PSTALN_TP04_embeddings

December 15, 2021

0.1 Word embeddings (statiques)

- Notebook créé par Benoit Favre
- Adapté par Carlos Ramisch

La plupart de ces notebooks peuvent tourner sans acceleration GPU. Vous pouvez utiliser Google Colab ou installer jupyter-notebook sur votre ordinateur. Pytorch est déjà installé sur Colab, par contre il faut suivre https://pytorch.org/get-started/locally/ en local.

Notez que pour ce TP nous utilisons les bibliothèques `numpy`, `nltk`, `sklearn`, `umap-learn` et `regex` que nous n'avons pas utilisées jusqu'à présent.

[3]:
```
## example de deploiement local
# virtualenv -ppython3.8 pstaln-env
# source pstaln-env/bin/activate
# pip3 install torch==1.10.0+cpu -f https://download.pytorch.org/whl/cpu/
 ↪torch_stable.html
# pip3 install matplotlib ipykernel
# pip3 install numpy
# pip3 install sklearn
# pip3 install umap-learn
# pip3 install nltk
# pip3 install regex
# python3 -m ipykernel install --user --name=pstaln-env
# jupyter-notebook
## puis sélectionner le noyeau pstaln-evn
```

Nous nous intéressons à créer des plongements lexicaux (word embeddings) avec plusieurs méthodes. Pour cela, nous allons télécharger un corpus, le pré-traiter pour en extraire des coocurrences, puis calculer les emebddings à partir de ces coocurrences et finalement évaluer la qualité de ces embeddings.

Le corpus que nous allons utiliser est relativement petit. Il s'agit du livre "Le tour du monde en 80 jours" de Jules Vernes. Ce livre est disponible en .txt sur le site de l'ABU, la bibliothèque universelle du CNAM (http://abu.cnam.fr) qui contient plus de 200 ouvrages par des grands auteurs français.

Ce corpus est relativement petit pour garder des temps de calculs raisonnables, et il est généralement admis que pour apprendre des embeddings, il faut au minimum 10 millions de mots, voir plusieurs centaines de millions.

[4]:
```
%%bash
wget -q 'http://abu.cnam.fr/cgi-bin/donner_unformated?tdm80j2' -O - | iconv -f␣
 ↪latin1 -t utf-8 > tdm80j2.txt
# for about 200 times more data, use:
#curl -s https://pageperso.lis-lab.fr/benoit.favre/files/abu-all.txt.xz | xz -d␣
 ↪> input.txt
head tdm80j2.txt
```

```
--- ATTENTION : CONSERVEZ CETTE LICENCE SI VOUS REDISTRIBUEZ CE FICHIER ---
License ABU
-=-=-=-=-=-
Version 1.1, Aout 1999

Copyright (C) 1999 Association de Bibliophiles Universels
   http://abu.cnam.fr/
   abu@cnam.fr

La base de textes de l'Association des Bibliophiles Universels (ABU)
```

Avant de continuer, nous allons importer les librairies dont nous avons besoin aujourd'hui.

[17]:
```
import re
import regex
import collections
import nltk
nltk.download('punkt')

import numpy as np
import math
from matplotlib import pyplot as plt

import sklearn
import umap.umap_ as umap

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
```

```
[nltk_data] Downloading package punkt to /home/ceramisch/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

0.1.1 Pré-traitements du texte

Comme on peut le voir dans la sortie précédente, le fichier contient une license que nous ne voulons pas traiter. La première étape est donc de charger le texte sans la license. Notez que ce dernier est encodé en latin1 et qu'il faut spécifier cet encodage lors du chargement. On pourrait aussi

filtrer tous les autres éléments textuels non pertinents comme le titre ou le sommaire, mais nous considèrerons que leur impact est négligeable.

[6]:
```
with open('tdm80j2.txt', encoding="utf-8") as fp:
  text = fp.read().replace('\n', '\\n')
  found = re.search('- DEBUT DU FICHIER tdm80j2 -*', text)
  if found:
    text = text[found.end():]
  found = re.search('-* FIN DU FICHIER tdm80j2', text)
  if found:
    text = text[:found.start()]
  text = text.replace('\\n', '\n')

print(text[:500])
```

```
                          LE TOUR DU MONDE
                                 EN
                          QUATRE-VINGTS JOURS
                ====================================

                          par Jules Verne

                                 I
                          --------------------
             DANS LEQUEL PHILEAS FOGG ET PASSEPARTOUT
           S'ACCEPTENT RÉCIPROQUEMENT L'UN COMME MAÎTRE,
                       L'AUTRE COMME DOMESTIQUE

En l'année 1872, la maison portant
```

La prochaine étape est de tokeniser le texte, c'est à dire convertir la séquence de caractères en séquence de mots. La tokenisation est un processus délicat généralement effectué à l'aide de règles et de listes d'exceptions. Nous utiliserons NLTK qui fournit un tokeniseur standard qui marche raisonnablement bien pour le français. Mais ce tokeniseur n'est pas parfait car il ne détache pas les déterminants élidés (comme d') des mots, ce que nous devons faire manuellement.

Notez que NLTK nécessite de télécharger un modèle pour effectuer la tokenisation. Il existe de nombreux tutoriels pour NLTK (par exemple https://www.guru99.com/nltk-tutorial.html) qui démontrent la plupart des possibilités de cette bibliothèque.

Trouvez la fonction NLTK qui permet de segmenter en texte en mots en français et appliquez-la sur le texte pour obtenir une liste de mots.

[7]:
```
## à compléter ##
# words = ...
```

```
print(words[:100], words[-100:])
print(len(words))
```

```
['LE', 'TOUR', 'DU', 'MONDE', 'EN', 'QUATRE-VINGTS', 'JOURS',
'====================================', 'par', 'Jules', 'Verne', 'I', '--',
'--', '--', '--', '--', '--', '--', '--', '--', '--', 'DANS', 'LEQUEL',
'PHILEAS', 'FOGG', 'ET', 'PASSEPARTOUT', "S'ACCEPTENT", 'RÉCIPROQUEMENT',
"L'UN", 'COMME', 'MAÎTRE', ',', "L'AUTRE", 'COMME', 'DOMESTIQUE', 'En',
"l'année", '1872', ',', 'la', 'maison', 'portant', 'le', 'numéro', '7', 'de',
'Saville-row', ',', 'Burlington', 'Gardens', '--', 'maison', 'dans', 'laquelle',
'Sheridan', 'mourut', 'en', '1814', '--', ',', 'était', 'habitée', 'par',
'Phileas', 'Fogg', ',', 'esq.', ',', "l'un", 'des', 'membres', 'les', 'plus',
'singuliers', 'et', 'les', 'plus', 'remarqués', 'du', 'Reform-Club', 'de',
'Londres', ',', 'bien', "qu'il", 'semblât', 'prendre', 'à', 'tâche', 'de', 'ne',
'rien', 'faire', 'qui', 'pût', 'attirer', "l'attention", '.'] ['lequel',
'Phileas', 'Fogg', 'engage', 'une', 'lutte', 'directe', 'contre', 'la',
'mauvaise', 'chance', 'XXXIII', '.', 'Où', 'Phileas', 'Fogg', 'se', 'montre',
'à', 'la', 'hauteur', 'des', 'circonstances', 'XXXIV', '.', 'Qui', 'procure',
'à', 'Passepartout', "l'occasion", 'de', 'faire', 'un', 'jeu', 'de', 'mots',
'atroce', ',', 'mais', 'peut-être', 'inédit', 'XXXV', '.', 'Dans', 'lequel',
'Passepartout', 'ne', 'se', 'fait', 'pas', 'répéter', 'deux', 'fois', "l'ordre",
'que', 'son', 'maître', 'lui', 'a', 'donné', 'XXXVI', '.', 'Dans', 'lequel',
'Phileas', 'Fogg', 'fait', 'de', 'nouveau', 'prime', 'sur', 'le', 'marché',
'XXXVII', '.', 'Dans', 'lequel', 'il', 'est', 'prouvé', 'que', 'Phileas',
'Fogg', 'n', "'", 'a', 'rien', 'gagné', 'à', 'faire', 'ce', 'tour', 'du',
'monde', ',', 'si', 'ce', "n'est", 'le', 'bonheur']
82174
```

Cette tokénisation n'est pas parfaite, par exemple, elle garde certains articles attachés aux noms qui les suivent *l'ordre*. Mais pour le moment ce sera suffisant pour ce TP.

On peut voir en observant la séquence de tokens produite précédemment qu'il peut être souhaitable de normaliser un peu les mots avant de les traiter. Par exemple, on pourrait mettre les mots en miniscule, filtrer les séquences de ponctuation et les tokens trop peu fréquents.

Une manière de filtrer les tokens pour n'obtenir que des mots consiste à ne conserver que les mots contenant au moins un caractère alphanumérique. La bibliothèque standard `re` de python ne permet pas d'écrire facilement des expressions régulières prenant en compte les caratères accentués. Nous utiliserons pour cela le module `regex` qui supporte les classes de caractères unicode dont `\p{Ll}` qui représente les caractères alphabétiques. Nous en profitons pour mettre les tokens en minuscule.

[8]:
```
filtered_words = [word.lower() for word in words if regex.
 ↪search(r'(\p{Ll}|\d)', word.lower())]
print(filtered_words[:30])
```

```
['le', 'tour', 'du', 'monde', 'en', 'quatre-vingts', 'jours', 'par', 'jules',
'verne', 'i', 'dans', 'lequel', 'phileas', 'fogg', 'et', 'passepartout',
"s'acceptent", 'réciproquement', "l'un", 'comme', 'maître', "l'autre", 'comme',
```

```
'domestique', 'en', "l'année", '1872', 'la', 'maison']
```

Commençons par calculer la fréquence des tokens et afficher les moins et plus fréquents. Nous utilisons un `collections.defaultdict` pour facilement compter le nombre d'occurrences des mots.

Ensuite, c'est à vous de créer une liste des mots `most_frequent` triée par ordre de fréquence, des plus fréquents (au début de la liste) jusqu'aux moins fréquents (à la fin de la liste).

Questions :

- Quels sont les mots les plus fréquentes du texte ? Et les moins fréquents ?
- D'après ces observations, quelles sont les catégories morphosyntaxiques des mots les plus fréquents ? et des mots les moins fréquents ?

[9]:
```
freq = collections.defaultdict(int)
for word in filtered_words:
  freq[word] += 1

## à compléter ##
# most_frequent = ...

#for word in most_frequent[:10]:
#  print(freq[word], word)

#for word in most_frequent[-10:]:
#  print(freq[word], word)
```

```
2896 de
1891 le
1669 à
1619 la
1601 et
1054 les
1002 il
832 en
770 du
702 un
```

0.1.2 Matrice de co-occurrence

Nous allons maintenant construire un vocabulaire associant les mots à des identifiants numériques. Nous ignorerons les mots qui apparaissent moins de 4 fois pour écarter une partie du bruit et les mots les moins fréquents pour lesquels nous n'avons de toute façons pas suffisemment de données pour apprendre des représentations raisonnables.

[10]:
```
# trick: each new word is given a new id the first time we look it up in the␣
 ↪dictionary
vocab = collections.defaultdict(lambda: len(vocab))

