TP 5.2 - Vision et language

In [ ]:

```
nom='jai'
prenom='ilyass'
```

Exercice 1 : Simplification du vocabulaire

In [ ]:

```
import pandas as pd
import _pickle as pickle
import matplotlib.pyplot as plt
import numpy as np

filename = 'flickr_8k_train_dataset.txt'
df = pd.read_csv(filename, delimiter='\t')
nb_samples = df.shape[0]
iter = df.iterrows()

bow = {}
nbwords = 0

for i in range(nb_samples):
 x = iter.__next__()
 cap_words = x[1][1].split() # split caption into words
 cap_wordsl = [w.lower() for w in cap_words] # remove capital letters
 nbwords += len(cap_wordsl)
 for w in cap_wordsl:
  if (w in bow):
   bow[w] = bow[w]+1
  else:
   bow[w] = 1

bown = sorted([(value,key) for (key,value) in bow.items()], reverse=True)
```

In [ ]:

```
nbkeep = 1000 # 100 is needed for fast processing
freqnc = np.cumsum([float(w[0])/nbwords*100.0 for w in bown])
print("number of kept words="+str(nbkeep)+" - ratio="+str(freqnc[nbkeep-1])+" %"
```

In [ ]:

```
x_axis = [str(bown[i][1]) for i in range(100)]
plt.figure(dpi=300)
plt.xticks(rotation=90, fontsize=3)
plt.ylabel('Word Frequency')
plt.bar(x_axis, freqnc[0:100])
```

In [4]:

```
nbkeep = 100 # 100 is needed for fast processing

outfile = 'Caption_Embeddings.p'
[listwords, embeddings] = pickle.load( open( outfile, "rb" ) )

embeddings_new = np.zeros((nbkeep,102))
listwords_new = []

for i in range(nbkeep):
 listwords_new.append(bown[i][1])
 embeddings_new[i,:] = embeddings[listwords.index(bown[i][1]), :]
 embeddings_new[i,:] /= np.linalg.norm(embeddings_new[i,:]) # Normalization


listwords = listwords_new
embeddings = embeddings_new
outfile = "Caption_Embeddings_"+str(nbkeep)+".p"
with open(outfile, "wb" ) as pickle_f:
 pickle.dump( [listwords, embeddings], pickle_f)
```

Simplification du vocabulaire

La simplification du vocabulaire est le processus de réduction de la taille et de la complexité d'un vocabulaire, souvent en supprimant des mots rares ou peu utilisés et en les remplaçant par des alternatives plus simples et fréquemment utilisées. Cela peut améliorer la lisibilité, la compréhension et l'accessibilité globale d'un langage écrit ou parlé.

Dans cette première partie du TP, nous allons extraire un histogramme d'occurrences de mots à partir des légendes du sous-ensemble d'entraînement de Flickr8k. Pour accélérer le temps d'entraînement du modèle, nous allons utiliser un sous-ensemble du vocabulaire provenant de l'intégration textuelle du TP précédent.

Pour ce faire :

1. Nous avons d'abord chargé les mots des légendes du sous-ensemble d'entraînement de Flickr8k, puis les avons triés selon leur fréquence d'occurrence.
2. Ensuite, nous avons chargé le fichier d'intégration du TP précédent et conservé les 1000 mots les plus fréquents. Nous avons ensuite sauvegardé ce sous-ensemble de mots ainsi que les vecteurs d'intégration Glove correspondants.
3. Enfin, nous avons calculé la fréquence cumulée des 100 premiers mots conservés.

Exercice 2 : Création des données d'apprentissage et de test

In [ ]:

```
filename = 'flickr_8k_train_dataset.txt'
df = pd.read_csv(filename, delimiter='\t')
nbTrain = df.shape[0]
iter_w = df.iterrows()


# Legends
caps = []

# Images
imgs = []
for i in range(nbTrain):
    x = iter_w.__next__()
    caps.append(x[1][1])
    imgs.append(x[1][0])

maxLCap = 0

for caption in caps:
    l = 0
    words_in_caption = caption.split()
    for j in range(len(words_in_caption) - 1):
        current_w = words_in_caption[j].lower()
        if current_w in listwords:
            l += 1
        if l > maxLCap:
            maxLCap = l

print("max caption length =" + str(maxLCap))
```

