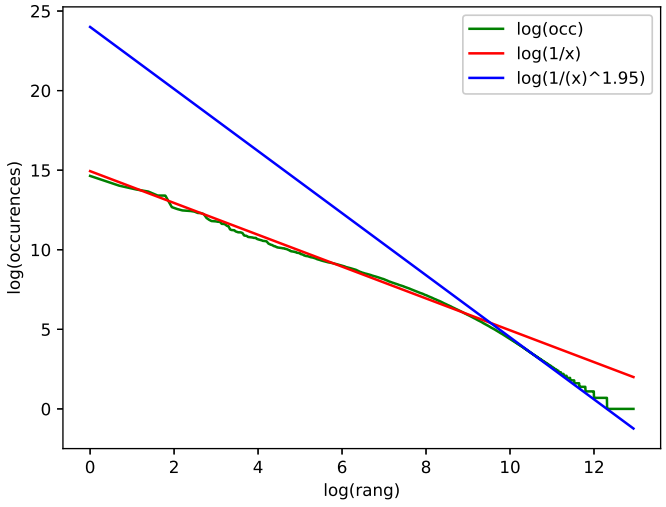


FIGURE – Répartition statistique des mots sur 100000 articles Wikipedia

On constate que :

- Pour $rang \to 0$; $occ \simeq \frac{cte}{rang}$
- Pour $rang \gg 1$; $occ \simeq \frac{cte}{rang^{1,95}}$

Les données précédentes montrent :

- Les mots de rang faible suivent une loi de Zipf de paramètre 1, ie

$$\boxed{occ \simeq \frac{cte}{rang}} \tag{4}$$

- Les mots de rang élevé ($\geqslant 10^9$) suivent une loi de Zipf de paramètre 1,95, ie

$$\boxed{occ \simeq \frac{cte}{rang^{1,95}}} \tag{5}$$

$\rightarrow$ Utiliser ces données pour conserver les mots de rang compris dans $[|50, 25000|]$:
$\Rightarrow$ Gain de calculs & mémoire considérable

```
In [13]: mots_plus_proches("king", 15, data_glove)
Out[13]:
[('king', 0.0),
 ('queen', 5.96625796001019),
 ('monarch', 6.083297165298084),
 ('prince', 6.122456097997923),
 ('kingdom', 6.225537847862595),
 ('reign', 6.494721516242904),
 ('ii', 6.502198343950834),
 ('iii', 6.5304885520536615),
 ('brother', 6.580043387276114),
 ('crown', 6.628460159726231),
 ('uncle', 6.64258473441442),
 ('nephew', 6.659535081094129),
 ('henry', 6.78951710687801),
 ('later', 6.797747398995977),
 ('throne', 6.802164776900861)]
```

FIGURE – Mots les plus proches de roi

```
In [14]: mots_plus_proches("queen", 15, data_glove)
Out[14]:
[('queen', 0.0),
 ('elizabeth', 5.2941372727562355),
 ('king', 5.96625796001019),
 ('princess', 6.073591278300331),
 ('monarch', 6.104674484535929),
 ('majesty', 6.523137268257814),
 ('victoria', 6.547503930700133),
 ('crown', 6.7411931159455145),
 ('lady', 6.764459792886336),
 ('prohertrib', 6.771072466507776),
 ('mary', 6.819301829953269),
 ('wife', 6.8633691090720825),
 ('anne', 6.86376528645953),
 ('mother', 6.864046644429264),
 ('royal', 6.867405366603026)]
```

FIGURE – Mots les plus proches de reine

```
In [15]: mots_plus_proches("nephew", 15, data_glove)
Out[15]:
[('nephew', 0.0),
 ('cousin', 3.5200366309877458),
 ('grandson', 3.6324881792730697),
 ('brother', 3.654198738566214),
 ('uncle', 3.7441776140532674),
 ('niece', 3.936105838655449),
 ('brother-in-law', 3.979447219599491),
 ('son', 4.185539655985056),
 ('son-in-law', 4.19687688239283),
 ('grandfather', 4.465476708488064),
 ('eldest', 4.466465659520938),
 ('father', 4.559710975904477),
 ('father-in-law', 4.666673726350015),
 ('granddaughter', 4.745449900974179),
 ('half-brother', 4.754146546118959)]
```

FIGURE – Mots les plus proches de neveu

Résultats

```
In [17]: mots_plus_proches("versailles", 15, data_glove)
Out[17]:
[('versailles', 0.0),
 ('prohertrib', 5.888411858187896),
 ('saint-quentin-en-yvelines', 6.159482198747226),
 ('em96', 6.261298106633043),
 ('65stk', 6.268195963097953),
 ('kd97', 6.2699838474707175),
 ('bulletinyyy', 6.302017721787259),
 ('k587-1', 6.303111796493487),
 ('str95bb', 6.303608654231275),
 ('http://www.mediabynumbers.com', 6.303809791002046),
 ('bdb94', 6.304077160577641),
 ('k978-1', 6.30424385867805),
 ('bb96', 6.3044518374327465),
 ('mo95', 6.30454667298314),
 ('k977-1', 6.305381577890493)]
```

FIGURE – Mots les plus proches de Versailles

```
In [16]: mots_plus_proches("france", 15, data_glove)
Out[16]:
[('france', 0.0),
 ('french', 5.4231850633513),
 ('paris', 6.006624457649388),
 ('belgium', 6.313914426017833),
 ('prohertrib', 6.4955934328064675),
 ('britain', 6.7551159046846845),
 ('europe', 6.846438641180952),
 ('spain', 6.846878038710985),
 ('notably', 6.9474968708639455),
 ('italy', 6.991577495690887),
 (',', 6.9993011212307215),
 (',', 6.9993011212307215),
 ('chirac', 7.067905420848022),
 ('germany', 7.080351922873561),
 ('switzerland', 7.118400747335977)]
```