# build vocab by decreasing frequency
```

```
for word in most_frequent:
  if freq[word] > 3:
    vocab[word]

# build array with word ids
int_words = []
for word in filtered_words:
  if word in vocab:
    int_words.append(vocab[word])

# freeze content (trying to get the id of new words with fail)
vocab = dict(vocab)

print(len(vocab))
print(vocab)
```

```
2034
{'de': 0, 'le': 1, 'à': 2, 'la': 3, 'et': 4, 'les': 5, 'il': 6, 'en': 7, 'du':
8, 'un': 9, 'que': 10, 'fogg': 11, 'des': 12, 'se': 13, 'dans': 14, 'qui': 15,
'ne': 16, 'ce': 17, 'une': 18, 'son': 19, 'pas': 20, 'passepartout': 21, 'mais':
22, 'sur': 23, 'au': 24, 'plus': 25, 'lui': 26, 'phileas': 27, 'était': 28,
'pour': 29, 'mr': 30, 'par': 31, 'fix': 32, 'vous': 33, 'avait': 34, 'cette':
35, 'sa': 36, 'ses': 37, 'heures': 38, 'a': 39, 'je': 40, 'on': 41, 'avec': 42,
'répondit': 43, 'tout': 44, "qu'il": 45, 'bien': 46, 'si': 47, 'sans': 48,
'comme': 49, 'y': 50, 'ces': 51, 'est': 52, 'deux': 53, 'monsieur': 54, 'dit':
55, 'aouda': 56, 'mrs': 57, "d'un": 58, 'quelques': 59, 'maître': 60, 'même':
61, 'après': 62, "c'était": 63, "c'est": 64, 'été': 65, 'dont': 66, 'train': 67,
'donc': 68, 'nous': 69, 'fut': 70, 'elle': 71, 'ni': 72, 'où': 73, 'aux': 74,
'ou': 75, 'mille': 76, 'quand': 77, 'encore': 78, "d'une": 79, 'temps': 80,
'leur': 81, 'pendant': 82, 'jours': 83, 'peu': 84, 'faire': 85, 'eût': 86,
'non': 87, 'gentleman': 88, 'avoir': 89, 'aussi': 90, 'là': 91, 'vers': 92,
'pouvait': 93, 'ils': 94, 'rien': 95, 'entre': 96, 'fit': 97, 'cela': 98,
'fait': 99, 'moins': 100, 'puis': 101, 'milles': 102, 'livres': 103, 'mon': 104,
'devait': 105, 'tous': 106, 'point': 107, 'jeune': 108, 'paquebot': 109,
'moment': 110, 'toute': 111, 'très': 112, 'voyageurs': 113, 'femme': 114,
'huit': 115, 'hong-kong': 116, 'londres': 117, 'homme': 118, "n'était": 119,
'demanda': 120, 'trois': 121, 'alors': 122, 'étaient': 123, 'vingt': 124,
'monde': 125, 'cet': 126, 'mer': 127, 'sous': 128, 'bombay': 129, 'fort': 130,
"n'avait": 131, 'être': 132, 'effet': 133, 'cent': 134, "s'était": 135, 'votre':
136, 'voyage': 137, 'ainsi': 138, 'bord': 139, 'lequel': 140, 'dire': 141, 'me':
142, 'serait': 143, 'sir': 144, 'garçon': 145, 'police': 146, 'soir': 147,
'jamais': 148, 'oui': 149, 'fois': 150, 'ville': 151, 'francis': 152,
'domestique': 153, 'jour': 154, 'enfin': 155, 'avaient': 156, 'minutes': 157,
"s'écria": 158, 'depuis': 159, 'cinq': 160, 'départ': 161, 'main': 162, "s'il":
163, 'vent': 164, 'cromarty': 165, 'car': 166, 'sont': 167, "l'inspecteur": 168,
'anglais': 169, 'cependant': 170, 'n': 171, 'station': 172, 'leurs': 173, 'dit-
il': 174, 'matin': 175, 'yokohama': 176, 'route': 177, 'toujours': 178,
```

'contre': 179, 'tour': 180, 'jusqu': 181, 'avant': 182, 'reprit': 183, 'guide': 184, 'travers': 185, 'capitaine': 186, 'prendre': 187, 'comment': 188, 'assez': 189, 'toutes': 190, 'grand': 191, 'vitesse': 192, 'wagon': 193, 'qu': 194, 'moi': 195, "n'est": 196, 'cents': 197, 'reform-club': 198, 'peut-être': 199, 'faut': 200, 'allait': 201, 'quatre': 202, 'devant': 203, 'grande': 204, 'fer': 205, 'fût': 206, 'bientôt': 207, 'maintenant': 208, 'nuit': 209, 'dix': 210, 'furent': 211, 'déjà': 212, 'gare': 213, 'quelque': 214, 'mongolia': 215, 'traversée': 216, 'pilote': 217, "l'un": 218, 'tête': 219, 'seul': 220, "d'ailleurs": 221, "j'ai": 222, "l'on": 223, 'six': 224, 'terre': 225, 'milieu': 226, "l'autre": 227, 'quant': 228, 'coup': 229, 'partie': 230, 'voir': 231, 'aussitôt': 232, 'pris': 233, 'suis': 234, 'place': 235, 'pays': 236, 'midi': 237, 'doute': 238, 'retard': 239, "l'heure": 240, 'parti': 241, 'vapeur': 242, 'pont': 243, 'soit': 244, 'heure': 245, 'eh': 246, 'seulement': 247, 'pourquoi': 248, 'aurait': 249, 'autres': 250, 'pu': 251, 'neuf': 252, 'stuart': 253, 'liverpool': 254, 'suez': 255, 'new': 256, 'san': 257, "l'agent": 258, 'lendemain': 259, 'rangoon': 260, 'faisait': 261, "qu'un": 262, 'français': 263, 'or': 264, 'yeux': 265, 'voleur': 266, 'york': 267, 'francisco': 268, "l'inde": 269, 'consul': 270, 'passagers': 271, 'près': 272, 'carnatic': 273, 'colonel': 274, 'laquelle': 275, 'chemin': 276, 'suivant': 277, "n'en": 278, 'prit': 279, 'trop': 280, 'trouvait': 281, 'affaire': 282, 'pied': 283, 'calcutta': 284, 'compagnie': 285, 'général': 286, 'maison': 287, 'cas': 288, 'quitté': 289, 'peut': 290, 'porte': 291, 'êtes': 292, 'chambre': 293, 'fallait': 294, 'instant': 295, 'andrew': 296, 'pari': 297, 'ah': 298, 'coups': 299, 'tard': 300, "qu'elle": 301, 'quatre-vingts': 302, 'navire': 303, 'compte': 304, 'personne': 305, 'onze': 306, 'partir': 307, 'ma': 308, 'autour': 309, 'eut': 310, 'sept': 311, 'loin': 312, 'voix': 313, 'mandat': 314, 'aucun': 315, 'aucune': 316, 'somme': 317, 'question': 318, 'regardait': 319, 'john': 320, 'montre': 321, "d'abord": 322, 'f': 323, 'décembre': 324, 'quoi': 325, 'mécanicien': 326, 'quel': 327, 'rues': 328, 'proctor': 329, 'passa': 330, 'chez': 331, 'bon': 332, 'octobre': 333, "s'en": 334, '000': 335, "qu'on": 336, 'sorte': 337, 'bras': 338, 'coeur': 339, 'marche': 340, 'voie': 341, 'prix': 342, 'pagode': 343, 'parsi': 344, 'juge': 345, 'pieds': 346, 'ayant': 347, 'dernier': 348, 'certain': 349, 'premier': 350, 'difficile': 351, "l'honorable": 352, 'sieur': 353, 'quai': 354, 'absolument': 355, 'locomotive': 356, "l'henrietta": 357, 'anglaise': 358, 'auquel': 359, 'semblait': 360, 'lieu': 361, 'mains': 362, 'plusieurs': 363, 'autre': 364, 'celui-ci': 365, 'voici': 366, 'détective': 367, 'brigadier': 368, 'tankadère': 369, 'savait': 370, 'nom': 371, 'chaque': 372, 'gagner': 373, 'nouveau': 374, 'afin': 375, 'ici': 376, "qu'une": 377, 'vol': 378, 'ont': 379, 'passer': 380, "l'air": 381, 'voulait': 382, 'simplement': 383, 'passage': 384, 'arrivé': 385, 'conducteur': 386, 'banque': 387, 'vie': 388, 'toutefois': 389, 'longues': 390, 'jeu': 391, 'demie': 392, 'tenait': 393, 'quelle': 394, 'instants': 395, 'eu': 396, 'vint': 397, 'peine': 398, 'sera': 399, 'prime': 400, 'avez': 401, 'hommes': 402, 'port': 403, 'côte': 404, 'voiles': 405, 'moyen': 406, 'compagnons': 407, 'douze': 408, 'goélette': 409, 'voilà': 410, 'pouvaient': 411, 'ceux': 412, 'collègues': 413, 'service': 414, 'vrai': 415, 'brave': 416, 'journée': 417, 'voulez-vous': 418, "qu'ils": 419, 'prêt': 420, 'face': 421, 'put': 422, 'machine': 423, 'croire': 424, 'territoire': 425, 'cris': 426, "l'éléphant":

427, 'elles': 428, 'madame': 429, 'traîneau': 430, 'gentlemen': 431, 'reçu':
432, 'fortune': 433, 'chose': 434, 'perdu': 435, 'conditions': 436, 'quitter':
437, 'calme': 438, 'quarante-cinq': 439, "jusqu'au": 440, 'bank-notes': 441,
'ajouta': 442, 'raison': 443, 'cinquante': 444, 'sac': 445, 'foule': 446, 'sud':
447, 'compagnon': 448, 'ligne': 449, 'devoir': 450, 'restait': 451, 'américain':
452, 'neige': 453, 'speedy': 454, 'saville-row': 455, 'ans': 456, 'vue': 457,
'possible': 458, 'mots': 459, 'tant': 460, 'évidemment': 461, 'au-dessus': 462,
'impossible': 463, 'regard': 464, 'aller': 465, 'ralph': 466, 'choses': 467,
'compris': 468, 'soleil': 469, 'montagnes': 470, 'simple': 471, 'prêtres': 472,
'bois': 473, "n'eût": 474, 'pacifique': 475, 'souvent': 476, 'whist': 477,
'nature': 478, 'rouge': 479, 'salon': 480, 'rapidement': 481, 'arrivait': 482,
'part': 483, 'eux': 484, 'ciel': 485, 'gagné': 486, 'passeport': 487, 'long':
488, 'parcours': 489, 'steamer': 490, 'façon': 491, 'distance': 492, 'notre':
493, 'cabine': 494, "l'avant": 495, 'immense': 496, "l'est": 497, "l'hôtel":
498, 'entendre': 499, 'wagons': 500, 'pensée': 501, 'demain': 502, 'voiture':
503, 'bunsby': 504, "l'avait": 505, 'vu': 506, 'parmi': 507, 'club': 508,
'mouvement': 509, 'arriver': 510, 'table': 511, 'corps': 512, 'venait': 513,
'nez': 514, 'celui': 515, 'sait': 516, 'passé': 517, 'mettre': 518, 'nombre':
519, 'allons': 520, 'hasard': 521, 'japon': 522, 'trains': 523, 'froid': 524,
'honneur': 525, 'froidement': 526, 'nouvelle': 527, 'navires': 528, 'bateau':
529, 'arrêté': 530, "n'ai": 531, 'sommes': 532, 'eaux': 533, 'comptait': 534,
'côté': 535, "l'état": 536, 'intérêt': 537, 'shangaï': 538, 'sioux': 539,
'membres': 540, 'disait': 541, 'cité': 542, 'droit': 543, 'mieux': 544,
"s'étaient": 545, 'petit': 546, 'paris': 547, 'angleterre': 548, 'parole': 549,
'lui-même': 550, 'trouva': 551, 'va': 552, 'vaste': 553, 'chine': 554,
'indiens': 555, '21': 556, 'autant': 557, 'mit': 558, 'gouvernement': 559,
'savoir': 560, 'singapore': 561, 'nord': 562, 'conduire': 563, 'environ': 564,
"l'affaire": 565, 'demander': 566, "d'eau": 567, 'petite': 568, 'arrière': 569,
'kearney': 570, 'avait-il': 571, 'vingt-quatre': 572, 'honnête': 573, "l'homme":
574, 'paraissait': 575, 'large': 576, 'forte': 577, 'programme': 578, 'celle':
579, 'gauthier': 580, "d'angleterre": 581, 'signalement': 582, 'reprendre': 583,
'sais': 584, 'paquebots': 585, 'mauvais': 586, 'perdre': 587, "s'agit": 588,
'poche': 589, 'samedi': 590, 'quitta': 591, 'droite': 592, 'surtout': 593,
"l'arrivée": 594, 'presque': 595, 'faite': 596, 'indes': 597, 'directement':
598, 'tandis': 599, 'suivi': 600, 'retenir': 601, 'parler': 602, 'laissa': 603,
'rencontre': 604, 'idée': 605, 'mot': 606, 'mort': 607, 'roues': 608, 'parfois':
609, 'rajah': 610, "l'honnête": 611, 'tempête': 612, "d'omaha": 613, 'pût': 614,
'grands': 615, 'personnage': 616, 'paroles': 617, 'seconde': 618, 'gens': 619,
'passait': 620, 'frais': 621, 'crois': 622, 'énorme': 623, 'vraiment': 624,
'divers': 625, 'voyait': 626, 'lèvres': 627, 'étant': 628, 'rendit': 629, 'feu':
630, 'extrêmement': 631, 'revint': 632, 'attendant': 633, 'chances': 634,
'autrefois': 635, 'disant': 636, 'trouvé': 637, 'messieurs': 638, 'suite': 639,
'certainement': 640, 'sol': 641, 'sifflet': 642, "d'avoir": 643, 'rapide': 644,
'coquin': 645, 'laisser': 646, 'malgré': 647, 'sud-est': 648, 'situation': 649,
'derniers': 650, 'vit': 651, 'récit': 652, 'demi': 653, 'lorsque': 654,
'soldats': 655, 'lac': 656, 'prévenir': 657, 'capitale': 658, 'venir': 659,
'journaux': 660, 'lutte': 661, 'salle': 662, 'extrême': 663, 'jour-là': 664,
'attendait': 665, 'rendre': 666, 'faisant': 667, 'leva': 668, 'quarante': 669,

'figure': 670, 'belle': 671, 'mal': 672, 'maisons': 673, 'déjeuner': 674,
'portait': 675, 'puisque': 676, 'mettait': 677, 'conversation': 678,
'contraire': 679, 'europe': 680, 'ceci': 681, 'immédiatement': 682,
'précisément': 683, "d'être": 684, 'retour': 685, 'pourrait': 686, 'demanda-t-
il': 687, 'voulu': 688, 'cours': 689, 'soin': 690, 'obstacles': 691, 'affaires':
692, 'directeur': 693, "d'arrestation": 694, 'tenir': 695, 'comprendre': 696,
'vivement': 697, 'trouver': 698, 'indienne': 699, 'courait': 700, 'entrait':
701, "l'intérieur": 702, 'hors': 703, 'chef': 704, 'colère': 705, "d'autres":