In [10]:

```
from requests import get  # to make GET request

def download(url, file_name):
    # open in binary mode
    with open(file_name, "wb") as file:
        # get request
        response = get(url)
        # write to file
        file.write(response.content)
```

In [ ]:

```
# Load features
download('http://cedric.cnam.fr/~thomen/cours/US330X/encoded_images_PCA.p', "enc
encoded_images = pickle.load(open("encoded_images_PCA.p", "rb"))

indexwords = {}
for i in range(len(listwords)):
    indexwords[listwords[i]] = i

tinput = 202

tVocabulary = len(listwords)

X_train = np.zeros((nbTrain, maxLCap, tinput))
Y_train = np.zeros((nbTrain, maxLCap, tVocabulary), bool)

ll = 50
nbtot = 0
nbkept = 0

for i in range(nbTrain):
    words_in_caption = caps[i].split()

    nbtot += len(words_in_caption) - 1
    indseq = 0
    for j in range(len(words_in_caption) - 1):

        current_w = words_in_caption[j].lower()

        if j == 0 and current_w != '<start>':
            print("PROBLEM")
        if current_w in listwords:
            X_train[i, indseq, 0:100] = encoded_images[imgs[i]]
            X_train[i, indseq, 100:202] = embeddings[listwords.index(current_w),

        next_w = words_in_caption[j + 1].lower()

        index_pred = 0
        if next_w in listwords:
            nbkept += 1
            index_pred = indexwords[next_w]
            Y_train[i, indseq, index_pred] = True

            indseq += 1

outfile = 'Training_data_' + str(nbkeep)


np.savez(outfile, X_train=X_train, Y_train=Y_train)
```

In [ ]:

```
# Test data
download('http://cedric.cnam.fr/~thomen/cours/US330X/flickr_8k_test_dataset.txt'
filename = 'flickr_8k_test_dataset.txt'
df = pd.read_csv(filename, delimiter='\t')
nbTest = df.shape[0]
iter_w = df.iterrows()


# Legends
caps = []
# Images
imgs = []
for i in range(nbTest):
    x = iter_w.__next__()
    caps.append(x[1][1])
    imgs.append(x[1][0])

indexwords = {}
for i in range(len(listwords)):
    indexwords[listwords[i]] = i

# Features
encoded_images = pickle.load(open("encoded_images_PCA.p", "rb"))

tVocabulary = len(listwords)
X_test = np.zeros((nbTest, maxLCap, tinput))
Y_test = np.zeros((nbTest, maxLCap, tVocabulary), bool)

for i in range(nbTest):
    words_in_caption = caps[i].split()
    indseq = 0
    for j in range(len(words_in_caption) - 1):
        current_w = words_in_caption[j].lower()
        if current_w in listwords:
            X_test[i, indseq, 0:100] = encoded_images[imgs[i]]
            X_test[i, indseq, 100:202] = embeddings[listwords.index(current_w),

        next_w = words_in_caption[j + 1].lower()
        if next_w in listwords:
            index_pred = indexwords[next_w]
            Y_test[i, indseq, index_pred] = True
            indseq += 1

outfile = 'Test_data_' + str(nbkeep)
np.savez(outfile, X_test=X_test, Y_test=Y_test)
```

Stockage des données d’entraînement

Nous allons stocker nos données d’entraînement, c’est-à-dire les tenseurs contenant les données et les étiquettes. Le tenseur de données **X** aura une taille de **Ns×Ls×d**, où :

- **Ns** représente le nombre de séquences (légendes),
- **Ls** est la longueur des séquences,
- **d** est la taille du vecteur décrivant chaque mot de la séquence.

Nous avons construit les tenseurs de données et d’étiquettes pour les données d’entraînement ainsi que pour les données qui seront utilisées pour tester notre modèle.