FIGURE – Mots les plus proches de France

Résultats

```
In [21]: mot_plus_proche_vect("king","man","woman", 15,data_glove)
Out[21]:
[(5.955312135833397, 'queen'),
 (6.899857138395974, 'monarch'),
 (7.178615168672639, 'mother'),
 (7.252287843194893, 'princess'),
 (7.27729814118346, 'daughter'),
 (7.294615427527313, 'elizabeth'),
 (7.294839211997365, 'throne'),
 (7.348560267830511, 'kingdom'),
 (7.363602855741058, 'wife'),
 (7.540207175395389, 'prince'),
 (7.556018088358674, 'crown'),
 (7.563486257853268, 'niece'),
 (7.646080158330081, 'husband'),
 (7.669424984313133, 'sister'),
 (7.6781205726680986, 'granddaughter')]
```

FIGURE – Roi - Homme + Femme

```
In [24]: mot_plus_proche_vect("france","paris","germany",
15,data_glove)
Out[24]:
[(8.837405020700695, 'austria'),
 (8.93496862619395, 'german'),
 (9.117724391126702, 'italy'),
 (9.14127963304116, 'poland'),
 (9.15142646771015, 'denmark'),
 (9.2483992127262, 'belgium'),
 (9.25646363230096, 'britain'),
 (9.271052672933141, 'netherlands'),
 (9.34359864434531, 'switzerland'),
 (9.411433585379783, 'sweden'),
 (9.466193337411747, 'germans'),
 (9.475954371270683, 'europe'),
 (9.49907043312509, 'spain'),
 (9.556791993410897, 'hungary'),
 (9.582111536504652, 'slovakia')]
```

FIGURE – France - Paris + Allemagne

Les graphiques précédents ont été réalisés avec Glove, entraîné sur wikipedia_en,
Vecteurs de dimension 300 et 15 training epoch

Projection de vecteurs

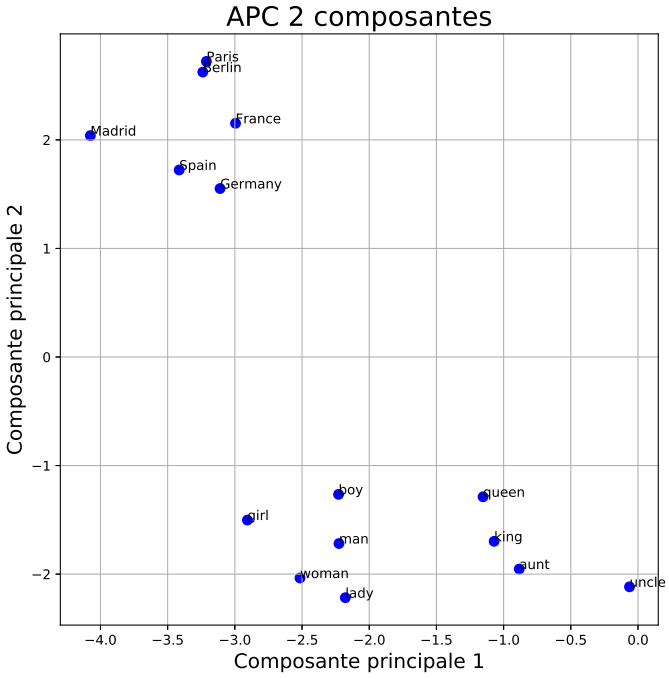


FIGURE – Projection de vecteurs sur un plan

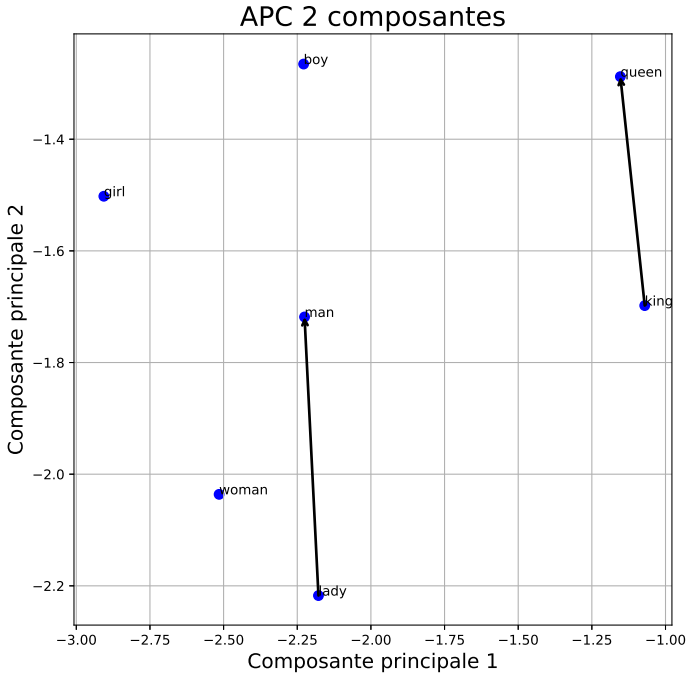


FIGURE – Mise en évidence des opérations algébriques

Les graphiques ont été réalisés sur des vecteurs produits avec Glove de dimension 300, entraînés sur Wikinews