706, 'troupe': 707, 'victime': 708, 'reconnaissance': 709, 'chance': 710,
'novembre': 711, 'américains': 712, 'patron': 713, "l'union": 714, 'rail-road':
715, '7': 716, "l'angleterre": 717, 'haute': 718, 'temple': 719, 'riche': 720,
'manquait': 721, 'quelquefois': 722, 'jouer': 723, 'connaissait': 724, 'minuit':
725, 'donné': 726, 'première': 727, 'bruit': 728, 'donnait': 729, 'force': 730,
'devaient': 731, 'arriva': 732, 'échapper': 733, 'cinquante-cinq': 734, 'tel':
735, 'fond': 736, 'bureaux': 737, 'brindisi': 738, 'succès': 739, 'mis': 740,
'manquer': 741, 'cartes': 742, 'est-ce': 743, 'parce': 744, 'allahabad': 745,
'railway': 746, 'faute': 747, 'projet': 748, 'surpris': 749, 'évident': 750,
"s'arrêta": 751, 'suivit': 752, 'pauvre': 753, 'inutile': 754, 'premiers': 755,
'parut': 756, 'états-unis': 757, 'vieux': 758, 'prenait': 759, 'prenant': 760,
'soixante': 761, 'charbon': 762, 'doit': 763, 'rive': 764, 'passager': 765,
'bureau': 766, 'laissait': 767, 'poste': 768, 'décidément': 769, 'agent': 770,
'ton': 771, 'reste': 772, 'gange': 773, 'tracé': 774, 'marché': 775,
'tranquillement': 776, 'regardant': 777, 'l': 778, 'chargé': 779, 'attendre':
780, 'religion': 781, 'dernière': 782, 'fumée': 783, 'veille': 784,
'heureusement': 785, 'éléphant': 786, 'pillaji': 787, 'dès': 788, 'répétait':
789, 'descendre': 790, 'dieu': 791, 'obadiah': 792, 'greffier': 793, 'digne':
794, 'brise': 795, 'rapidité': 796, 'japonais': 797, 'batulcar': 798, 'dollars':
799, 'sinon': 800, 'but': 801, 'certaine': 802, 'surface': 803, 'courant': 804,
'esprit': 805, 'tient': 806, 'disposition': 807, 'toile': 808, 'apparut': 809,
'mes': 810, 'gauche': 811, 'cheveux': 812, 'épaules': 813, "jusqu'alors": 814,
'apprit': 815, "d'en": 816, 'confortable': 817, 'parfaitement': 818, 'cher':
819, 'beau': 820, 'dirigea': 821, 'flanagan': 822, 'amérique': 823, 'saurait':
824, 'gardes': 825, "l'ordre": 826, 'métropolitaine': 827, 'manière': 828,
'rails': 829, 'cause': 830, 'offrit': 831, 'allez': 832, 'rester': 833, 'donna':
834, 'aperçut': 835, 'itinéraire': 836, 'quittant': 837, 'considérable': 838,
'traverser': 839, 'mauvaise': 840, 'telle': 841, 'nombreux': 842, 'plupart':
843, 'nécessaire': 844, 'derrière': 845, 'serviteur': 846, 'venez': 847, '20':
848, '11': 849, 'a-t-il': 850, 'premières': 851, "l'horizon": 852, 'curieux':
853, 'bundelkund': 854, 'compromettre': 855, 'poing': 856, 'sentait': 857,
'beaucoup': 858, 'forêt': 859, "l'indien": 860, 'bête': 861, 'cou': 862,
'grandes': 863, 'ferait': 864, 'bonne': 865, 'sauver': 866, 'décidé': 867,
'rivière': 868, 'atteint': 869, 'scène': 870, 'plaine': 871, 'convenu': 872,
'chinois': 873, 'prudent': 874, 'libre': 875, 'omaha': 876, 'smyth': 877,
'mudge': 878, 'société': 879, 'sûr': 880, 'frères': 881, 'parlait': 882, 'dû':
883, 'promenait': 884, "c'étaient": 885, 'noir': 886, 'souliers': 887, 's': 888,
'habitude': 889, 'royaume-uni': 890, 'mercredi': 891, 'chapeau': 892, 'foi':
893, 'haut': 894, 'dents': 895, 'font': 896, "qu'en": 897, 'allant': 898,
'entendu': 899, 'pouvoir': 900, 'circonstances': 901, 'réglementaire': 902,

'sûreté': 903, 'faisaient': 904, 'partenaires': 905, 'sullivan': 906, 'samuel': 907, 'thomas': 908, 'observer': 909, "l'argent": 910, 'passant': 911, "d'or": 912, 'mission': 913, 'espérer': 914, 'singulièrement': 915, 'ajouta-t-il': 916, 'mois': 917, 'indous': 918, 'sérieux': 919, 'volontiers': 920, 'monta': 921, "l'esprit": 922, 'sauta': 923, "s'approcha": 924, 'coin': 925, 'émotion': 926, 'femmes': 927, 'longs': 928, "d'eux": 929, 'avance': 930, "j'en": 931, 'rade': 932, "s'écria-t-il": 933, 'visa': 934, 'vais': 935, 'pressé': 936, 'aura': 937, "n'aurait": 938, 'feux': 939, 'péninsule': 940, 'commerce': 941, 'auraient': 942, 'bénarès': 943, 'enchanté': 944, 'retrouver': 945, 'ceinture': 946, 'base': 947, 'palanquin': 948, 'dos': 949, 'qualité': 950, 'seule': 951, 'air': 952, 'plein': 953, 'sauvages': 954, "s'arrêtait": 955, 'contrée': 956, 'manquaient': 957, 'séparent': 958, 'présence': 959, 'donner': 960, 'kiouni': 961, 'tenter': 962, "d'allahabad": 963, 'gros': 964, 'devint': 965, 'sacrifice': 966, 'veuve': 967, 'brûler': 968, 'sachant': 969, 'portes': 970, 'bande': 971, 'désappointement': 972, 'hauteur': 973, 'montrait': 974, 'prison': 975, 'policeman': 976, 'manqué': 977, 'dut': 978, 'chacun': 979, "l'amérique": 980, 'craindre': 981, 'circonstance': 982, "l'arrière": 983, 'ruiné': 984, 'désert': 985, 'japonaise': 986, 'ai': 987, 'pacific': 988, 'meeting': 989, 'mormon': 990, 'beaux': 991, 'docks': 992, 'nombreuses': 993, 'était-il': 994, 'mathématiquement': 995, 'connaissance': 996, 'silence': 997, 'suffisait': 998, 'servir': 999, '2': 1000, 'rendu': 1001, "l'eau": 1002, 'marcher': 1003, 'montra': 1004, 'tranquille': 1005, "d'argent": 1006, 'entendit': 1007, 'soigneusement': 1008, 'taille': 1009, 'front': 1010, 'plutôt': 1011, 'flegmatique': 1012, "l'expression": 1013, 'ami': 1014, 'commença': 1015, 'gaz': 1016, 'lumière': 1017, 'joie': 1018, 'date': 1019, 'fenêtres': 1020, 'arbres': 1021, 'couvert': 1022, 'remit': 1023, 'dîner': 1024, 'joueurs': 1025, 'fallentin': 1026, 'sérieusement': 1027, 'accompli': 1028, 'répondre': 1029, 'magnifique': 1030, 'etc.': 1031, 'partance': 1032, 'continent': 1033, 'facile': 1034, '13': 1035, '6': 1036, 'commençait': 1037, 'employé': 1038, 'quinze': 1039, 'adversaires': 1040, 'valeur': 1041, 'vingt-cinq': 1042, 'voyant': 1043, 'partait': 1044, 'oublié': 1045, 'voitures': 1046, "l'extrémité": 1047, 'noire': 1048, 'tombait': 1049, 'bec': 1050, 'uns': 1051, 'autrement': 1052, 'sentit': 1053, 'retards': 1054, 'compromis': 1055, 'entra': 1056, 'incident': 1057, 'celles': 1058, 'péninsulaire': 1059, 'vif': 1060, 'certaines': 1061, 'savez-vous': 1062, 'arrêter': 1063, 'voit': 1064, "l'horloge": 1065, 'réflexion': 1066, 'pourtant': 1067, 'demande': 1068, 'raconta': 1069, 'grosse': 1070, 'tiens': 1071, 'quart': 1072, 'franchir': 1073, 'détroit': 1074, 'paraître': 1075, 'révérend': 1076, 'naturel': 1077, 'célèbre': 1078, 'mers': 1079, 'population': 1080, 'branches': 1081, "l'aspect": 1082, 'résolut': 1083, 'fureur': 1084, 'vigoureux': 1085, 'jeté': 1086, "l'eût": 1087, 'européenne': 1088, 'forêts': 1089, 'convoi': 1090, "c'est-à-dire": 1091, 'descendit': 1092, 'fin': 1093, 'amené': 1094, 'lit': 1095, 'atteindre': 1096, 'prononcer': 1097, 'reconnut': 1098, 'portaient': 1099, 'soie': 1100, 'lever': 1101, "l'ivresse": 1102, 'bûcher': 1103, 'au-dehors': 1104, 'vallée': 1105, 'charmante': 1106, 'prend': 1107, 'besoin': 1108, 'dévouement': 1109, 'chinoise': 1110, 'surprise': 1111, 'faites': 1112, 'voulaient': 1113, "l'océan": 1114, 'lames': 1115, 'steamers': 1116, 'visage': 1117, 'marin': 1118, 'malheureux': 1119, "l'embarcation": 1120, 'direction': 1121, 'certes': 1122, 'salé': 1123,

"l'utah": 1124, 'portant': 1125, 'impassible': 1126, 'bourse': 1127,
"n'avaient": 1128, "l'institution": 1129, 'baring': 1130, 'régulièrement': 1131,
'partout': 1132, 'utile': 1133, "d'autant": 1134, 'possédait': 1135, 'endroit':
1136, "l'honneur": 1137, 'connaître': 1138, 'importante': 1139, 'remarquer':
1140, 'combat': 1141, 'parents': 1142, 'amis': 1143, 'vérité': 1144, 'coucher':
1145, 'admirable': 1146, "d'amérique": 1147, 'excentrique': 1148, 'james': 1149,
'degrés': 1150, 'assis': 1151, 'resté': 1152, 'devenu': 1153, 'voulant': 1154,
'd': 1155, 'demeura': 1156, 'convaincu': 1157, 'magnifiques': 1158, 'repos':
1159, "l'oeil": 1160, 'immobile': 1161, 'sang-froid': 1162, 'court': 1163,
'geste': 1164, 'bonnes': 1165, 'exercices': 1166, 'servi': 1167,
'communication': 1168, 'quittait': 1169, 'moyenne': 1170, 'véritable': 1171,
'millions': 1172, 'ports': 1173, 'public': 1174, 'quelconque': 1175, 'alla':
1176, 'agents': 1177, 'confiance': 1178, 'folie': 1179, 'tournant': 1180,
'moitié': 1181, 'vingtaine': 1182, 'présentait': 1183, "n'avons": 1184, 'bas':
1185, 'tomba': 1186, 'mesure': 1187, 'audacieux': 1188, 'certains': 1189,
'points': 1190, 'voyageur': 1191, 'lignes': 1192, 'course': 1193, 'dépêche':
1194, 'trait': 1195, 'impatience': 1196, 'quelles': 1197, 'consulaire': 1198,
'reconnaître': 1199, 'race': 1200, 'longue': 1201, 'treize': 1202, 'savez':
1203, 'coque': 1204, 'passeports': 1205, 'penser': 1206, 'sortit': 1207,
'recherche': 1208, 'rentra': 1209, 'perte': 1210, 'visiter': 1211, 'superbe':
1212, "d'arriver": 1213, 'arrivés': 1214, 'revenir': 1215, 'filait': 1216,
'marchait': 1217, 'pointe': 1218, 'ceux-ci': 1219, 'indigènes': 1220,
'gouverneur': 1221, 'disparaissaient': 1222, 'houle': 1223, 'accident': 1224,
'songeait': 1225, 'villes': 1226, 'bords': 1227, 'occupait': 1228, 'vastes':
1229, 'franchit': 1230, 'nord-ouest': 1231, 'grave': 1232, 'ingénieurs': 1233,
'battait': 1234, 'rangs': 1235, 'grâce': 1236, 'heureux': 1237, 'quitte': 1238,
'renversé': 1239, 'comprend': 1240, 'époque': 1241, 'anglaises': 1242,
'transport': 1243, "l'île": 1244, 'nord-est': 1245, 'malebar-hill': 1246,
'accepta': 1247, 'voulut': 1248, 'ordre': 1249, 'comprit': 1250, 'oreilles':
1251, 'jambes': 1252, 'indien': 1253, 'située': 1254, 'petits': 1255, 'groupes':
1256, 'projets': 1257, 'croyait': 1258, 'méridien': 1259, 'halte': 1260,
'défaut': 1261, "d'aller": 1262, 'suivait': 1263, 'bout': 1264, 'ruine': 1265,
'debout': 1266, 'sommeil': 1267, 'blanche': 1268, 'sutty': 1269, 'moindre':
1270, 'influence': 1271, "l'opium": 1272, 'conséquent': 1273, 'voyez': 1274,
'humaine': 1275, 'suprême': 1276, 'terrible': 1277, 'bonheur': 1278, 'songer':
1279, 'inquiétude': 1280, 'bâtie': 1281, 'blanches': 1282, 'satisfaction': 1283,
'dame': 1284, 'prisonniers': 1285, 'serons': 1286, "s'ouvrit": 1287,
'semblaient': 1288, 'pleine': 1289, 'aperçu': 1290, 'attendu': 1291, 'complice':
1292, 'avons': 1293, "l'itinéraire": 1294, 'deviné': 1295, 'blanc': 1296,
'lame': 1297, "l'équipage": 1298, 'passerelle': 1299, 'directe': 1300,
'séparer': 1301, 'inutilement': 1302, 'inspecteur': 1303, 'américaine': 1304,
'brumes': 1305, 'position': 1306, 'barre': 1307, 'douane': 1308, 'fils': 1309,
'william': 1310, 'general-grant': 1311, 'éprouva': 1312, 'sacramento': 1313,
'revolver': 1314, 'stamp': 1315, 'w.': 1316, 'express': 1317, 'mormons': 1318,
'chassés': 1319, 'platte-river': 1320, 'passerelles': 1321, 'prophète': 1322,
'chicago': 1323, 'bordeaux': 1324, 'queenstown': 1325, "l'année": 1326, 'banc':
1327, 'lesquels': 1328, 'crédit': 1329, 'ouvert': 1330, 'invariablement': 1331,
'au-delà': 1332, 'probable': 1333, 'indiquait': 1334, 'voyager': 1335, 'années':

1336, 'jouait': 1337, 'vivait': 1338, 'convenir': 1339, 'habitudes': 1340,
'congé': 1341, 'forster': 1342, 'présenter': 1343, 'pendule': 1344, 'secondes':
1345, 'tirer': 1346, "j'étais": 1347, 'convient': 1348, 'connaissez': 1349,
'tirant': 1350, "n'importe": 1351, 'suffit': 1352, 'rue': 1353, 'sortait': 1354,