Exercice 3 : Entraînement du modèle

In [19]:

```
class DataGenerator:
    def __init__(self, filename, batch_size):
        self.filename = filename
        self.batch_size = batch_size

    def generator(self):
        while True:
            data = np.load(self.filename)
            total_samples = len(data['X_train'])
            indexes = np.arange(total_samples)
            np.random.shuffle(indexes)

            for start in range(0, total_samples, self.batch_size):
                end = min(start + self.batch_size, total_samples)
                batch_indexes = indexes[start:end]
                X_batch = data['X_train'][batch_indexes]
                y_batch = data['Y_train'][batch_indexes]
                yield X_batch, y_batch

# Utilisation :
train_generator = DataGenerator('Training_data_100.npz', 50)
```

In [ ]:

```
# Importation des bibliothèques
from keras.layers import SimpleRNN
from keras.models import Sequential
from keras.layers import Dense, Activation, Masking
from keras.optimizers import Adam
from keras.models import model_from_json

# Creation of the model
SEQLEN = 35
taille_chars = 202
HSIZE = 100
model = Sequential()
model.add(Masking(mask_value=0.0, input_shape=(SEQLEN, taille_chars)))
model.add(SimpleRNN(HSIZE, return_sequences=True, input_shape=(SEQLEN, taille_ch
model.add(Dense(1000, name='fc1'))
model.add(Activation("softmax"))
model.summary()
nbkeep = 1000

# Load train data
# Load train data TRAIN GENERATOR


# Load test data
Test_data = np.load('Test_data_' + str(nbkeep) + '.npz')
X_test = Test_data['X_test']
Y_test = Test_data['Y_test']

# Compiling
BATCH_SIZE = 10
NUM_EPOCHS = 10
optim = Adam()
model.compile(loss="categorical_crossentropy", optimizer=optim, metrics=['accura
model.summary()

# Train
model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS)

# Evaluation
scores_train = model.evaluate(X_train, Y_train, verbose=1)
scores_test = model.evaluate(X_test, Y_test, verbose=1)
print("PERFS TRAIN: %s: %.2f%%" % (model.metrics_names[1], scores_train[1] * 100
print("PERFS TEST: %s: %.2f%%" % (model.metrics_names[1], scores_test[1] * 100))
```

In [ ]:

```
def save_model(model, savename):
    model_json = model.to_json()
    with open(savename + ".json", "w") as yaml_file:
        yaml_file.write(model_json)
    print("json Model ", savename, ".json saved to disk")
    model.save_weights(savename + ".h5")
    print("Weights ", savename, ".h5 saved to disk")


def load_model(savename):
    with open(savename + ".json", "r") as yaml_file:
        model = model_from_json(yaml_file.read())
    print("json Model ", savename, ".json loaded ")
    model.load_weights(savename + ".h5")
    print("Weights ", savename, ".h5 loaded ")
    return model
```

In [ ]:

```
nameModel = 'vision'
save_model(model, nameModel)
```

EXERCICE 4

In [ ]:

```
def sampling(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    predsN = pow(preds, 1.0 / temperature)
    predsN /= np.sum(predsN)
    probas = np.random.multinomial(1, predsN, 1)
    return np.argmax(probas)
```

In [ ]:

In [ ]:

```
# Libraries
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import _pickle as pickle
import pandas as pd

from keras.optimizers import Adam


# Load Model
nameModel = 'vision'
model = load_model(nameModel)

# Compilation
optim = Adam()
model.compile(loss="categorical_crossentropy", optimizer=optim, metrics=['accura

# Load Test data
nbkeep = 1000
outfile = 'Test_data_' + str(nbkeep) + '.npz'
Test_data = np.load(outfile)
X_test = Test_data['X_test']
Y_test = Test_data['Y_test']

outfile = "Caption_Embeddings_" + str(nbkeep) + ".p"
[listwords, embeddings] = pickle.load(open(outfile, "rb"))
indexwords = {}
for i in range(len(listwords)):
    indexwords[listwords[i]] = i

# Display one image
ind = np.random.randint(X_test.shape[0])
filename = 'flickr_8k_test_dataset.txt'
df = pd.read_csv(filename, delimiter='\t')
iter_w = df.iterrows()

for i in range(ind + 1):
    x = iter_w.__next__()

imname = x[1][0]
print("image name=" + imname + " caption=" + x[1][1])
dirIm = "./Flicker8k_Dataset/"

img = mpimg.imread(dirIm + imname)
plt.figure(dpi=100)
plt.imshow(img)
plt.axis('off')
plt.show()

# Prediction
pred = model.predict(X_test[ind:ind + 1, :, :])


nbGen = 5
temperature = 0.1
for s in range(nbGen):
    wordpreds = "Caption n° " + str(s + 1) + ": "
    indpred = sampling(pred[0, 0, :], temperature)
    wordpred = listwords[indpred]
    wordpreds += str(wordpred) + " "
    X_test[ind:ind + 1, 1, 100:202] = embeddings[indpred]
    cpt = 1
    while str(wordpred) != '<end>' and cpt < 30:
        pred = model.predict(X_test[ind:ind + 1, :, :])
        indpred = sampling(pred[0, cpt, :], temperature)
        wordpred = listwords[indpred]
        wordpreds += str(wordpred) + " "
        cpt += 1
        X_test[ind:ind + 1, cpt, 100:202] = embeddings[indpred]

    print(wordpreds)
```