Résultats

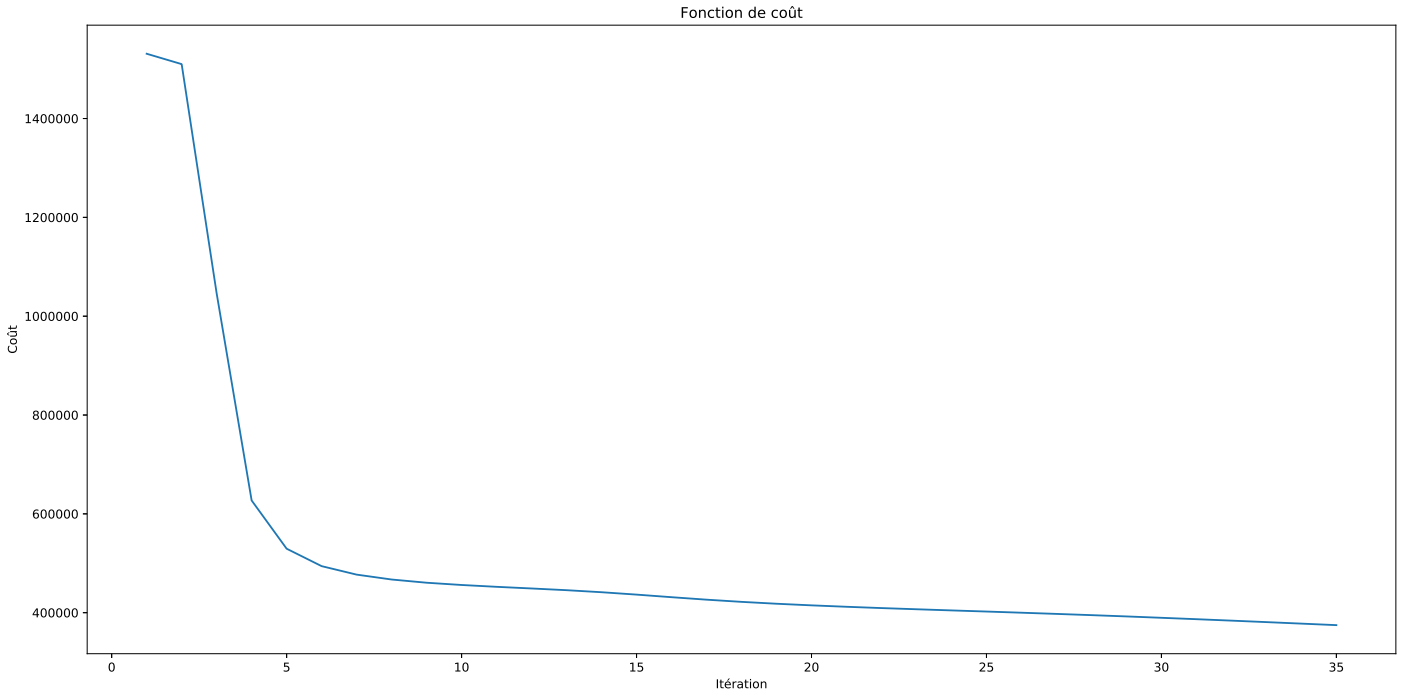


FIGURE – Fonction de coût entraînée sur 500 textes, Vecteurs de dimension 300, sans AdaGrad

Résultats

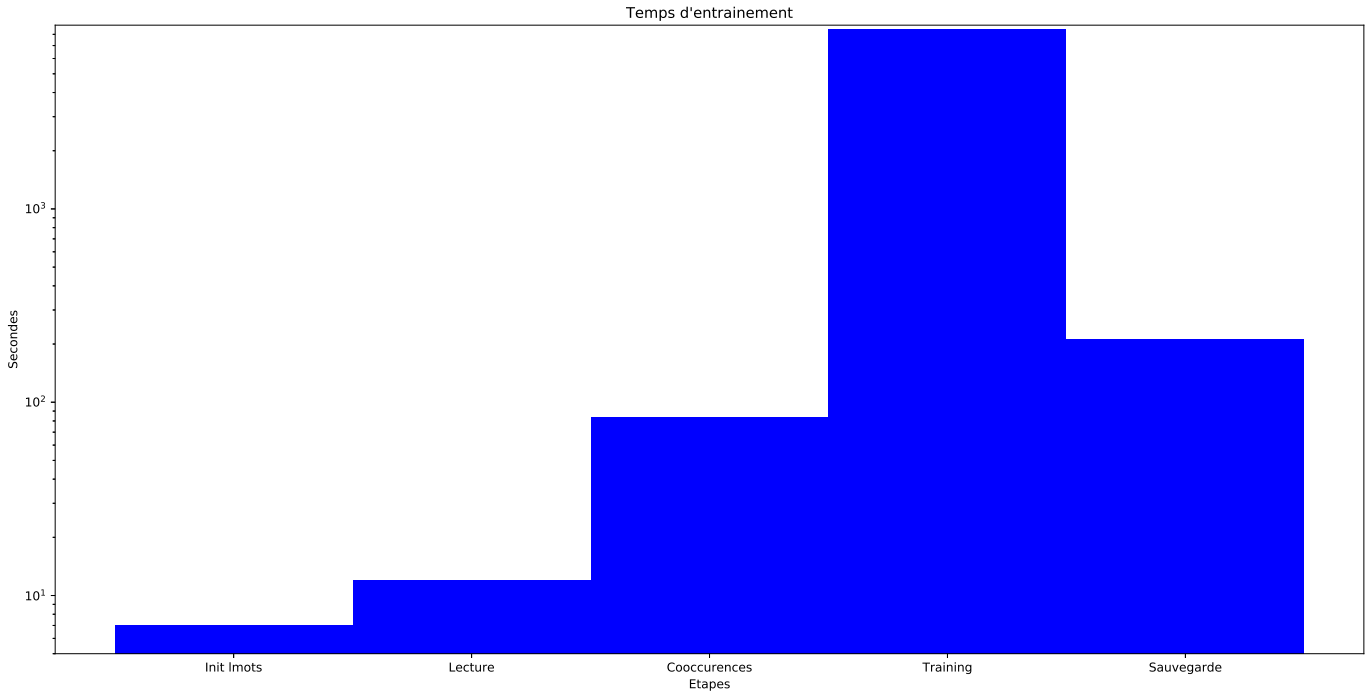


FIGURE – Durée des étapes
Vecteurs de dimension 300, sans AdaGrad

Résultats

```
In [20]: mots_plus_proches("lady",20)
Out[20]:
[(0.0, 'lady'),
 (0.23292236785484646, 'captain'),
 (0.27719593992168196, 'father'),
 (0.27938671184864516, 'mother'),
 (0.3143063914974763, 'woman'),
 (0.32758380808374205, 'poor'),
 (0.34843276988799543, 'hath'),
 (0.3593950893311916, 'voice'),
 (0.3625992175955851, 'name'),
 (0.3799267771742325, 'mind'),
 (0.38107050342354365, 'dear'),
 (0.38356040648475304, 'heard'),
 (0.3837026253244614, 'boy'),
 (0.395315476376302, 'cried'),
 (0.3986441330426899, 'began'),
 (0.4024505384891481, 'myself'),
 (0.40518466848728985, 'seemed'),
 (0.4077631312446678, 'rather'),
 (0.40902632133321104, 'girl'),
```

FIGURE – Mots les plus proches de lady

```
In [13]: mots_plus_proches("queen",10)
Out[13]:
[(0.0, 'queen'),
 (0.08566231527389288, 'paul'),
 (0.11066148795219595, 'colonel'),
 (0.11904788948144623, 'bit'),
 (0.12220375679099774, 'knight'),
 (0.12549152681483147, 'tired'),
 (0.13268016939306704, 'grace'),
 (0.13281605044386746, 'tristram'),
 (0.13767118112389104, 'martin'),
 (0.14130333489212446, 'prince')]
```