'achevé': 1355, 'merveilleusement': 1356, "l'idée": 1357, 'animaux': 1358,
'mouvements': 1359, 'cherché': 1360, 'hautes': 1361, 'teint': 1362, 'jeunesse':
1363, 'admirablement': 1364, 'chercher': 1365, 'lord': 1366, 'cheminée': 1367,
'comprenait': 1368, 'détails': 1369, 'prévu': 1370, 'saison': 1371, 'vêtements':
1372, 'eussent': 1373, "l'une": 1374, "s'accomplit": 1375, 'morning': 1376,
'chronicle': 1377, 'demi-heure': 1378, 'riches': 1379, "l'auteur": 1380,
'francs': 1381, 'réponse': 1382, 'liasse': 1383, 'principal': 1384, 'paraît':
1385, 'merci': 1386, 'meilleurs': 1387, 'passèrent': 1388, 'reconnu': 1389,
'arrivée': 1390, 'faveur': 1391, 'vite': 1392, 'avouer': 1393, 'ensemble': 1394,
'500': 1395, "aujourd'hui": 1396, 'sonnaient': 1397, 'démesurément': 1398,
'faillit': 1399, 'oh': 1400, 'poliment': 1401, 'prirent': 1402, 'abasourdi':
1403, 'cri': 1404, 'moyens': 1405, 'traité': 1406, 'compter': 1407,
"d'attendre": 1408, 'masse': 1409, 'resta': 1410, 'rappela': 1411, 'hélice':
1412, 'chevaux': 1413, 'réglementaires': 1414, 'bourgade': 1415, 'britannique':
1416, 'oeil': 1417, 'venant': 1418, 'hier': 1419, 'sens': 1420, 'comprenez':
1421, 'populaire': 1422, 'aden': 1423, 'combustible': 1424, "l'étranger": 1425,
'réfléchir': 1426, "l'intention": 1427, "l'atlantique": 1428, 'plaira': 1429,
'traces': 1430, 'frappait': 1431, 'attentivement': 1432, 'droits': 1433, '5':
1434, 'pensait': 1435, 'obligé': 1436, 'semble': 1437, "d'heure": 1438, 'entré':
1439, 'emportée': 1440, 'hâte': 1441, 'aucunement': 1442, 'naturellement': 1443,
'emplettes': 1444, 'repris': 1445, 'veut': 1446, 'trompe': 1447, 'lança': 1448,
'traverse': 1449, 'civils': 1450, 'officiers': 1451, 'présent': 1452, 'simples':
1453, '12': 1454, 'allaient': 1455, 'purser': 1456, 'viande': 1457, 'chants':
1458, 'rendait': 1459, 'temples': 1460, 'fakirs': 1461, 'tigres': 1462,
'champs': 1463, '14': 1464, 'seize': 1465, 'palmiers': 1466, 'îles': 1467,
'quais': 1468, 'rajahs': 1469, 'terribles': 1470, 'occupé': 1471, 'modifier':
1472, 'légèrement': 1473, 'tours': 1474, 'sectateurs': 1475, 'négociants': 1476,
'peuvent': 1477, 'soudain': 1478, 'précipitèrent': 1479, 'délit': 1480,
'auprès': 1481, 'âge': 1482, 'terrestre': 1483, 'produit': 1484, 'traversé':
1485, 'descend': 1486, "l'aventure": 1487, 'retomba': 1488, 'rouges': 1489,
'sortes': 1490, 'déesse': 1491, 'sang': 1492, 'empêcher': 1493, 'fleuve': 1494,
'regarda': 1495, "s'élança": 1496, 'annoncé': 1497, 'transporter': 1498,
'voulez': 1499, 'tôt': 1500, "d'avance": 1501, '25': 1502, 'trouvèrent': 1503,
"l'animal": 1504, 'soins': 1505, 'vendre': 1506, "d'agir": 1507, "s'agissait":
1508, 'vindhias': 1509, 'clown': 1510, 'aspect': 1511, 'plaines': 1512,
'bandes': 1513, 'firent': 1514, 'arbre': 1515, 'dernières': 1516, 'conduisit':
1517, 'fidèles': 1518, 'vêtus': 1519, 'char': 1520, 'larges': 1521, 'statue':
1522, 'signe': 1523, 'habits': 1524, 'robe': 1525, 'histoire': 1526, 'sortir':
1527, 'risquer': 1528, 'parvenir': 1529, 'arrivèrent': 1530, 'basse': 1531,
'interrompu': 1532, 'çà': 1533, 'ramena': 1534, 'attendit': 1535, "l'obscurité":
1536, 'contretemps': 1537, 'poings': 1538, "l'endroit": 1539, 'éclair': 1540,
'cerveau': 1541, 'purent': 1542, 'couché': 1543, "s'éleva": 1544, 'apparence':
1545, 'flèches': 1546, 'sépare': 1547, 'courut': 1548, 'céleste': 1549,
'poignée': 1550, 'regards': 1551, 'apparaissait': 1552, 'personnes': 1553,

'fixé': 1554, 'suivre': 1555, 'parbleu': 1556, 'celui-là': 1557, 'acte': 1558,
'condamnation': 1559, 'pavillon': 1560, 'mât': 1561, 'sauveur': 1562,
'définitivement': 1563, 'servait': 1564, 'île': 1565, 'équipage': 1566, 'poids':
1567, "l'europe": 1568, 'rire': 1569, 'fuir': 1570, 'regagner': 1571, 'marée':
1572, 'cabines': 1573, 'voile': 1574, 'bâtiments': 1575, 'tabagie': 1576,
'réservé': 1577, 'devait-il': 1578, 'passe': 1579, 'bâtiment': 1580, 'n°': 1581,
'embarcation': 1582, 'ailes': 1583, 'mangea': 1584, 'brume': 1585, 'violence':
1586, 'enlevé': 1587, 'solidement': 1588, 'trente': 1589, 'battre': 1590,
'indigène': 1591, 'conclu': 1592, 'tingou': 1593, 'case': 1594, 'banquettes':
1595, 'écoutait': 1596, 'adversaire': 1597, 'revolvers': 1598, 'kamerfield':
1599, 'mandiboy': 1600, 'détonations': 1601, 'irrésistible': 1602, 'hitch':
1603, 'mormonisme': 1604, 'saints': 1605, "l'elder": 1606, "l'américain": 1607,
'détachement': 1608, 'mariage': 1609, 'figurait': 1610, 'reine': 1611,
'négociant': 1612, "l'ouest": 1613, 'direct': 1614, 'principalement': 1615,
'membre': 1616, 'mystérieux': 1617, 'généreuse': 1618, 'cherchait': 1619,
'propos': 1620, 'perdus': 1621, 'lire': 1622, 'caractère': 1623, 'rentrait':
1624, 'toilette': 1625, 'état': 1626, 'vivre': 1627, 'salua': 1628, 'venu':
1629, 'exact': 1630, 'renseignements': 1631, 'constater': 1632, "l'écart": 1633,
'disparut': 1634, 'auxquelles': 1635, 'léger': 1636, 'pâle': 1637, 'degré':
1638, 'existence': 1639, 'parties': 1640, 'parfait': 1641, 'prêts': 1642,
'permettait': 1643, 'ému': 1644, 'habitait': 1645, 'métier': 1646, 'sec': 1647,
'aimable': 1648, 'têtes': 1649, 'présenta': 1650, 'propre': 1651, 'sévère':
1652, 'thé': 1653, 'vingt-trois': 1654, 'chaussures': 1655, 'lettres': 1656,
'guerre': 1657, 'régulier': 1658, 'placé': 1659, 'élevé': 1660, 'relevé': 1661,
'coupé': 1662, 'lecture': 1663, 'dura': 1664, 'repas': 1665, 'reparut': 1666,
"l'ingénieur": 1667, 'argent': 1668, "j'espère": 1669, 'principaux': 1670,
"d'embarquement": 1671, '1': 1672, 'papier': 1673, 'septembre': 1674, 'prise':
1675, 'caissier': 1676, 'horloge': 1677, 'sonna': 1678, 'détectives': 1679,
'voleurs': 1680, 'théâtre': 1681, 'exactement': 1682, 'douter': 1683,
'recherches': 1684, 'zèle': 1685, 'collègue': 1686, 'discussion': 1687,
'interrompue': 1688, 'reprenait': 1689, 'puisse': 1690, 'rapides': 1691,
'fuite': 1692, 'établi': 1693, 'railways': 1694, '3': 1695, '22': 1696, '9':
1697, "s'ils": 1698, 'dis': 1699, 'calculé': 1700, 'sauter': 1701, "s'être":
1702, 'actuellement': 1703, 'pareille': 1704, 'consciencieusement': 1705,
'malles': 1706, 'chemises': 1707, 'phrase': 1708, 'ferma': 1709, 'belles': 1710,
'descendirent': 1711, 'montèrent': 1712, 'tenant': 1713, 'nus': 1714,
'entrèrent': 1715, 'revenu': 1716, 'revoir': 1717, 'compartiment': 1718,
'pluie': 1719, 'guère': 1720, 'insensé': 1721, "d'après": 1722, 'relativement':
1723, 'neiges': 1724, "l'hiver": 1725, 'marcheurs': 1726, 'forcé': 1727,
'suivirent': 1728, 'cheval': 1729, 'orientale': 1730, 'canal': 1731,
'promenaient': 1732, 'marques': 1733, "d'impatience": 1734, 'nommait': 1735,
'commis': 1736, 'filer': 1737, 'dites': 1738, 'gens-là': 1739, 'travail': 1740,
'diverses': 1741, 'charge': 1742, 'entrer': 1743, 'possessions': 1744,
'criminel': 1745, 'pressentiment': 1746, 'longtemps': 1747, 'au-devant': 1748,
'vigoureusement': 1749, 'lut': 1750, 'involontaire': 1751, 'matière': 1752,
'formalité': 1753, '4': 1754, 'embarqué': 1755, 'dépensées': 1756, 'inscrivit':
1757, 'gain': 1758, 'parle': 1759, 'rêve': 1760, 'fameuse': 1761, 'vitres':
1762, 'regrette': 1763, 'partis': 1764, 'causait': 1765, 'tort': 1766, 'quarts':

1767, 'produire': 1768, 'prétexte': 1769, 'diable': 1770, 'conviction': 1771,
"jusqu'aux": 1772, 'fonctionnaires': 1773, 'proprement': 1774, 'dite': 1775,
'troupes': 1776, 'quelques-unes': 1777, 'rafale': 1778, 'imperturbable': 1779,
'scènes': 1780, 'dangers': 1781, 'parlé': 1782, 'emprisonné': 1783,
'rejoignait': 1784, 'sourire': 1785, 'pagodes': 1786, "l'occasion": 1787,
'murailles': 1788, 'desquelles': 1789, 'larmes': 1790, 'escale': 1791, "d'aden":
1792, 'établir': 1793, "d'atteindre": 1794, 'parsis': 1795, "d'européens": 1796,
'habitants': 1797, 'favorable': 1798, 'appuyé': 1799, 'dimanche': 1800,
'salcette': 1801, 'bengale': 1802, 'fondé': 1803, 'met': 1804, 'suit': 1805,
'indépendant': 1806, 'débarqué': 1807, 'sortant': 1808, 'crut': 1809, 'lapin':
1810, 'drôle': 1811, 'sacrés': 1812, "d'arrêt": 1813, 'refusa': 1814, 'liberté':
1815, 'robes': 1816, 'tam-tams': 1817, 'malheureusement': 1818, 'dirigeait':
1819, "l'entrée": 1820, 'pénétrer': 1821, 'laissé': 1822, 'releva': 1823,
'indou': 1824, "l'ombre": 1825, 'monter': 1826, 'monture': 1827, 'emportait':
1828, 'véritablement': 1829, 'globe': 1830, 'froide': 1831, 'hauts': 1832,
'couverts': 1833, 'probablement': 1834, 'retarder': 1835, 'campagne': 1836,
'affluents': 1837, 'sous-affluents': 1838, 'regardaient': 1839, 'funeste': 1840,
'cadavre': 1841, 'paire': 1842, 'babouches': 1843, 'jeter': 1844, 'occupaient':
1845, 'cru': 1846, "puisqu'il": 1847, 'sommes-nous': 1848, 'kholby': 1849,
'cinquantaine': 1850, 'furieux': 1851, 'vos': 1852, 'intérêts': 1853, "qu'au":
1854, 'travaux': 1855, "j'irai": 1856, 'découverte': 1857, 'animal': 1858,
'langue': 1859, 'admettant': 1860, 'acheter': 1861, 'cacolets': 1862, 'vivres':
1863, "l'épaisse": 1864, "s'ensuit": 1865, 'capricieuses': 1866, 'monts': 1867,
'coupant': 1868, 'allure': 1869, 'soumis': 1870, 'tantôt': 1871, 'riait': 1872,
'signal': 1873, "l'influence": 1874, 'singes': 1875, "qu'est-ce": 1876,
'principale': 1877, 'espérait': 1878, 'bassin': 1879, 'espace': 1880,
'procession': 1881, 'tint': 1882, 'pensa': 1883, 'feuillage': 1884,
'instruments': 1885, "d'hommes": 1886, 'groupe': 1887, 'légère': 1888, 'armés':
1889, 'fanatiques': 1890, 'fracas': 1891, 'disparu': 1892, 'supplice': 1893,
'chanvre': 1894, 'passera': 1895, "s'adressant": 1896, 'sourit': 1897,
"n'hésita": 1898, 'résolution': 1899, 'apercevoir': 1900, 'hurlements': 1901,
"s'avança": 1902, 'lueur': 1903, 'reposait': 1904, "s'élevait": 1905,
'veillaient': 1906, 'attendons': 1907, 'lune': 1908, 'parois': 1909, 'briques':
1910, 'contenir': 1911, "l'impassible": 1912, 'précipiter': 1913, 'basses':
1914, 'retentirent': 1915, 'sembla': 1916, 'rôle': 1917, 'enlèvement': 1918,
'précipités': 1919, 'songeant': 1920, 'brandy': 1921, "qu'après": 1922,
'partit': 1923, 'brahma': 1924, 'payé': 1925, 'donne': 1926, 'courageux': 1927,
'étonnement': 1928, 'valait': 1929, 'couleur': 1930, "s'arrêter": 1931,
'ferrée': 1932, 'venaient': 1933, 'avancée': 1934, 'milliers': 1935, 'loi':
1936, 'places': 1937, 'laissant': 1938, 'poursuivi': 1939, 'contenta': 1940,
'citoyen': 1941, 'sacrilège': 1942, 'retenu': 1943, 'caution': 1944,
"s'embarquer": 1945, 'occasion': 1946, 'rejoindre': 1947, 'portion': 1948,
'baie': 1949, 'spectacle': 1950, 'malacca': 1951, 'opérer': 1952, 'employer':
1953, 'étonné': 1954, 'marquait': 1955, 'singulier': 1956, 'explication': 1957,
'nos': 1958, 'région': 1959, 'parcouru': 1960, 'voilure': 1961, "l'appareil":
1962, 'continuait': 1963, 'croyez': 1964, 'prévenu': 1965, 'soupapes': 1966,
'serrées': 1967, 'tourmente': 1968, 'matelots': 1969, 'baromètre': 1970,
'instrument': 1971, 'prochain': 1972, 'trente-cinq': 1973, 'permission': 1974,