Dans cette partie, nous entraînons notre modèle. Pour cela, nous devons définir son architecture.
Le modèle sera de type séquentiel : il contiendra d'abord une couche de **masque** qui ne calculera pas l'erreur pour les positions où il n'y a pas de mot dans la séquence d'entrée. Ensuite, il inclura une couche de réseau récurrent constitué de **100 neurones** de type *SimpleRNN*. Enfin, nous aurons une couche entièrement connectée, suivie d'une **fonction d'activation softmax**.

La fonction de coût choisie est la **cross-entropy**, et nous utilisons l'optimiseur **Adam** en conservant son pas de gradient par défaut. Nous adoptons une **taille de batch** (batch size) de 10.

In [ ]:

```
from keras.optimizers import Adam
import _pickle as pickle
import pandas as pd
import nltk



nbkeep = 1000
outfile = 'Test_data_' + str(nbkeep) + '.npz'
npzfile = np.load(outfile)
X_test = npzfile['X_test']
Y_test = npzfile['Y_test']

nameModel = 'model_exo5'
model = load_model(nameModel)

optim = Adam()
model.compile(loss="categorical_crossentropy", optimizer=optim, metrics=['accura
scores_test = model.evaluate(X_test, Y_test, verbose=1)
print("PERFS TEST: %s: %.2f%%" % (model.metrics_names[1], scores_test[1] * 100))

outfile = "Caption_Embeddings_" + str(nbkeep) + ".p"
[listwords, embeddings] = pickle.load(open(outfile, "rb"))
indexwords = {}
for i in range(len(listwords)):
    indexwords[listwords[i]] = i

predictions = []
nbTest = X_test.shape[0]
for i in range(0, nbTest, 5):
    pred = model.predict(X_test[i:i + 1, :, :])
    wordpreds = []
    indpred = np.argmax(pred[0, 0, :])
    wordpred = listwords[indpred]
    wordpreds.append(str(wordpred))
    X_test[i, 1, 100:202] = embeddings[indpred]
    cpt = 1
    while str(wordpred) != '<end>' and cpt < (X_test.shape[1] - 1):
        pred = model.predict(X_test[i:i + 1, :, :])
        indpred = np.argmax(pred[0, cpt, :])
        wordpred = listwords[indpred]
        if wordpred != '<end>':
            wordpreds.append(str(wordpred))
            cpt += 1
        X_test[i, cpt, 100:202] = embeddings[indpred]

    if i % 1000 == 0:
        print("i=" + str(i) + " " + str(wordpreds))

    predictions.append(wordpreds)

references = []
filename = 'flickr_8k_test_dataset.txt'
df = pd.read_csv(filename, delimiter='\t')
iter_w = df.iterrows()

ccpt = 0
for i in range(nbTest // 5):
    captions_image = []
    for j in range(5):
        x = iter_w.__next__()
        ll = x[1][1].split()
        caption = []
        for k in range(1, len(ll) - 1):
            caption.append(ll[k])

    captions_image.append(caption)
    ccpt += 1

    references.append(captions_image)

# BLUE-1, BLUE-2, BLUE-3, BLUE-4
blue_scores = np.zeros(4)
weights = np.zeros((4, 4))
weights[0, 0] = 1
weights[1, 0] = 0.5
weights[1, 1] = 0.5
weights[2, 0] = 1.0 / 3.0
weights[2, 1] = 1.0 / 3.0
weights[2, 2] = 1.0 / 3.0
weights[3, :] = 1.0 / 4.0

for i in range(4):
    blue_scores[i] = nltk.translate.bleu_score.corpus_bleu(references, predictic
        weights[i, 0], weights[i, 1], weights[i, 2], weights[i, 3]))
    print("blue_score - " + str(i) + "=" + str(blue_scores[i]))
```