FIGURE – Mots les plus proches de queen

Les graphiques ont été réalisés sur des vecteurs produits avec Glove de dimension 200, entraînés sur 500 textes

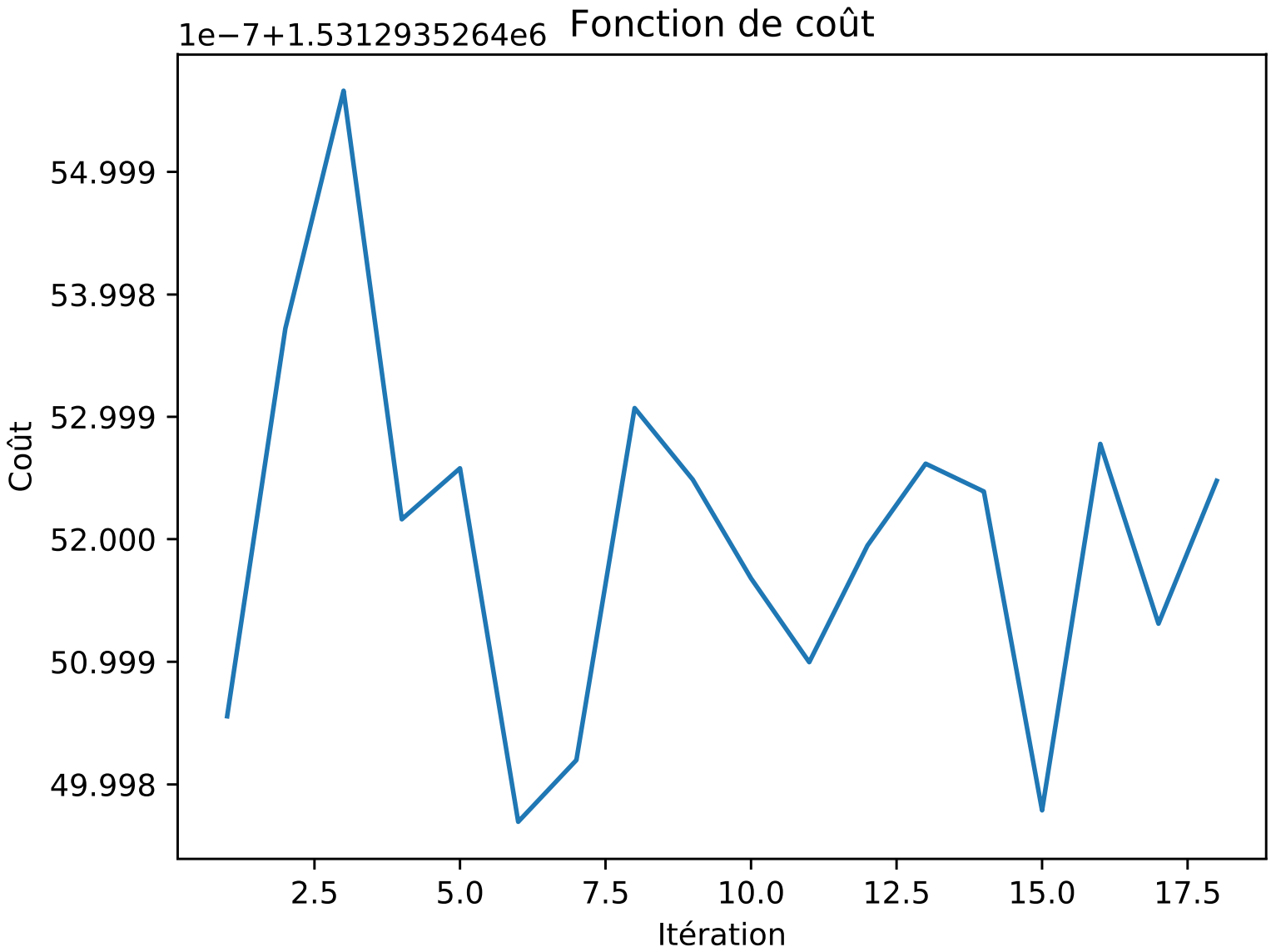


FIGURE – Fonction de coût entraînée sur 500 textes,
Vecteurs de dimension 300, avec AdaGrad