```
'continuer': 1975, "l'embouchure": 1976, 'antipodes': 1977, 'largement': 1978,
'levant': 1979, "jusqu'ici": 1980, "d'acheter": 1981, 'véhicule': 1982,
'bagages': 1983, 'terminé': 1984, 'tournait': 1985, 'tonneaux': 1986,
'nagasaki': 1987, 'brigantine': 1988, 'amener': 1989, 'espoir': 1990,
'changement': 1991, 'typhon': 1992, 'flanc': 1993, 'trompé': 1994, 'nourriture':
1995, 'bannières': 1996, 'pensa-t-il': 1997, 'représentation': 1998, 'longs-
nez': 1999, 'abandonnant': 2000, 'secours': 2001, '23': 2002, "l'instant": 2003,
'omnibus': 2004, "l'escalier": 2005, 'prononça': 2006, 'yankee': 2007,
"l'illinois": 2008, 'nebraska': 2009, 'colorado': 2010, 'branche': 2011,
'nevada': 2012, 'rocheuses': 2013, 'californie': 2014, 'prairies': 2015,
'tenté': 2016, 'portière': 2017, 'brigham': 2018, 'young': 2019, 'missouri':
2020, 'vivant': 2021, 'garde-voie': 2022, 'medicine-bow': 2023, 'marchant':
2024, 'pique': 2025, 'envie': 2026, 'pression': 2027, 'entendait': 2028,
"l'hudson": 2029, 'china': 2030, 'erreur': 2031, 'wilson': 2032, 'lundi': 2033}
```

Pour les approches qui travaillent sur des coocurrences, nous allons créer une matrice creuse contenant le nombre de fois où deux mots apparaissent ensemble dans une fenêtre glissante. Cette matrice prendra la forme d'un dictionnaire dont les clés sont des paires d'entiers (les identifiants des deux mots cooccurant, classés dans l'ordre des entiers pour neutraliser l'ordre des coocurrences), et les valeurs sont le nombre de coocurrences. Le principe est relativement simple. On parcourt le corpus, position par position. Puis il faut compter le nombre de fois où le premier mot de la fenêtre apparait avec chacun des autres mots de la fenêtre.

Attention, chaque fois que l'on essaie d'accéder à une paire dans cette matrice, une entrée est créée, même si elle est à zéro. Donc si on parcourt les occurences pour tout le vocabulaire, la matrice risque de prendre toute la mémoire. D'où l'importance dans la suite de ne travailler que sur les cooccurrences non nulles.

[11]:
```python
window_size = 10

