Lab Deep Learning/ Recurrent Neural Networks/ in keras

Using Many-to-One for movie rating predicton

Author: geoffroy.peeters@telecom-paris.fr

Version: 2021/10/05 (changed to tensorfow.keras)

For any remark or suggestion, please feel free to contact me.

Objective:

We will implement two different networks to perform automatic rating (0 or 1) of a movie given the text of its review. We will use the `imdb` (internet movie database) dataset.

The reviews are already available in the form of indexes that point to a word dictionary: each word is already encoded as an index in the dictionary.

Import packages

In [1]:
```python
import numpy as np
from tensorflow.keras.datasets import imdb

from tensorflow.keras import backend as K
from tensorflow.keras import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.layers import Dense, Activation, Embedding, Dropout, Input, LSTM, Reshape, Lambda, RepeatVector

colab = True
student = True
```

Parameters of the model

- We only consider the `top_words` first words in the word dictionary
- We truncate/zero-pad each sequence a length `max_review_length`

In [2]:
```python
top_words = 5000
max_review_length = 100
INDEX_FROM = 3
embedding_vector_length = 32
```

Import IMDB data

In [3]:
```python
# --- Import the IMDB data and only consider the ``top_words``` most used words
np.load.__defaults__=(None, True, True, 'ASCII')
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words, index_from=INDEX_FROM)
np.load.__defaults__=(None, False, True, 'ASCII')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

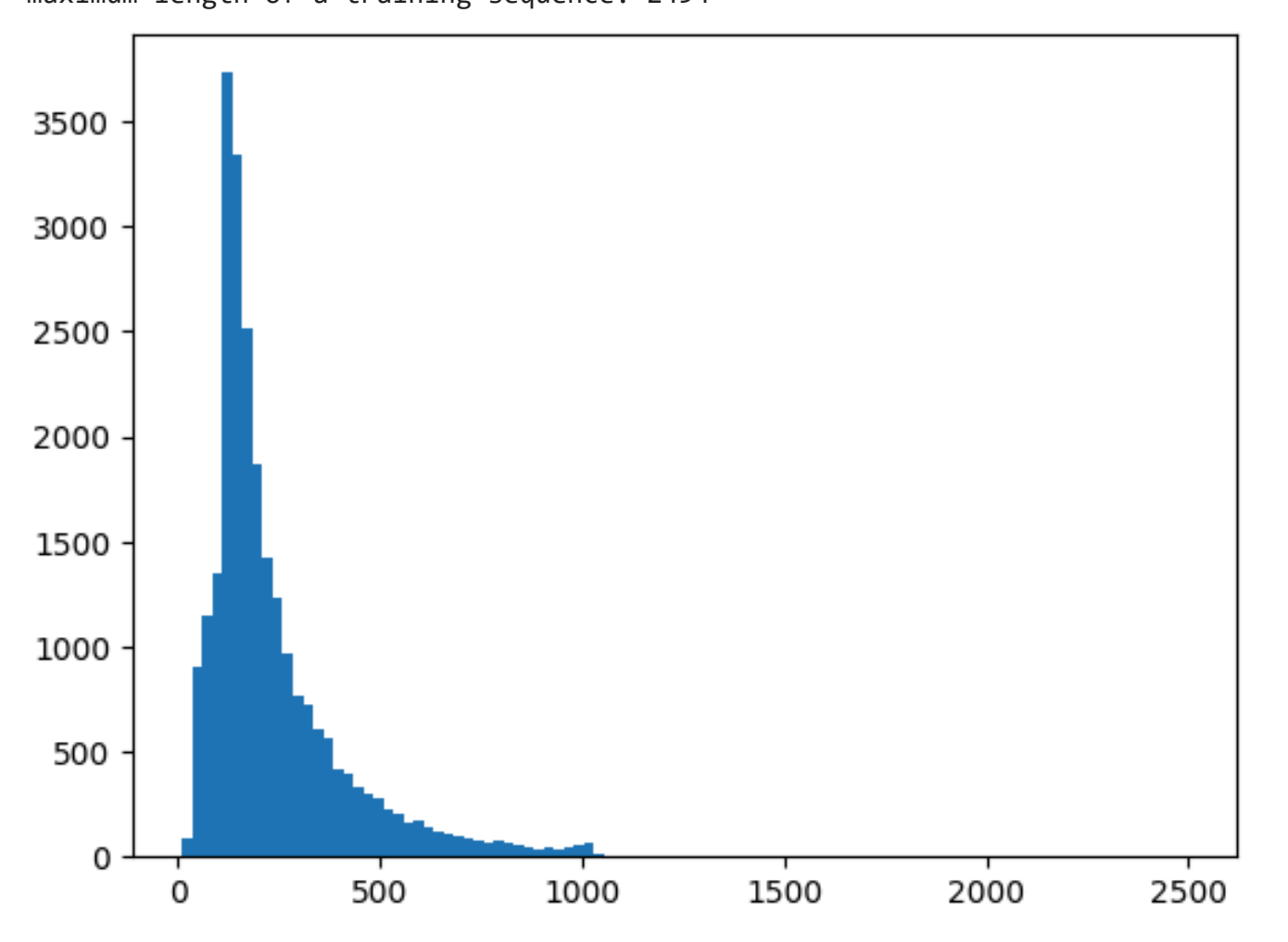
Data content

- `X_train` and `X_test` are numpy arrays of lists.
  - each item in a list is the index in the word dictionary. So that a list is the sequence of index of words.
- `y_train` and `y_test` are a numpy arrays of the same dimension as `X_train` and `X_test`
  - they contains the values 0 (bad movie) or 1 (good movie)

In [4]:
```python
print("type(X_train):", type(X_train))
print("number of training sequences: X_train.shape:", X_train.shape)
print("type(X_train[0]):",type(X_train[0]))
print("length of the first training sequence: len(X_train[0]):",len(X_train[0]))
print("length of the second training sequence: len(X_train[1]):",len(X_train[1]))
print("list of data of the first training sequence: X_train[0]:", X_train[0] )
len_list = [len(train) for train in X_train]
print("maximum length of a training sequence:", max(len_list))