Résultats

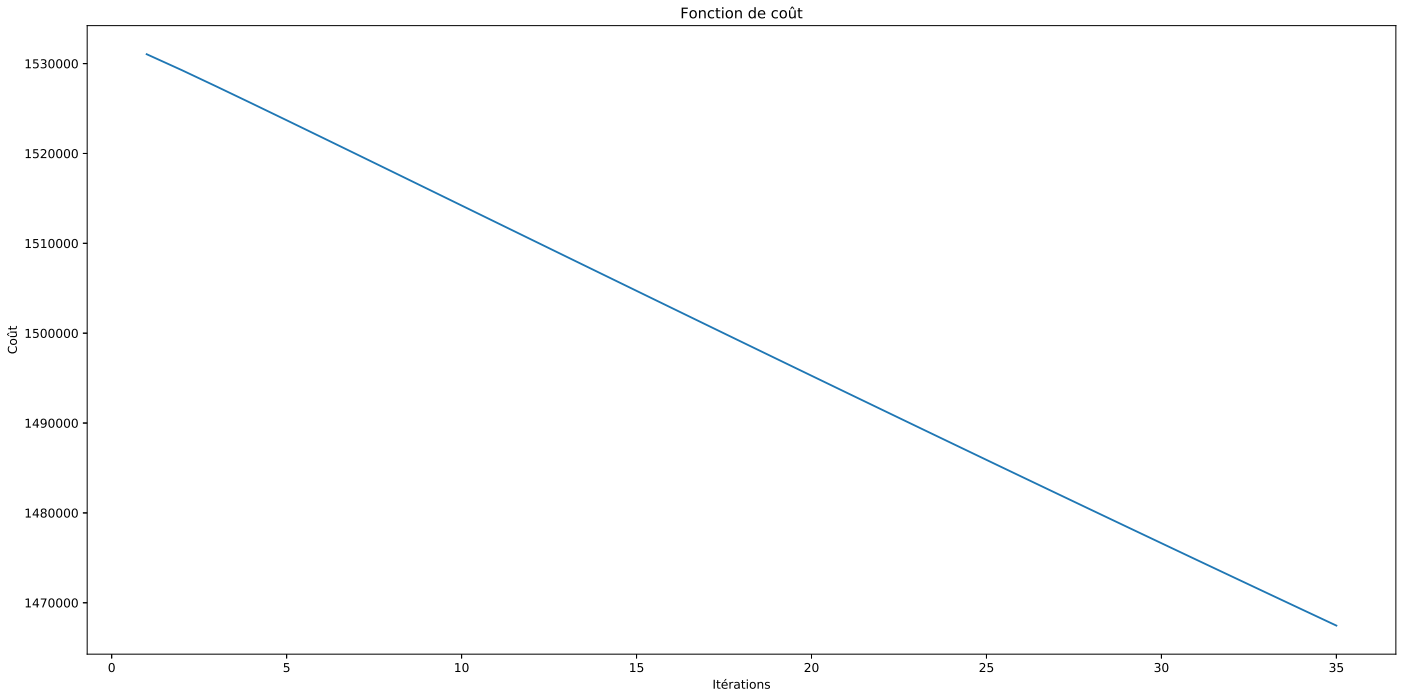


FIGURE – Fonction de coût entraînée sur 500 textes,
Vecteurs de dimension 300, avec Adam

Durée du training : 7798 s ($\simeq$ 2h) contre 4800s sans Adam

Paramètres : $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 0.002$

Résultats

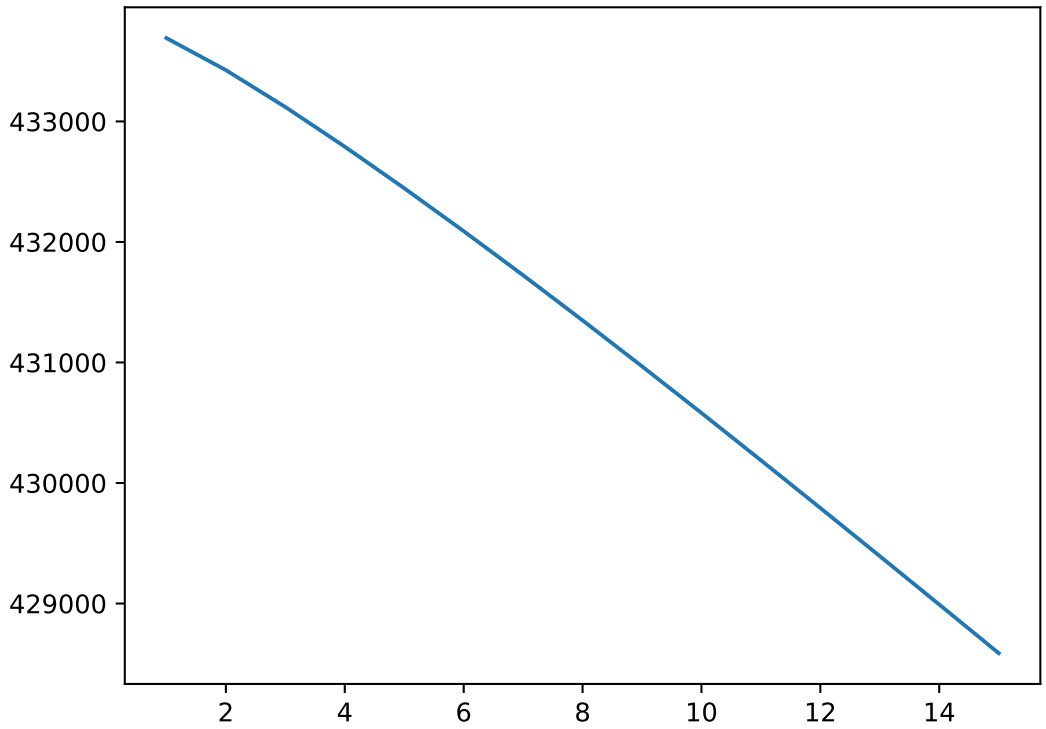


FIGURE – Fonction de coût entraînée sur 500 textes
Vecteurs de dimension 300, avec Adam

$\rightarrow$ Entraînement après 15 itérations sans Adam

Word2Vec

Caractéristiques :

Réseau de neurones : Modèle similaire à Glove : pas de réseau de neurones profond,

Continuous Bag Of Words (CBOW) : Prédire un mot à partir des mots qui l'entourent

Skip Gram Model : Prédire le contexte du mot d'entrée (ie les mots qui l'entourent)

Word2Vec : Principe de fonctionnement

Matrice des vecteurs d'entrée : (matrice imput layer -> hidden layer)

$$\underbrace{\begin{pmatrix} v_{1,1} & \dots & v_{i,1} & \dots & v_{V,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ v_{1,j} & \vdots & v_{i,j} & \vdots & v_{V,j} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ v_{1,N} & \dots & v_{i,N} & \dots & v_{V,N} \end{pmatrix}}_{v \in Mat_{V,N}(\mathbb{R})}$$

Chaque colonne correspond à la représentation d'un mot d'entrée en vecteur $\in \mathbb{R}^N$

Word2Vec : Principe de fonctionnement

Matrice des vecteurs de sortie : (matrice hidden layer -> output layer)

$$\underbrace{\begin{pmatrix} v'_{1,1} & \cdots & v'_{i,1} & \cdots & v'_{V,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ v'_{1,j} & \vdots & v'_{i,j} & \vdots & v'_{V,j} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ v'_{1,N} & \cdots & v'_{i,N} & \cdots & v'_{V,N} \end{pmatrix}}_{v' \in Mat_{V,N}(\mathbb{R})}$$

Chaque colonne correspond à la représentation d'un mot de sortie en vecteur $\in \mathbb{R}^N$

Word2Vec : CBOW (1 Input / 1 Output)