cooc = collections.defaultdict(int)

for i in range(len(int_words) - window_size + 1):
  window = int_words[i: i + window_size]
  word = window[0]
  for other in window[1:]:
    if other > word:
      cooc[(word, other)] += 1
    elif other < word:
      cooc[(other, word)] += 1

print(len(cooc))
print(cooc[(vocab['de'], vocab['le'])])
```

```
157189
1554
```

0.1.3 Modèle LSA par factorisation

La première méthode que nous allons utiliser est LSA. L'idée est d'effectuer une décomposition en valeurs propres de la matrice de coocurrences à l'aide d'une SVD. Soit la matrice de coocurrence A de taille (n, n), la SVD produit 3 matrices U, Σ et V de taille (n, n). Σ une matrice diagonale contenant les valeurs propres ordonnées par valeur décroissante. U et V sont des bases de l'espace produit.

$A = U\Sigma V^\top$

Si on met à zero les éléments les plus petits de Σ, on obtient une approximation de A de rang inférieur. Cela revient à mettre à zéro dans U les colonnes correspondant et dans $V^\top$ les lignes correspondant. Nous utiliserons les 50 premières dimensions de la base U comme embeddings, ignorant les autres valeurs propres.

Il existe de nombreuses variantes pour la création de la matrice de coocurrences, comme l'utilisation de documents, de phrases, d'un sous vocabulaire, comme 2nde dimension de la matrice. Lorsque LSA a été proposé, il était admis que les premières valeurs propres correspondaient à des thèmes représentés dans le corpus (si on ignore les mots outils). Si la matrice de cooccurrence n'est pas carrée, il faut utiliser l'argument `full_matrices` de `svd()`.

Notez que la SVD est calculée sur la matrice pleine. C'est très coûteux est pas raisonnable lorsque l'on utilise un vocabulaire réaliste comme dans wikipedia (plusieurs centaines de milliers de mots différents). On peut utiliser `sklearn.decomposition.TruncatedSVD` pour ne calculer que les k valeurs propres les plus grandes, sur des matrices creuses.

[12]:
```
# note: reduce to 2000 if it doesn't fit memory
size = min(2000, len(vocab))

A = np.zeros((size, size), dtype=np.float32)
## à compléter :
### remplir la matrice A avec le contenu de cooc ##

U, S, Vt = np.linalg.svd(A)

print(S[:50])

U = U[:,:50]
print(U.shape)
```

```
[5936.308    1989.2228   1112.6202    889.06934   676.9641    599.85065
  547.76697   462.71518   401.22665   380.71002   331.3185    288.7164
  259.5473    245.29692   212.80443   206.39507   182.4104    172.54489
  163.1707    151.9385    146.26186   137.49902   131.39679   130.33217
  122.835144  120.01087   116.78016   111.22374   106.239975   99.76463
   95.162315   92.741745   89.19979    87.59835    85.443436   84.455734
   80.12914    76.58719    75.69645    73.734474   72.627335   71.29623
   70.37158    69.819115   68.11927    66.77544    65.56111    63.454063
   60.71173    60.00937 ]
(2000, 50)
```

0.1.4 Embeddings créés

Une fois les embeddings entraînés, on peut calculer la similarité entre les représentations associées à deux mots. Pour cela, on utilisera la similarité cosinus, c'est à dire l'angle entre deux vecteurs, qui peut être calculée en divisant le produit scalaire des deux vecteurs par le produit de leur norme.

$cosine(a, b) = \frac{\langle a,b \rangle}{||a||_2||b||_2}$

[13]:
```
def similarity(embedding, word1, word2):
  if word1 in vocab and word2 in vocab:
    v1 = embedding[vocab[word1]]
    v2 = embedding[vocab[word2]]
    return np.dot(v1.T,v2.T)/(np.linalg.norm(v1)*np.linalg.norm(v2))
    #return sklearn.metrics.pairwise.cosine_similaritycosine_similarity(v1,␣
 ↪v2)[0][0]

print('cosine(rues, ville) =', similarity(U, 'rues', 'ville'))
print('cosine(tour, monde) =', similarity(U, 'tour', 'monde'))
```