import matplotlib.pyplot as plt
plt.hist(len_list, 100);
```

```
type(X_train): <class 'numpy.ndarray'>
number of training sequences: X_train.shape: (25000,)
type(X_train[0]): <class 'list'>
length of the first training sequence: len(X_train[0]): 218
length of the second training sequence: len(X_train[1]): 189
list of data of the first training sequence: X_train[0]: [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 48
0, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 3
8, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 2, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25,
1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 2, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 1
5, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 2, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 2, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472,
113, 103, 32, 15, 16, 2, 19, 178, 32]
maximum length of a training sequence: 2494
```



Details of how the reviews are encoded

In [5]:
```python
word_to_id = imdb.get_word_index()
word_to_id = {k:(v+INDEX_FROM) for k,v in word_to_id.items()}
word_to_id["<PAD>"] = 0
word_to_id["<START>"] = 1
word_to_id["<UNK>"] = 2

id_to_word = {value:key for key,value in word_to_id.items()}
print(' '.join(id_to_word[id] for id in X_train[1000] ))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
<START> although i had seen <UNK> in a theater way back in <UNK> i couldn't remember anything of the plot except for vague images of kurt thomas running and fighting against a backdrop of s
tone walls and disappointment regarding the ending br br after reading some of the other reviews i picked up a copy of the newly released dvd to once again enter the world of <UNK> br br it
turns out this is one of those films produced during the <UNK> that would go directly to video today the film stars <UNK> <UNK> kurt thomas as jonathan <UNK> <UNK> out of the blue to <UNK>
the nation of <UNK> to enter and hopefully win the game a <UNK> <UNK> <UNK> by the khan who <UNK> his people by yelling what sounds like <UNK> power the goal of the mission involves the sta
r wars defense system jonathan is trained in the martial arts by princess <UNK> who never speaks or leaves the house once trained tries to blend in with the <UNK> by wearing a bright red <U
NK> with <UNK> of blue and white needless to say <UNK> finds himself running and fighting for his life along the stone streets of <UNK> on his way to a date with destiny and the game br br
star kurt thomas was ill served by director robert <UNK> who it looks like was never on the set the so called script is just this side of incompetent see other reviews for the many <UNK> th
roughout the town of <UNK> has a few good moments but is ultimately ruined by bad editing the ending <UNK> still there's the <UNK> of a good action adventure here a hong kong version with m
ore <UNK> action and faster pace might even be pretty good
```

In [6]:
```python
print("type(y_train):", type(y_train))
print("y_train.shape:", y_train.shape)
```

```
type(y_train): <class 'numpy.ndarray'>
y_train.shape: (25000,)
```

In [7]:
```python
print("X_test.shape:", X_test.shape)
print("y_test.shape:", y_test.shape)
```

```
X_test.shape: (25000,)
y_test.shape: (25000,)
```

Data processing

Sequences (represented as a list of values) in `X_train` represent the reviews. They can have different length. To train the network we should modify them so that they all have the same length. We do this by:

- truncating the ones that are too long
- padding-with-zero them the ones that are too short.

This is obtained using `sequence.pad_sequences` of keras.

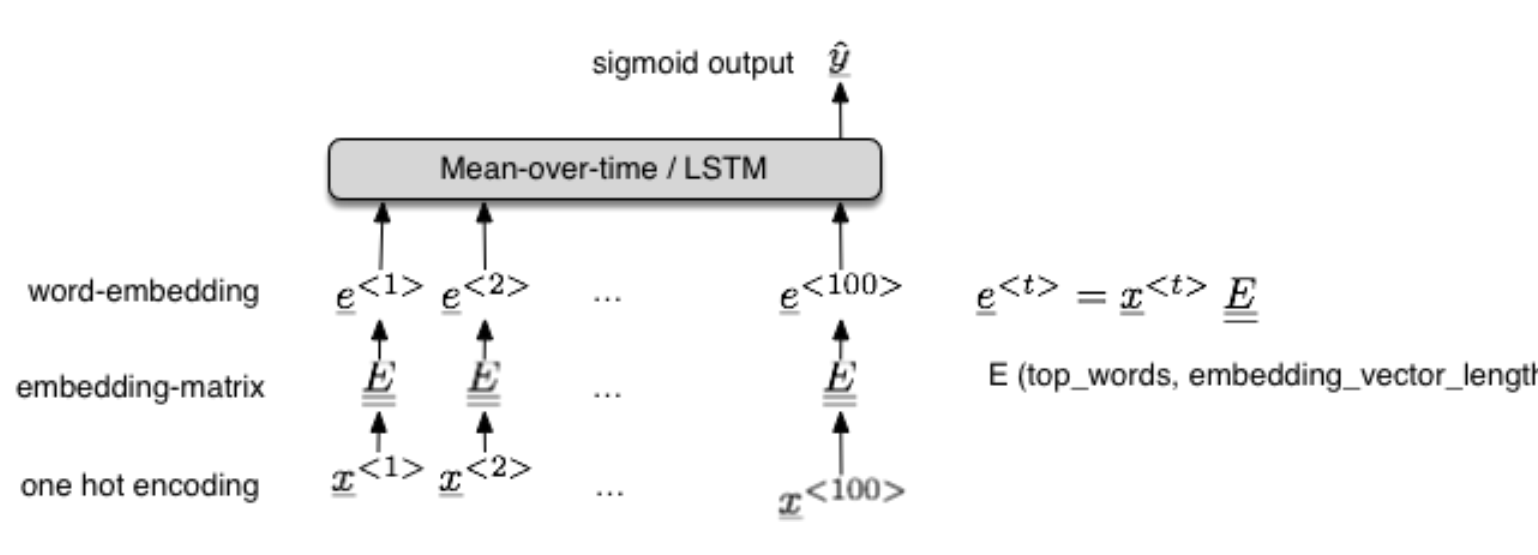
In [14]:
```python
# --- truncate and pad input sequences