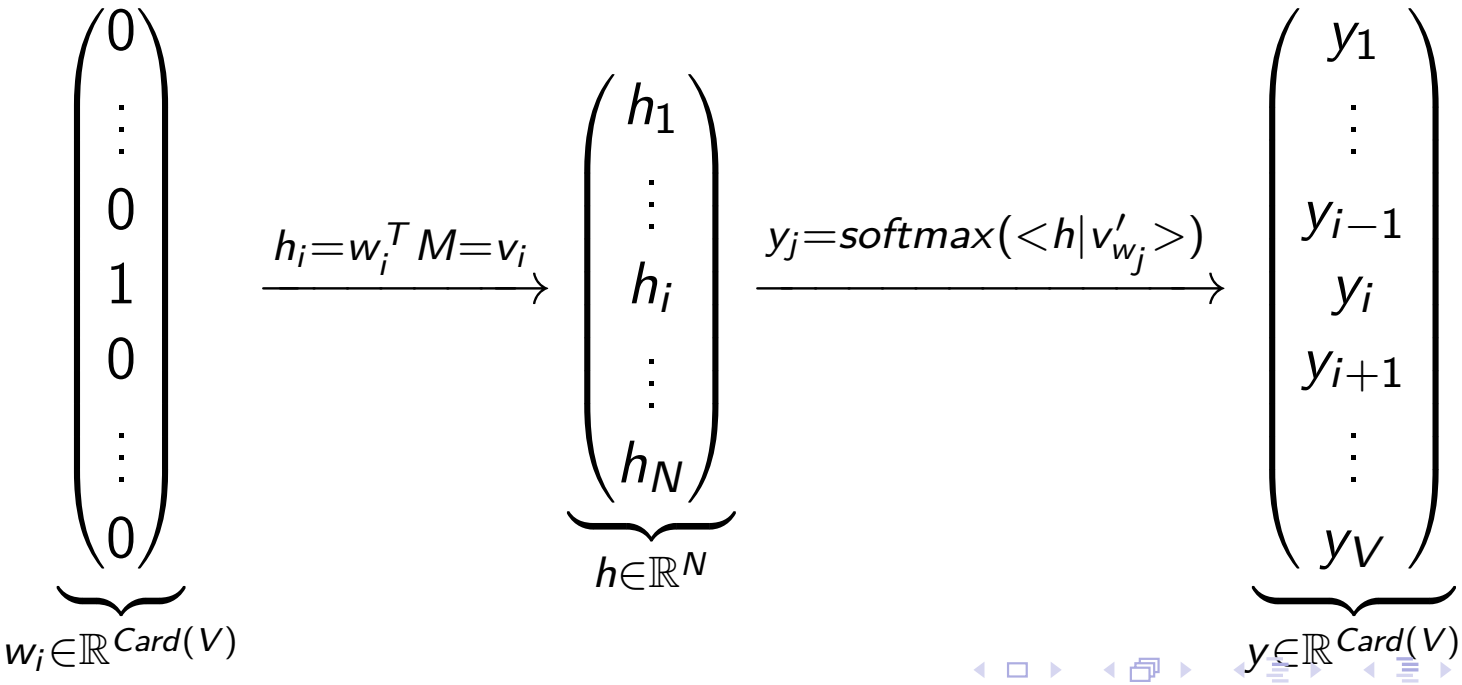
Definition

w_i : *mot d'entrée* ; vecteur de la forme $(\delta_{i,j})_{j\in[|1,V|]}$,

v_{w_i} : *imput vector* du mot d'entrée w_i

v'_{w_j} : *output vector* du mot de sortie (mot dont le contexte est connu) w_j

y : *mot de sortie* (vecteur $\in \mathbb{R}^N$) $\rightarrow$ $y \simeq w_{i\pm1}$



Loi Softmax : Loi de probabilité de la forme :

Definition

w_i : un mot d'entrée $\in V$, de représentation vectorielle v_{w_i}

w_j : un mot dont w_i appartient au contexte, de représentation vectorielle v'_{w_j}

La probabilité d'apparition du mot w_j si w_i est un mot de contexte :

$$P(w_j/w_i) := \frac{\exp(< h|v'_{w_j} >)}{\sum_{k\in V} \exp(< h|v'_{w_k} >)}$$

$$= \frac{\exp(< m_{w_i}|c_{w_j} >)}{\sum_{k\in V} \exp(< m_{w_i}|c_{w_k} >)} \quad (6)$$

$\rightarrow$ Calculer le produit scalaire sur tous les mots du vocabulaire !

Word2Vec : CBOW (1 Input / 1 Output)

Exemple

Le $\underbrace{chat}_{\text{mot entrée}}$ $\underbrace{gris}_{\text{mot sortie}}$ est malade

- chat $= \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$; chat $\rightarrow v_{chat}$
- gris $\rightarrow v'_{gris}$
- y $= softmax(< v_{chat} | v'_{gris} >)$
- On cherche à orienter v'_{gris} et v_{chat} de sorte que $y \simeq \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$