```
cosine(rues, ville) = 0.76411384
cosine(tour, monde) = 0.7166353
```

On peut aussi s'intéresser aux voisin d'un mot dans l'espace d'embeddings. Pour cela il faut calculer la similarité cosine de son vecteur avec tous les autres mots du vocabulaire. Numpy nous donne ensuite les n mots les plus similaires.

On utilisera la bibliothèque `sklearn`, qui offre une implémentation plus efficace de la simlarité du cosinus applicable à des tenseurs.

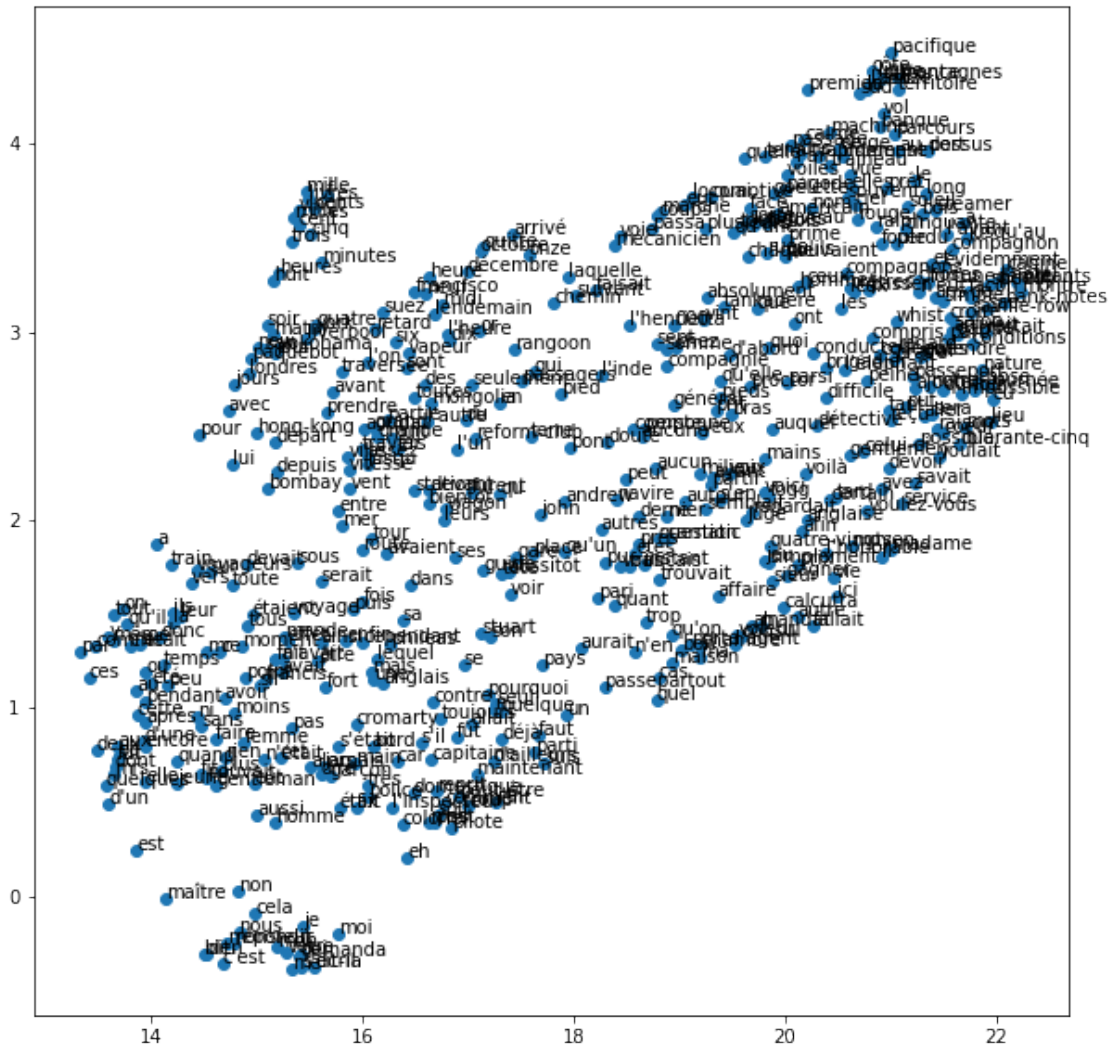
[18]:
```
reverse_vocab = {j: i for i, j in vocab.items()}

def most_similar(embedding, word, n=10):
  if word in vocab:
    v = embedding[vocab[word]].reshape(1, -1)
    scores = sklearn.metrics.pairwise.cosine_similarity(v, embedding).
 ↪reshape(-1)
    result = []

    # argsort gives n-best scores
    for i in reversed(scores.argsort()[-n:]):
      result.append((reverse_vocab[i], scores[i]))
    return result

print(most_similar(U, 'monde'))
print(most_similar(U, 'pays'))
```

```
[('monde', 0.9999999), ('pari', 0.7939632), ('voyage', 0.76716936), ('fait',
0.76655555), ('collègues', 0.7385684), ('tour', 0.7166352), ('londres',
0.68741137), ('cet', 0.6724174), ('transport', 0.67084193), ("j'ai", 0.6619778)]
[('pays', 1.0), ('domestique', 0.8715195), ('pris', 0.861529), ('car',
```

```
0.8595866), ("n'est", 0.8490747), ('rangoon', 0.8473806), ('passagers',
0.8455054), ('question', 0.8428835), ('sieur', 0.83760107), ('place',
0.8280313)]
```

Enfin, on peut tenter de visualiser l'espace d'embedding entier. Pour cela, on peut projeter les vecteurs en 50 dimensions vers le plan à l'aide d'une méthode de projection qui conserve les voisinages. La PCA est fidèle mais difficile à interpréter. La transformation t-SNE est très populaire (https://lvdmaaten.github.io/tsne/) – elle s'attache spécifiquement à conserver les voisinages locaux au détriment des voisinages globaux. Mais cette projection ne représente pas toujours fidèlement les données (https://pageperso.lis-lab.fr/benoit.favre/understanding-tsne). Un concurrent un peu plus rapide mais soumis aux mêmes problèmes est la projection umap. Qu'observe-t-on ?

[16]:
```python
# change size of figure
plt.rcParams['figure.dpi'] = 72
plt.rcParams['figure.figsize'] = [10, 10]

def plot_projection(embedding, n=500):
  projection = umap.UMAP().fit_transform(embedding[:n])
  fig, ax = plt.subplots()
  ax.scatter(x=projection[:,0], y=projection[:,1])

  for word, index in vocab.items():
    if index < n:
      ax.annotate(word, projection[index])
  plt.show()

plot_projection(U)
```



0.1.5 Création d'embeddings par modèle de langage

Une autre façon de créer des embeddings consiste à créer un modèle de langage prédictif qui essaie de prédire le prochain mot en fonction d'une fenêtre de mots précédants. Cette idée a été proposée par Collobert, Weston et al. Ici, nous allons implémenter une version simplifiée de ce modèle, inspirée du tutoriel pytorch sur les word embeddings

Nous allons d'abord créer une série d'exemples d'entraînement composés d'un n-gramme et d'un mot à prédire.

[140]:
```
context_size = 2
embedding_dim = 10
batch_size = 128

ngrams = [
```

```
    ([words[i - j - 1] for j in range(context_size-1,-1,-1)], words[i])
    for i in range(context_size, len(words))
]

ngrams[:4]
```

[140]:
```
[(['LE', 'TOUR'], 'DU'),
 (['TOUR', 'DU'], 'MONDE'),
 (['DU', 'MONDE'], 'EN'),
 (['MONDE', 'EN'], 'QUATRE-VINGTS')]
```

La prochaine étape consiste à transformer ces mots en indices d'entiers. Au fait, nous avons déjà une version du texte au format d'indices entiers de mots dans le vocabulaire, il suffit de l'utiliser à la place du texte pour créer les n-grammes :

[141]:
```
int_ngrams = [
    ([int_words[i - j - 1] for j in range(context_size-1,-1,-1)], int_words[i])
    for i in range(context_size, len(int_words))
]
int_ngrams[:4]

X = torch.tensor(list(map(lambda x: x[0],int_ngrams )))
Y = torch.tensor(list(map(lambda x: x[1],int_ngrams )))

print(X[:10])
print(X.shape)
print(X.type())

print(Y[:10])
print(Y.shape)
print(Y.type())
```

```
tensor([[  1, 180],
        [180,   8],
        [  8, 125],
        [125,   7],
        [  7, 302],
        [302,  83],
        [ 83,  31],
        [ 31,  14],
        [ 14, 140],
        [140,  27]])
torch.Size([56970, 2])
torch.LongTensor
tensor([  8, 125,   7, 302,  83,  31,  14, 140,  27,  11])
torch.Size([56970])
torch.LongTensor
```

Le modèle prédictif est assez simple : il s'agit tout simplement d'un réseau MLP (dense) qui prend en entrée les embeddings des trois mots du contexte et qui essaie de prédire une représentation one-hot du mot de sortie. Il contient deux couches linéaires avec une non-linéarité ReLU entre les deux, et avec un softmax à la sortie.

Ce modèle en soi n'est pas très intéressant. Cependant, les embeddings initialisés aléatoirement pour les mots de l'entrée seront appris de façon a encoder de l'information utile du contexte et, par conséquent, du sens des mots.

[170]:
```
class NGramLanguageModeler(nn.Module):

    def __init__(self, vocab_size, embedding_dim, context_size):
        super(NGramLanguageModeler, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.linear1 = nn.Linear(context_size * embedding_dim, 128)
        self.linear2 = nn.Linear(128, vocab_size)

    def forward(self, x):
        embeds = self.embeddings(x)
        # Flatten/concatenate embeddings
        embeds = embeds.view((-1, 1, x.shape[1] * embeds.shape[2]))
        out = F.relu(self.linear1(embeds))
        out2 = self.linear2(out)
        log_probs = F.log_softmax(out2, dim=2)
        return log_probs.view(-1,log_probs.shape[2])

model = NGramLanguageModeler(len(vocab), embedding_dim, context_size)
```

On peut maintenant écrire la fonction d'apprentissage, qui est assez classique. Comme nous ne nous intéressons pas au résultat de la tâche de prédiction, il n'est pas nécessaire de calculer une performance sur un jeu de validation. Si l'apprentissage marche, alors la loss doit diminuer…

[171]:
```
train_set = TensorDataset(X, Y)
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)

def fit(model, epochs, train_loader):
    criterion = nn.NLLLoss()
    optimizer = optim.Adam(model.parameters())
    for epoch in range(epochs):
        total_loss = num = 0
        for x, y in train_loader:
            optimizer.zero_grad()
            y_scores = model(x)
            #print(x[0],y[0])
            #print(y_scores[0])
            loss = criterion(y_scores, y)
            loss.backward()
            optimizer.step()
```

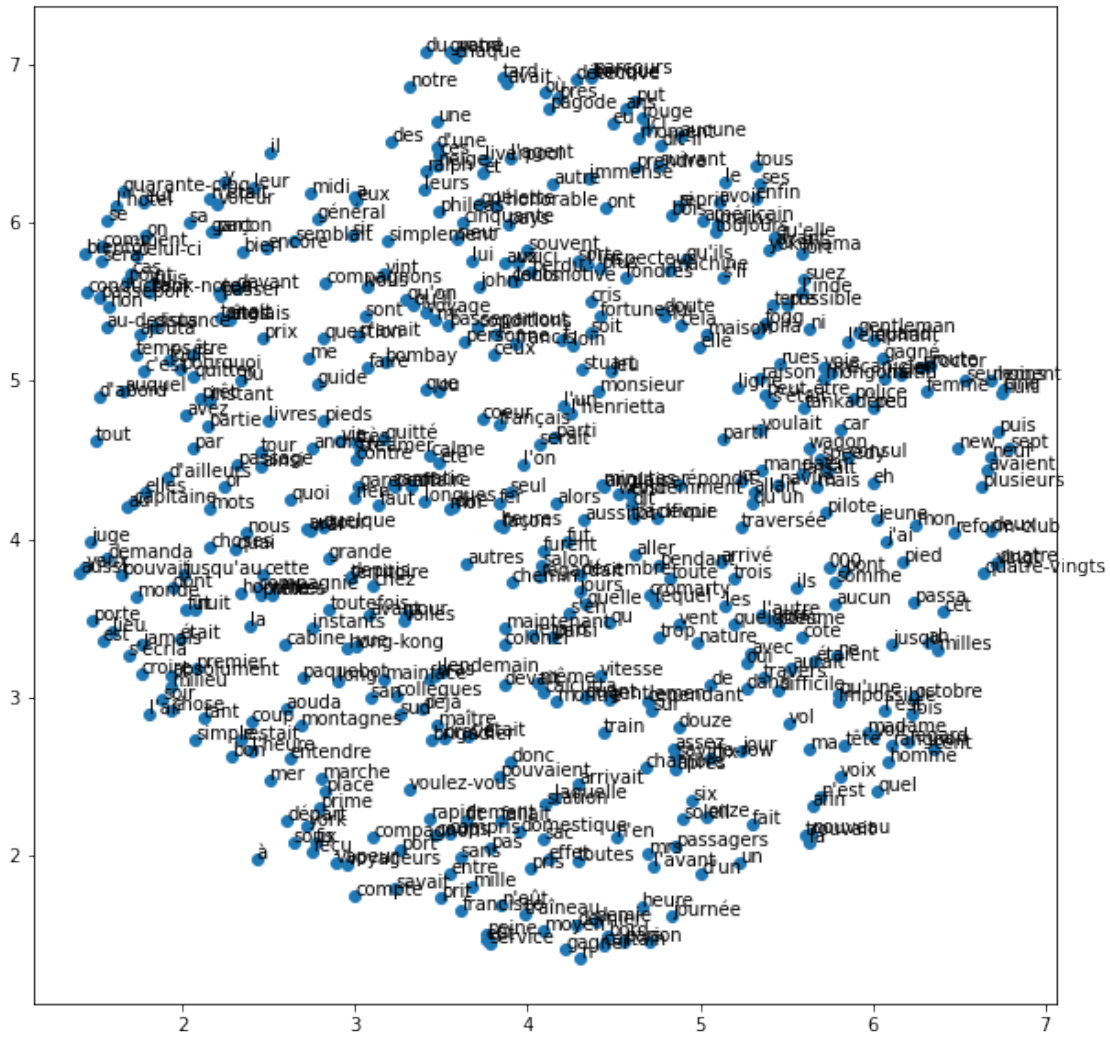
```
            total_loss += loss.item()
            num += len(y)
        print(epoch, total_loss / num)