if student:
    # --- START CODE HERE (01)
    X_train = sequence.pad_sequences(X_train, maxlen= max_review_length)
    X_test = sequence.pad_sequences(X_test, maxlen= max_review_length)
    # --- END CODE HERE

print("len(X_train[0]):", len(X_train[0]))
print("len(X_train[1]):", len(X_train[1]))
print("X_train[0]:", X_train[0])
```

```
len(X_train[0]): 100
len(X_train[1]): 100
X_train[0]: [1415   33    6   22   12  215   28   77   52    5   14  407   16   82
    2    8    4  107  117    2   15  256    4    2    7 3766    5  723
   36   71   43  530  476   26  400  317   46    7    4    2 1029   13
  104   88    4  381   15  297   98   32 2071   56   26  141    6  194
    2   18    4  226   22   21  134  476   26  480    5  144   30    2
   18   51   36   28  224   92   25  104    4  226   65   16   38 1334
   88   12   16  283    5   16 4472  113  103   32   15   16    2   19
  178   32]
```

First model



In the first model, we will simply

- learn a word embedding (`Embedding` layer in keras) and apply it to each item of the sequence,
  - in keras, embedding is not a matrix going from one-hot-encoding to embedding, but is a layer that goes from index-in-word-dictionary to embedding
  - the embedding goes from `top_words` dimensions to `embedding_vector_length` dimensions
- average the embedding obtained for each word of a sequence over all words of the sequence (you should use `K.mean` and `Lambda` from the keras backend)
- apply a fully connected (`Dense` layer in keras) which output activation is a sigmoid (predicting the 0 or 1 rating)

We will code this model

- First, using the Sequential API of keras (https://keras.io/models/sequential/)
- Secondly, using the Functional API of keras (https://keras.io/getting-started/functional-api-guide/)

In [15]:
```python
K.clear_session()
```

In [16]:
```python
X_train[0]
```

Out[16]:
```
array([1415,   33,    6,   22,   12,  215,   28,   77,   52,    5,   14,
        407,   16,   82,    2,    8,    4,  107,  117,    2,   15,  256,
          4,    2,    7, 3766,    5,  723,   36,   71,   43,  530,  476,
         26,  400,  317,   46,    7,    4,    2, 1029,   13,  104,   88,
          4,  381,   15,  297,   98,   32, 2071,   56,   26,  141,    6,
        194,    2,   18,    4,  226,   22,   21,  134,  476,   26,  480,
          5,  144,   30,    2,   18,   51,   36,   28,  224,   92,   25,
        104,    4,  226,   65,   16,   38, 1334,   88,   12,   16,  283,
          5,   16, 4472,  113,  103,   32,   15,   16,    2,   19,  178,
         32], dtype=int32)
```

In [17]:
```python
# --- create the model
# CODE-RNN1-2
if student:
    # --- START CODE HERE (02)
    # --- Using the Sequential API
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length, input_length=max_review_length))
    model.add(Lambda(lambda x: K.mean(x, axis=1)))
    model.add(Dense(1, activation='sigmoid'))
    # --- END CODE HERE

    # --- START CODE HERE (03)
    # --- Using the Functional API
    inputs = Input(shape=(max_review_length,))
    embedding = Embedding(top_words, embedding_vector_length)(inputs)
    mean_embedding = Lambda(lambda x: K.mean(x, axis=1))(embedding)
    outputs = Dense(1, activation='sigmoid')(mean_embedding)
    model_functional = Model(inputs=inputs, outputs=outputs)
    # --- END CODE HERE

print(model.summary())
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lambda (Lambda)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