Glove vs Word2Vec ; axes d'optimisation

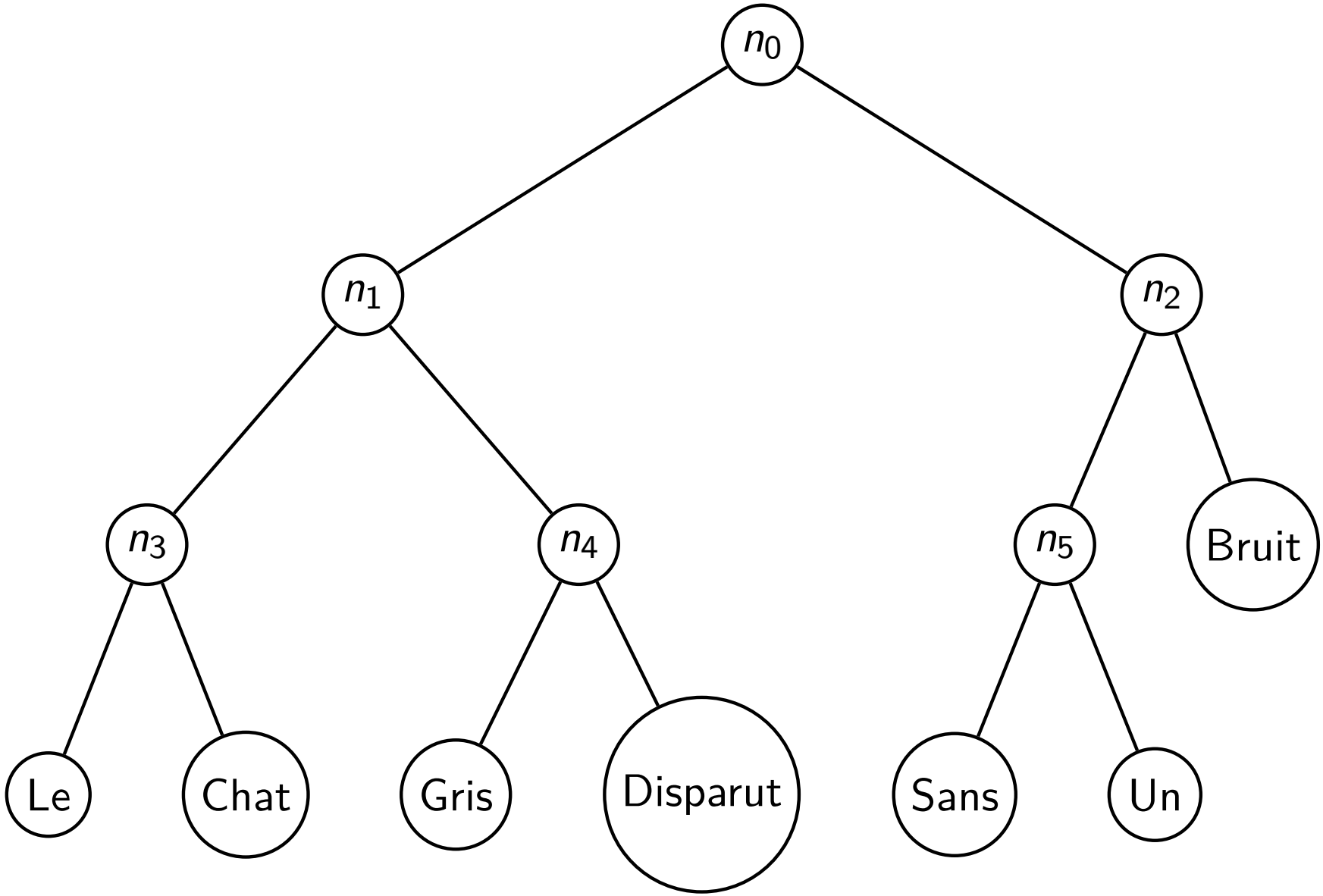
Propriété	*Glove*	*Word2Vec*
Opération	Efficace en temps de calcul $\Theta(Long(Texte))$	Très coûteux en calcul $\Theta(Long(Texte) \times Card(V))$
Espace mémoire	Très coûteux en espace mémoire $\Theta(Card(V)^2)$	Nécessite moins d'espace mémoire $\Theta(N \times Card(V))$
Gradient adaptatif	Adaptative Gradient Learning (AdaGrad)	Décroissance linéaire
Axes d'amélioration	Filtres pour éviter mots trop / peu fréquents	Hiercharchical Softmax / Negative Sampling

Optimisations de Word2Vec

Hierarchical Softmax : Ne plus utiliser des vecteurs, mais des *chemins* dans un arbre de Huffman

Negative Sampling : Ne mettre à jour qu'une portion du vocabulaire.

Softmax hierarchique



Definition

Noeuds : On associe à chaque noeud un vecteur $v'_{n_{m_j}}$

Pas de "'output vector"'

Loi de probabilité : Loi logistique de paramètres 0 et 1 :

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

Espérance $= 0$, Variance $= \frac{\pi}{3}$

Arbre de Huffman : Arbre binaire

- Hauteur h $= \Theta(\log_2(V))$
- *Parfait, équilibré*
- V-1 noeuds
- Chemin unique pour accéder à une feuille depuis la racine
- Probabilité de choisir un mot = produit des probabilités de choisir un noeud

Rétropropagation du gradient : orienter les vecteurs dans l'espace en *rétropropageant l'erreur* sur l'objectif à atteindre

Fonction de coût : On souhaite *maximiser la probabilité* d'obtenir le mot w_O apparaissant dans le contexte de w_I

D'où la fonction de coût :

Definition

Soit θ_{voc} le paramètre de l'algorithme à ajuster.

$$\begin{aligned} -\max(p(w_O|w_I,\theta_{voc})) &= -\max(\log(y_{w_O},\theta_{voc})) \\ &= -\max(< h|v'_{w_j} > -\log(\sum_{k=1}^{V}\exp(< h|v'_{w_k} >)),\theta_{voc}) \\ &:= \max(E,\theta_{voc}) \quad (8) \end{aligned}$$

Ici $E = -\log p(w_O|h,\theta_{voc})$ est la fonction de perte à minimiser.

Word2Vec : CBOW (1 Input / 1 Output)

Equations de mise à jour des poids des vecteurs de contextes :

Definition

Si $u_j := < h | c_{w_j} >$ alors :

$$\frac{\partial E}{\partial u_j} = p(w_j|w_I) - \delta_{I,j} := e_j \tag{9}$$

e_j n'est autre que l'erreur sur la prédiction pour le mot j

$$\frac{\partial E}{\partial (v'_{w_j})_k} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial (v'_{w_j})_k} = e_j \cdot h_k \tag{10}$$

$$(v'_{w_j})_k^{nouveau} = (v'_{w_j})_k^{ancien} - \eta \cdot e_j \cdot m_{w_k} \tag{11}$$

Word2Vec : CBOW (1 Input / 1 Output)

Equation de mise à jour des poids des vecteurs d'entrée :

Definition

Les règles de dérivation en chaîne donnent :

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{Card(V)} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{Card(V)} e_j \cdot (v'_{w_i})_j := EH_i \tag{12}$$

$$v_{w_I}^{nouveau} = v_{w_I}^{ancien} - \eta \cdot EH \tag{13}$$

CBOW multi-word context

En pratique, CBOW prend en compte un contexte de taille supérieure à 1 (typiquement l'algorithme regardera 5 mots avant et 5 mots après le mot principal)