fit(model,20,train_loader)
```

```
0 0.04747386990628034
1 0.04271675887098642
2 0.04044741562255249
3 0.03874138564588062
4 0.037504198401774026
5 0.03648748480271347
6 0.03568980530602735
7 0.034970302675530765
8 0.034355288542047266
9 0.033764942273646674
10 0.03324328264856163
11 0.032785866096058665
12 0.03231927813700213
13 0.03189040638594036
14 0.03150282537223625
15 0.031136398548114837
16 0.030793058313527442
17 0.030485181936532213
18 0.030166515032617757
19 0.029870785481682448
```

On peut maintenant extraire les embeddings de la première couche du réseau, et les inspecter comme on a fait ci-dessus.

[178]:
```
wordembeddings = model.embeddings.weight.detach().numpy()
#print(model.embeddings.weight[word_to_ix["beauty"]])
print('cosine(rues, ville) =', similarity(wordembeddings, 'rues', 'ville'))
print('cosine(tour, monde) =', similarity(wordembeddings, 'tour', 'monde'))
print(most_similar(wordembeddings, 'monde'))
print(most_similar(wordembeddings, 'pays'))
plot_projection(wordembeddings)
```

```
cosine(rues, ville) = 0.119806595
cosine(tour, monde) = 0.040541694
[('monde', 1.0), ('rester', 0.88376886), ('lieu', 0.86913365), ('dont',
0.8097212), ('brahma', 0.7896104), ('gouvernement', 0.7837359), ('amené',
0.77097934), ('attendit', 0.7709578), ("s'écria", 0.7635375), ('tourmente',
0.7626941)]
[('pays', 1.0), ('avancée', 0.8295889), ('phrase', 0.8286575), ('autre',
0.8084138), ('serviteur', 0.8047546), ('malebar-hill', 0.8039957), ('chapeau',
0.78077275), ('changement', 0.76721966), ("l'affaire", 0.7588147), ('perdu',
0.7560301)]
```



Un entraînement d'une vraie représentation utile et réutilisable demanderait un corpus beaucoup plus grand, une bonne optimisation de l'opération de softmax (p.ex. softmax hiérarchique), qui est très chère. C'est une des astuces employées par word2vec dans son implémentation de référence.

0.2 Exercice

Word2vec est un modèle similaire à celui ci-dessus, mais avec quelques modifications :

- Au lieu de prédire le prochain mot, il essaie de prédire le mot au milieu d'une fenêtre de k mots avant/après (CBOW)
- Dans sa version plus optimisée, il essaie de prédire si, oui ou non (0 ou 1) une paire mot-contexte est une vraie paire observée ou une paire aléatoirement générée (negative sampling)

Implémentez les deux algorithmes, skipgram et cbow, et inspectez les embeddings obtenus comme ci-dessus. Vous pouvez vous inspirer de ce tutoriel pour vous aider : https://github.com/jojonki/word2vec-pytorch/blob/master/word2vec.ipynb

0.3 Pour aller plus loin : GLove

Nous allons maintenant entraîner des plongements lexicaux avec la méthodes "GloVe" de Pennington et al. Une implémentation est disponible (https://nlp.stanford.edu/projects/glove/), mais pour l'exercice, nous allons la refaire à l'aide des équations décrites dans l'article décrivant l'approche. Dans ce dernier, l'équation (8), décrit la fonction de loss minimisée pour obtenir les embeddings (réécrite pour coller à notre contexte):

$J = \sum_{ij} f(\text{cooc}_{ij})(w_i^\top w_j + b_i + b_j - \log \text{cooc}_{ij})^2$

Dans cette équation, i et j sont des indentifiants de mots, w_i et w_j sont les embeddings associés, par exemple des vecteurs de dimension 50, b_i, b_j sont des biais scalaires, cooc_{ij} est le nombre de cooccurrences entre les mots i et j, et f est une fonction de pondération. En particulier,

$$f(x) = \begin{cases} 0, & \text{si } x = 0, \\ (\frac{x}{x_{max}})^\alpha, & \text{si } 0 < x < x_{max}, \\ 1, & \text{sinon.} \end{cases}$$

Le fait que $f(0) = 0$ permet de ne considérer que les coocurrences observées dans le corpus, ce qui réduit les calculs lorsque la taille du vocabulaire devient grande. Les hyper-paramètres utilisés dans l'article sont $x_{max} = 100$ et $\alpha = \frac{3}{4}$. Les embeddings w et les biais b sont initialisés aléatoirement.

Si l'on ignore f, on peut observer que minimiser J revient à minimiser le MSE entre $w_i^\top w_j + b_i + b_j$ et $\log X_{ij}$. C'est un problème de régression conventionnel.

Pour minimiser J à l'aide de pytorch, nous devons: 1. Créer des données X et Y représentant respectivement les paires de mots (i, j) et les valeurs X_{ij} associées. Le reste de J pourra être dérivé à partir de ces entrées 2. Créer une classe modèle `nn.Module` hébergeant les paramètres w et b sous forme de `nn.Parameter` 3. Créer une fonction qui calcule le loss J pour un batch échantilloné dans X, Y 4. Créer une boucle d'apprentissage classique qui minimise J par descente de gradient

Commençons par créer instancier les données sous forme de tenseurs. X est un tenseur de type `long` et de taille $(len(cooc), 2)$ qui contient les indices des mots cooccurrant (les clés du dictionnaire cooc). Y est un tenseur de type `float` et de taille $(len(cooc))$ contient les nombre de coocurrences associées (les valeurs du dictionnaire cooc).

[ ]:
```
#device = torch.device('cuda')

size = min(2000, len(vocab))

# input data: tensor with all pairs of words for which cooccurrences are␣
 ↪non-null
X = torch.tensor([(i, j) for i, j in cooc.keys() if i < size and j < size]).
 ↪long().to(device)
Y = torch.tensor([value for (i, j), v in cooc.items() if i < size and j <␣
 ↪size]).float().to(device)

# cast as batch loader
from torch.utils.data import TensorDataset, DataLoader
train_dataset = TensorDataset(X, Y)
train_loader = DataLoader(train_dataset, batch_size=1024, shuffle=True)
```

Le modèle `GloVe` contient deux tenseurs de paramètres : * `embedding`, de taille (nombre de mots, taille de l'embedding) * `bias`, de taille (nombre de mots) Tous deux sont encapsulés dans des `nn.Parameter` et initialisés avec une gaussienne de moyenne 0 et de variance 1.

Plutôt qu'une fonction `forward`, nous calculerons directement le loss J dans une fonction `compute_loss`. Cette fonction prend en entrée deux tenseurs x et y de taille respective (taile de batch, 2) et (taille de batch), contenant des valeurs extraites de X et Y. Cette fonction doit retourner J pour ces entrées, soit un tenseur de taille 1 (un scalaire).

Pour être rapide, cette fonction doit vectoriser les différentes opérations, et donc effectuer les calculs à l'aide d'opérations matricielles sur x et y. En particulier, il faut vectoriser l'opération $w_i^T w_j$, ce qui peut être effectué à l'aide de `torch.bmm`, la multiplication de matrices batchée.

[ ]:
```
class GloVe(nn.Module):
  def __init__(self):
    super().__init__()
    self.embedding = nn.Parameter(torch.randn((size, 50)))
    self.bias = nn.Parameter(torch.randn((size,)))

  # compute one round of loss
  def compute_loss(self, x, y):

    ## à compléter : calculer le loss de GloVe ##

    return glove_loss

model = GloVe().to(device)
model.compute_loss(X[:3], Y[:3])
```

L'apprentissage se fait à l'aide de l'optimiseur de votre choix, en effectuant plusieurs époques sur l'ensemble des données. Il faut bien veiller à voir toutes les données pour que tous les embeddings soient modifiés. Pour vérifier que l'apprentissage fonctionne, il faut que le loss diminue au fil des époques.

[ ]:
```
optimizer = optim.SGD(model.parameters(), lr=0.05)
for epoch in range(25):
  total_loss = 0
  num = 0
  for x, y in train_loader:
    optimizer.zero_grad()
    loss = model.compute_loss(x, y)
    loss.backward()
    optimizer.step()
    total_loss += loss.item()
    num += len(x)
  print(1 + epoch, total_loss / num)
```

À partir de ces embedding, on peut effectuer les mêmes opérations que sur les embeddings précédents et observer la différence de comportement.

[ ]:
```
# convert embeddings to numpy arrays
embedding = model.embedding.detach().cpu().numpy()

print('cosine(rues, ville) =', similarity(embedding, 'rues', 'ville'))
print('cosine(tour, monde) =', similarity(embedding, 'tour', 'monde'))
print(most_similar(embedding, 'monde'))
print(most_similar(embedding, 'pays'))
plot_projection(embedding)
```

Le manque de données peut générer des embeddings de qualité moyenne. Pour aller plus loin, téléchargez des embeddings préentraînés, puis vérifiez si les invarants linguistiques comme “roi - homme + femme = reine” sont valides en regardant le vecteur le plus proche de celui résultant du calcule $A - B + C$ pour trois mots d'intérêt. Construisez un petit corpus de test et calculez le nombre de fois où le résultat attendu pour un tel calcul est dans les 10 mots les plus proches.