None

In [18]:
```python
# --- compile and fit the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=3, batch_size=64, validation_data=(X_test, y_test))
```

```
Epoch 1/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.6551 - loss: 0.6647 - val_accuracy: 0.7869 - val_loss: 0.5411
Epoch 2/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8063 - loss: 0.4987 - val_accuracy: 0.8190 - val_loss: 0.4263
Epoch 3/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.8357 - loss: 0.4007 - val_accuracy: 0.8332 - val_loss: 0.3809
```

Out[18]:
```
<keras.src.callbacks.history.History at 0x7db1cb36ba40>
```

Results

After only 3 epochs, you should obtain an accuracy around 83-84% for the test data.

In [19]:
```python
# --- Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 83.32%
```

Using the trained embedding to find equivalence between words

Since the embedding is part of the models, we can look at the trained embedding matrix E and use it to get the most similar words (according to the trained matrix E) in the dictionary. Use the weights of the `Embedding` layer to find the most similar words to `great`. We will use an Euclidean distance for that.

- Retrieve the weights of the `Embedding layer`
- Get the position of `great` in the dictionary
- Get the word-embedding of `great`
- Find (using Euclidean distance), the closest embedded-words to `great`

In [ ]:
```python
model.layers[0].get_weights()[0].shape
```

In [20]:
```python
if student:
    # --- START CODE HERE (04)
    embedding_weights = model.layers[0].get_weights()[0]

    great_word = "great"
    great_id = word_to_id[great_word]

    great_embedding = embedding_weights[great_id]

    dist_v = np.sqrt(np.sum((embedding_weights - great_embedding)**2, axis=1))
    # --- END CODE HERE
for i in np.argsort(dist_v)[0:20]: print(id_to_word[i])
```

```
great
excellent
perfect
wonderful
7
highly
8
amazing
best
love
favorite
superb
today
9
brilliant
fantastic
well
loved
recommended
enjoyed
```

Second model

In the second model, we will replace

- the average over the sequence of the obtained embedding
- by a RNN layer (more precisely an `LSTM`) in a Many-To-One configuration with $n_a = 128$

We will code this model

- First, using the Sequential API of keras (https://keras.io/models/sequential/)
- Secondly, using the Functional API of keras (https://keras.io/getting-started/functional-api-guide/)

In [21]:
```python
K.clear_session()
```

In [22]:
```python
# --- create the model

if student:
    # --- START CODE HERE (05)
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length, input_length=max_review_length))
    model.add(LSTM(128))
    model.add(Dense(1, activation='sigmoid'))
    # --- END CODE HERE

    # --- START CODE HERE (06)
    inputs = Input(shape=(max_review_length,))
    embedding = Embedding(top_words, embedding_vector_length)(inputs)
    lstm_out = LSTM(128)(embedding)
    outputs = Dense(1, activation='sigmoid')(lstm_out)
    model_functional = Model(inputs=inputs, outputs=outputs)
    # --- END CODE HERE

print(model.summary())
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

None

In [23]:
```python
# --- compile and fit the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=3, batch_size=64, validation_data=(X_test, y_test))
```

```
Epoch 1/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.7007 - loss: 0.5475 - val_accuracy: 0.8358 - val_loss: 0.3766
Epoch 2/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.8722 - loss: 0.3118 - val_accuracy: 0.8212 - val_loss: 0.4063
Epoch 3/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.8899 - loss: 0.2744 - val_accuracy: 0.8462 - val_loss: 0.3684
```

Out[23]:
```
<keras.src.callbacks.history.History at 0x7db1d289e8a0>
```

Results

After only 3 epochs, you should obtain an accuracy around 84-85% for the test data.

In [ ]:
```python
# --- Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Evaluation

To evaluate the work, you should rate the code for

- 1. Data Pre-Processing (01)
- 2. First model using the Sequential API (02)
- 3. First model using the Functional API (03)
- 4. Find equivalence between words (04)
- 5. Second model using the Sequential API (05)
- 6. Second model using the Functional API (06)