Definition

On se donne k $\in \mathbb{N}$
Si C $= (w_{I-k}, \ldots, w_{I-1}, w_{I+1}, \ldots, w_{I+k})$ le contexte d'un mot du texte w_I
Le vecteur h défini sur le hidden layer devient :

$$h := \frac{1}{C} \cdot (v_{w_{I-k}} + \ldots + v_{w_{I+1}} + \ldots + v_{w_{I+k}}) \tag{14}$$

Skip-Gram vs CBOW

Inverse de CBOW : déterminer un mot à partir de son contexte

Imput Vector / Hidden Layer : Inchangé ;
une seule entrée : h $= v_{w_I}$

Output Vector : Devient un (Card(C))-uplet :
$(y_{1,O}, \ldots, y_{Card(C),O})$

Skip-Gram loss function

Definition

La fonction de coût pour le Skip-Gram devient :

$$E = -\log p(w_{O,1}, w_{O,2}, \ldots, w_{O,Card(C)} | w_I, \theta_{voc})$$

$$= -\sum_{c=1}^{Card(C)} < v'_{w_{O,i}} | v_{w_I} > + C \cdot \log \sum_{j'=1}^{Card(V)} \exp(< v'_{w_{j'}} | v_{w_I} >) \quad (15)$$

Par indépendance supposée des événements $(p(w_j \in context | w_I))_{w_j \in V}$

Modèle probabiliste

Definition

Soit $w \in V$

On note L(w) la longueur du chemin de la racine à w

La probabilité que w soit un mot de sortie est :

$$p(w = w_O | w_I) = \prod_{j=1}^{L(w)-1} \sigma(Bool[n(w, j+1) = fils_g(n(w, j))] \cdot < v'_{n(w,j)} | h >) \tag{16}$$

Où Bool est la fonction :

$$Bool : x \rightarrow \begin{cases} 1 & si\ x\ est\ vraie \\ -1 & sinon \end{cases} \tag{17}$$

Negative sampling

Negative sampling : Pallier d'une autre façon aux calculs de Word2Vec

Idée : Ne mettre à jour qu'une partie des vecteurs bien choisie

Ici on dispose de deux ensembles :

- Un ensemble D de (mots,contexte) tels que (w,c) appartienne effectivement au corpus
- Un ensemble D' de (mots,contexte) qui n'appartiennent pas au corpus

Negative sampling

Definition

Loi de probabilité σ : $P((w, c) \in Corpus) = \sigma(< w|h_c >)$
$\rightarrow$ independant binary classification task

Conditions :

- Considérer D' pour *éviter les solutions triviales*
- *Choisir judicieusement les vecteurs de D'* : prendre des mots représentatifs du corpus

Negative sampling

Definition

Choix des mots de D' : Probabilité de choisir le couple (w,c)

$$P((w,c)) \simeq \frac{occ(w) \cdot occ(c)^{\frac{3}{4}}}{Z} \tag{18}$$

Z étant une constante de normalisation.
Nouvelle fonction de coût

$$E = -\log \sigma(< v'_{w_O} | h_c >, \theta_{voc}) - \sum_{w_i \in W_{neg}} \log \sigma(- < v'_{w_i} | h >, \theta_{voc}) \tag{19}$$

Le principe de rétropropagation du gradient est le même
On ne met à jour que les vecteurs de D et D'

Softmax Hiérarchique

Definition

Fonction à minimiser (fonction de perte) :
On note Bool $= Bool(n(w, j+1) = fils\ gauche(n(w, j))$

$$E = -\log p(w = w_O | w_I) = -\sum_{j=1}^{L(w)-1} \log \sigma(Bool \cdot < v'_{n(w,j)} | h >) \tag{20}$$

On a par la suite dérivé cette fonction pour obtenir les équations de rétropropagation

Fonction de coût

Modèle : On cherche une fonction de la forme :

$$F(w_i, w_j, r_k) = \frac{P(r_k|w_i)}{P(r_k|w_j)} \tag{21}$$

$$F(w_i, w_j, r_k) = \frac{Cooc(r_k, w_i)}{\sum_{r_i \in Context(w_i)} Cooc(r_k, w_i)} \cdot \frac{Cooc(r_k, w_j)}{\sum_{r_j \in Context(w_j)} Cooc(r_k, w_j)}$$

Conditions : Pour respecter la structure d'espace vectoriel, on cherche une solution :

- Linéaire
- Symétrique en w_i, w_j et qui ne distingue pas w_s de $r_s \; \forall s \in [|1, Card(V)|]$

Ainsi on aboutit à :

$$E = \sum_{i,j=1}^{V} f(Cooc(m_i, m_j))(< m_i | c_j > + b_{m_i} + b_{cj} - \log Cooc(m_i, m_j))^2 \quad (22)$$

- f est la fonction :

$$f(x) = \begin{cases} (\frac{x}{x_{max}})^{\alpha} & \forall x < x_{max} \\ 1 & sinon \end{cases}$$

- b_{m_i} et b_{cj} sont des poids propres aux mots m_i et m_j
- $Cooc$ est la matrice de Cooccurences
- $\alpha = \frac{3}{4}$ (empiriquement)

AdaGrad / Adam

$\rightarrow$ Pallier aux problèmes de AdaGrad

Décroissance exponentielle de la moyenne des carrés des gradients :

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) G_t^2$$

Décroissance exponentielle de la moyenne des gradients :

$$m_t = \beta_1 v_{t-1} + (1 - \beta_1) G_t$$

Definition

Mise à jour du paramètre θ :

Correction des paramètres pour éviter une décroissance trop rapide

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \ \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \tag{23}$$

En règle générale : $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\eta = 0.002$

Convergence de la méthode

Definition

$(f_1, \ldots, f_T)$: l'ensemble des fonction convexes de coût aux étapes 1 à T

$\theta^* := argmin_{\theta \in F} \sum_{t=1}^{T} f_t(\theta)$, avec F l'ensemble des valeurs que peut prendre θ

Alors le regret sur l'objectif est défini par :

$$R(T) = \sum_{t=1}^{T} [f_t(\theta_t) - f_t(\theta^*)] \tag{24}$$

On montre que,

- Si f_t a un gradient borné
- Les distances entre les θ_i générés par Adam, sont bornées

Alors $\frac{R(T)}{T} = O(\frac{1}{\sqrt{T}})$